New Regular Radix-8 Scheme for Elliptic Curve Scalar Multiplication without Pre-Computation

Ebrahim A. H. Abdulrahman and Arash Reyhani-Masoleh, *Member*, *IEEE*

Abstract—The recent advances in mobile technologies have increased the demand for high performance parallel computing schemes. In this paper, we present a new algorithm for evaluating elliptic curve scalar multiplication that can be used on any abelian group. We show that the properties of the proposed algorithm enhance parallelism at both the point arithmetic and the field arithmetic levels. Then, we employ this algorithm in proposing a new hardware design for the implementation of an elliptic curve scalar multiplication on a prime extended twisted Edwards curve incorporating eight parallel operations. We further show that in comparison to the other simple side-channel attack protected schemes over prime fields, the proposed design of the extended twisted Edwards curve is the fastest scalar multiplication scheme reported in the literature.

Index Terms—Elliptic curve, scalar multiplication, side-channel attack, parallel computing schemes

1 Introduction

In 1976, Diffie and Hellman introduced the idea of public key cryptography (PKC) [1]. PKC is now widely used for key establishment, digital signature, data encryption, and other applications. Since then, several public-key cryptosystems have been proposed; the security in these systems is based on the difficulty of the mathematical problem [2], [3]. Although today commonly used PKC algorithms such as RSA [4], and ElGamal [5] are believed to be secure, some of their implementations have been challenged by the quick factoring and integer discrete logarithm attacks [6]–[8]. Elliptic curve cryptography (ECC) [9], [10] that can provide the same level of security with a shorter key size becomes more attractive in applications with embedded microprocessors [11]. While the ECC provides shorter key sizes, the required computational complexity may still be excessive. By exploiting parallelization in the design, a system will be able to reduce the computation time, and the energy expenditure will be minimized [12].

ECC algorithms belong to the class of group-based protocols, whose security is based on the difficulty of the discrete logarithm problem over a finite group. Using additive notation, this problem can be described as follows. Given points P and Q in the group, finding a number k such that $Q = kP$ is assumed to be not feasible in polynomial time [13]. The operation of computing the new point, i.e., kP, is called the elliptic curve scalar (or point) multiplication (ECSM) operation, which is the core building block in ECC [14]. ECSM computes a scalar point kP by performing multiple point additions, based on an s-bit scalar k, where $s = \lceil \log_2 k \rceil$, and a point P that is on an elliptic curve. This operation is achieved with the execution of iterated point addition (ADD) and point doubling (DBL), which involve the finite-field (or modular) arithmetic operations over either $GF(p)$ or $GF(2^m)$.

To efficiently compute the scalar multiplication, there are three main approaches. The first approach is to utilize efficient point arithmetic operation formulas based on a combination of the underlying finite-field operations. For instance, implementing point halving instead of the DBL operation over binary fields [15], point tripling over fields of characteristic three [16], [17], and using composite operations, i.e., $2Q + P$ [18]. The second approach is to use a representation of the scalar such that the number of point arithmetic operations is reduced. Nonadjacent form (NAF) [19], radix-r NAF (r-NAF) [20], width-w NAF (w-NAF) [21], [22], [8], and Frobenius map [21], [23] are some techniques based on this approach. The third approach is to use more hardware support, i.e., utilizing memory for pre-computation, and/or parallel operations [24]–[29], and/or pipelining methods [30], [31]. In this paper, we combine the first two approaches with the parallel computation in the third approach to yield a very efficient scalar multiplication scheme. The main contributions of this paper can be summarized as follows:

- We propose an approach to computing the ECSM operation that is based on processing three bits of the scalar in the exact same sequence of five point arithmetic operations, namely, 3 DBLs, and 2 ADDs for all eight different combinations of 3 bits without using any dummy operations. The scalar k and the point P in the proposed method are considered to be generic, and no memory lookup-table for pre-computed points is required.
- We analyse the security of our scheme and show that its security holds against both simple side-channel (or power analysis) attacks (SSCAs) [32], [33], and safe-error (or C-safe) fault attacks [34], [35].
- Finally, we show how the properties of the proposed ECSM scheme yields an efficient hardware design for the

• *The authors are with the Department of Electrical and Computer Engineering, Western University, London, Ontario, Canada.*
E-mail: {eabdulra, areyhani}@uwo.ca.

Manuscript received 10 Apr. 2013; revised 27 Sep. 2013; accepted 16 Oct. 2013. Date of publication 23 Oct. 2013; date of current version 16 Jan. 2015.
Recommended for acceptance by J. Bruguera.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2013.213

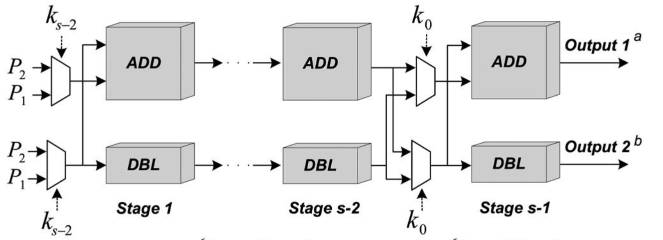
implementation of a single ECSM on a prime extended twisted Edwards curve incorporating 8 parallel multiplication operations. We show that this design is the fastest SSCA-protected scalar multiplication scheme over prime fields reported in the literature including the fast *x-coordinates only* method of the Montgomery Ladder on the Montgomery curves [36] for the parallel environment.

The organization of this paper is as follows. In the next section, preliminaries related to the SSCA-protected ECSM schemes are presented. In Section 3, the formula for a new radix-r method for evaluating the scalar multiplication is introduced. Then, the generalised radix-r algorithm is specified for the radix-8 one. Section 4 is the core of our paper, in which, a novel ECSM scheme that offers resistance against both SSCA and safe-error fault attacks is presented. Then, to illustrate the advantages of the proposed scheme, in Section 5, we evaluate and analyse the efficiency of the proposed ECSM scheme and compare it to the other well known ECSM schemes at the elliptic curve group operations level. Section 6, explains how a protected scalar multiplication using the proposed scheme for the prime extended twisted Edwards model can be performed faster than all the other parallel and SSCA-protected schemes reported in the literature. Finally, the conclusion is summarized in Section 7.

2 Preliminaries

The classical method for evaluating kP is the so-called Double-and-Add binary method [37]. On average, the computation complexity of the Double-and-Add binary method is $s-1$ DBLs, and $\frac{s-1}{2}$ ADDs [38]. In order to lower the number of ADDs, the scalar k is converted to a signed-representation. Let each bit of k be denoted by k_i, for $0 \leq i \leq s-1$. Then k_i in signed-representation becomes $k_i \in \{-1, 0, 1\}$. The signed-representation revises the Double-and-Add binary method to a new method called the signed-binary (or addition-subtraction) method [39], [19], [37]. Among the different signed-representation methods, the non-adjacent form (NAF) [14], [19], [20] and the mutual opposite form (MOF) [40] are the most popular methods. The computation of ECSM in the signed binary methods is more effective than in the Double-and-Add binary method. Representing the scalar k as NAF or MOF would save an average of 1/6 of ADDs in the computation of kP [38], [8]. The total run time of the ADD in both the Double-and-Add binary method and the signed binary methods depend on the Hamming-weight of the scalar k. Hence, an adversary observing the run time, could determine the Hamming-weight of the secret k.

From a mathematical point of view, ECC is regarded as being secure. However, real-world hardware implementations of ECC protocols may introduce leakage, which raises the issue of other threats that may not be addressed by the crypto-algorithms, e.g., the elapsed time or the power consumption that depends on analysing the VLSI implementation of the crypto-algorithm. Thus, an unsecured implementation can lead to the exposure of the secret key by utilizing attack techniques that analyse such information. Kocher in [32] reviewed these kinds of attacks and referred to them as side-channel attacks (SCAs). Of all the types of SCAs, the SSCAs are the common. In ECC cryptosystems, SSCA can reveal large features of the algorithm such as identifying the DBL and the ADD operations being executed in the iterations of the loop [33]. Thus, ECSM should be implemented using a specific sequence of point arithmetic operations that does not depend on the value of a particular scalar bit.



^{a} output 1 $= \begin{cases} P_2, & \text{if } k_i = 0 \\ P_1, & \text{if } k_i = 1 \end{cases}$ $\quad$ ^{b} output 2 $= \begin{cases} P_1, & \text{if } k_i = 0 \\ P_2, & \text{if } k_i = 1 \end{cases}$

Fig. 1. EC-operations dependency graph for the Montgomery Ladder ECSM method [44], [45], [46], which shows that only the operands are transposed.

2.1 Notations

In this work, we refer to the elliptic curve group (arithmetic point) operations as EC-operations. Also, ADD, and DBL stand for the EC-operations of addition, and doubling, respectively. Similarly, the EC-operation of subtraction is denoted by SUB in this paper. Also, the ADDDBL operation stands for considering both the ADD and the DBL operations as a single composite operation. In addition, mADD, and uADD stand for the cost of mixed addition, and unified addition, respectively. Computing the cost of field arithmetic operations is represented by capital bold-faced characters; hence, $\mathbf{I}$, $\mathbf{M}$, $\mathbf{S}$, $\mathbf{A}$, and $\mathbf{D}$ stand for the computing costs of field Inversion, multiplication, squaring, addition, and field multiplication by a curve constant, respectively.

2.2 The SSCA-Protected ECSMs

When both the ADD and the DBL operations are different, the only way to make an ECSM algorithm SSCA aware is to use a regular structure scalar multiplication scheme, which evaluates the point arithmetic operations in a uniform sequence. The author in [33] has masked the dependency between the scalar bit and the evaluated point arithmetic operation by inserting a dummy operation. However, it is noted in [41] and [42] that it may be easy for the adversaries to determine which point arithmetic ADDs are the dummy operations. A method proposed by Möller in [43] performs the scalar multiplication with a fixed pattern of point arithmetic DBLs and ADDs, Okeya et al. in [22] have also proposed a similar window-based method. The Montgomery Ladder binary method [36], [44]–[46] is especially suitable for hardware implementation because of the data independency of its underlying point arithmetic operations, and the resistance to SSCA. Fig. 1 shows how Montgomery's scalar multiplication method operates at the point arithmetic level. It can be seen that although there is a conditional statement at the beginning of each stage, which is represented by multiplexers, Montgomery's method is still considered to be a highly regular method as both the ADD and the DBL operations are repeatedly evaluated together at each iteration of the main loop. Joye in [47] has also developed a similar binary scalar multiplication method that eliminates power analysis information.

In this work, we present a new regular ECSM scheme. We show that we save $1/3$ of the computation of the ADD operations as compared to the regular binary schemes presented in [44]–[47]. We also show that at least 40% of the memory registers are less compared to the secured window-based schemes shown in [43] and [22]. Further, if the computational time complexity of 2 ADDs is less than the computational time complexity of $2\ \mathrm{DBLs} + \mathrm{mADD}$, the speed of the proposed scheme outperforms those of secured window-based schemes.

3 Proposed Radix-8 Scalar Multiplication Algorithm

Throughout this section, we present a method for evaluating the scalar multiplication in radix-r. We then explain how the scalar k in the radix-8 can be recoded to a signed-representation in the range $[-1, 6]$ so that the scheme we propose in the next section can thwart SSCAs.

3.1 High-Radix Scalar Expansion

It is assumed hereafter that the basis r has been chosen to be a power of 2, i.e., $r = 2^w$, where $2 \leq w \leq s-1$. Hence, the computation of rP requires only repeated DBLs. Let the scalar k (of length s-bits) be partitioned into l digits, i.e., $l = \lceil \frac{s}{w} \rceil$, and let each digit of k be denoted as k'_i for $0 \leq i \leq l-1$. The scalar k with radix-r expansion $(k'_{l-1}, \dots, k'_1, k'_0)_r$, where $k'_i \in \{0, 1, \dots, r-1\}$ for every $i \leq l-1$, can be presented as

$$k = \sum_{i=0}^{l-1} k'_i r^i, k'_i \in \{0, 1, \dots, r-1\}. \tag{1}$$

Scalar multiplication kP can then be computed as

$$kP = \sum_{i=0}^{l-1} (k'_i r^i) P. \tag{2}$$

In the following, we let $E\ (F_q)$ be an abelian group with an identity element $\mathcal{O}$, and we let $P \in E\ (F_q)$ be an input point element. Notice that our goal is to compute the scalar multiplication point kP that is also a point in $E\ (F_q)$, i.e., $kP \in E\ (F_q)$. Let P_{kP} and P_1 be two points on the curve, which are initialized by $\mathcal{O}$ and P, respectively. We define the point

$$P_{kP}^{(j)} = \sum_{i=0}^{j} (k'_i r^i) P, \tag{3}$$

for any $0 < j < l$.

Comparing (2) and (3), one can see that $kP = P_{kP}^{(l-1)}$. By removing the upper j-th term from the summation of (3), we get

$$P_{kP}^{(j)} = k'_j r^j P + P_{kP}^{(j-1)}. \tag{4}$$

Assuming

$$P_{Acc}^{(j)} = r^j P, \tag{5}$$

is another point on the curve that is initialized to P, i.e., $P_{Acc}^{(0)} = P$. Substituting this in (4), one can obtain $P_{kP}^{(j)}$ as

$$P_{kP}^{(j)} = k'_j P_{Acc}^{(j)} + P_{kP}^{(j-1)}. \tag{6}$$

Now, we define another recursive point on the curve

$$P_1^{(j)} = r^{j+1} P - P_{kP}^{(j)}. \tag{7}$$

In order to ensure the computation regularity for each specific input k'_j, the two recursive points $P_{kP}^{(j)}$, and $P_1^{(j)}$ have to be properly obtained by performing either the ADD or the SUB operations as presented below.

Lemma 1. *Consider j to be in the range $[1, l-1]$, and $0 \leq k'_j \leq r-1$, then $P_{kP}^{(j)}$ can be defined in one of the following two ways:*

$$P_{kP}^{(j)} = \begin{cases} P_{kP}^{(j-1)} + k'_j P_{Acc}^{(j)} \\ r P_{Acc}^{(j)} - P_1^{(j)}, \end{cases} \tag{8}$$

and $P_1^{(j)}$ can be obtained as follows

$$P_1^{(j)} = \begin{cases} r P_{Acc}^{(j)} - P_{kP}^{(j)} \\ (r - 1 - k'_j) P_{Acc}^{(j)} + P_1^{(j-1)}, \end{cases}$$

where $P_{Acc}^{(j)} = r^j P = r P_{Acc}^{(j-1)}$.

Proof. using (6) and (7), one can easily obtain (8). Changing j to $j-1$ and re-arranging the terms in (7), one can obtain $r^j P$ as

$$r^j P = P_1^{(j-1)} + P_{kP}^{(j-1)}. \tag{9}$$

Substituting $r^j P$ from (9) into (4), one can obtain $P_{kP}^{(j)}$ as

$$P_{kP}^{(j)} = k'_j P_1^{(j-1)} + (k'_j + 1) P_{kP}^{(j-1)}. \tag{10}$$

Substituting $P_{kP}^{(j)}$ from (10) into (7), one can obtain

$$P_1^{(j)} = r^{j+1} P - (k'_j P_1^{(j-1)} + (k'_j + 1) P_{kP}^{(j-1)}). \tag{11}$$

Substituting $P_{kP}^{(j-1)}$ from (9) into (11) and using (5), $P_1^{(j)}$ can be further obtained as

$$P_1^{(j)} = (r - (k'_j + 1)) P_{Acc}^{(j)} + P_1^{(j-1)}.$$

The proof is complete. □

3.2 Recoding the Scalar k into Signed Radix-8

In order to ensure that our scheme is entirely regular, we need to skip the digit k'_j that is equal to 7 and replace it with -1 with an increment to the next digit as $k'_{j+1} + 1$. Möller in [43] has described a recoding algorithm for m-array exponentiation where each digit that is equal to zero is replaced with $-m$, and the next most significant digit is incremented by one. In [48], the scalar digits are recoded in the set $\{1, \dots, m\}$, where each zero digit is replaced with m and the next digit is decremented by one. In our case, we replace the k'_j value that is equal to digit 7 with $(7 - 8 = -1)$. This representation was discussed by

Parhami in [49]. He used this representation in multiplication schemes that can handle more than one bit of the multiplier in each cycle. Intuitively, the recoding algorithm replaces the 7 digits by -1 and increments the next more significant digit to adjust the value. Let the scalar k of the length of s bits be given in the radix-8 digit representation, where k'_j is in the range $[0, 7]$. Algorithm 1 shows the steps to convert (1) for radix-8, i.e. $r = 8$, to the following non-seven representation

$$k = \sum_{i=0}^{t-1} k_j 8^i, k_j \in \{-1, 0, 1, \cdots, 6\}.$$

Algorithm 1 Proposed Non-Seven Encoding Method

```
Input: A $t-1$ digit Radix-8 of the scalar $k$,
    $k = (k'_{t-2}, \ldots, k'_1, k'_0)_8$, $k'_j \in \{0, 1, \ldots, 7\}$.
Output: $k = (k_{t-1}, \ldots, k_1, k_0)_8$, $k_j \in \{-1, 0, 1, \ldots, 6\}$.
Initialize: $k = (0, k'_{t-2}, \ldots, k'_1, k'_0)_8$;
Step 1: For $j = 0$ to $t-1$ do
    Step 1.1: If $k'_j \in \{7, 8\}$ Then
        Step 1.1.1: $k_j = k'_j - 8, k'_{j+1} = k'_{j+1} + 1$;
    Step 1.2: Else Leave the digit as it is, i.e., $k_j = k'_j$
Step 2: End For
Step 3: Return $k = (k_{t-1}, \ldots, k_1, k_0)_8$;
```

In the next subsection, we define a new radix-8 ECSM algorithm for a t-digit of k, where $t = \lceil \log_8 k \rceil + 1$, and $k_j \in [-1, 6]$, which, as will be shown in Section 4, yields to a regular ECSM scheme.

3.3 Proposed Radix-8 Algorithm for Scalar Multiplication

We perform the scalar multiplication with a new right-to-left radix-8 algorithm using the non-seven representation of k that is discussed in Section 3.2 and obtained in Algorithm 1. We notice that the evaluation of the scalar multiplication in the proposed radix-8 algorithm, is performed utilizing three EC-points, i.e., P_{kP}, P_1, and P_{Acc} without pre-computation.

One can extend Lemma 1 so that one can compute $P_{kP}^{(j)}$ for any $j > 0$, and $-1 \le k_j \le 6$, as follows

$$P_{kP}^{(j)} = \begin{cases} P_{kP}^{(j-1)} + k_j P_{Acc}^{(j)}, & \text{if } k_j \in \{-1, 0, 1, 2, 4\}, \\ 8P_{Acc}^{(j)} - P_1^{(j)}, & \text{if } k_j \in \{3, 5, 6\}. \end{cases} \quad (12)$$

Similarly, from the extension of Lemma 1, one can compute $P_1^{(j)}$ for any $j > 0$, and $-1 \le k_j \le 6$, as follows

$$P_1^{(j)} = \begin{cases} 8P_{Acc}^{(j)} - P_{kP}^{(j)}, & \text{if } k_j \in \{-1, 0, 1, 2, 4\}, \\ (7 - k_j)P_{Acc}^{(j)} + P_1^{(j-1)}, & \text{if } k_j \in \{3, 5, 6\}. \end{cases} \quad (13)$$

Note that the reason we have split the eight possible combinations of k_j in (12) into two cases is to have the k_j with a maximum of one Hamming-weight in one group list. Similarly, the reason we have split the eight possible combinations of of k_j in (13) into two cases is to have the $7 - k_j$ with a a maximum of one Hamming-weight in one group list. Based on (12) and (13), we propose Algorithm 2 in which the scalar k is obtained from the output of Algorithm 1. In Algorithm 2, it is shown that $8P_{Acc}$ is computed in each iteration, and the result of its computation is stored in a register known as P_{Acc} (see Steps 1.1.2, and 1.2.2). Hence, the value of point $P_{Acc}^{(j+1)} = 8P_{Acc}^{(j)}$ is evaluated in advance at the end of iteration j. The evaluation of kP involves a total of t computational iterations. At each iteration, the sum of the two points P_{kP} and P_1 are always equal to the value of point P_{Acc}. The final result of the kP is obtained at the last iteration, which is the content values of the register P_{kP} at the iteration $t-1$. It is noteworthy that both Algorithms 1, and 2 are evaluated from right to left; hence, they can be interleaved resulting in a significant memory register reduction, because it eliminates the need to store both the scalar and its recoding.

Algorithm 2 Proposed Signed Radix-8 Scalar Multiplication

```
Input: Point $P \in E(F_q)$, A $t$ digit of integer $k$, i.e.,
    $k = (k_{t-1}, k_{t-2}, \ldots, k_0)_8$, $k_j \in \{-1, 0, 1, \ldots, 6\}$.
Output: Point $Q = kP$.
Initialize: $P_{kP} \leftarrow \mathcal{O}$, $P_1 \leftarrow P$, $P_{Acc} \leftarrow P$;
Step 1: For $j = 0$ to $t-1$ do
    Step 1.1: If $k_j \in \{-1, 0, 1, 2, 4\}$ Then
        Step 1.1.1: $P_{kP} \leftarrow P_{kP} + k_j P_{Acc}$;
        Step 1.1.2: $P_{Acc} \leftarrow 8P_{Acc}$; /* Prepare $P_{Acc}^{(j+1)}$ */
        Step 1.1.3: $P_1 \leftarrow P_{Acc} - P_{kP}$;
    Step 1.2: Else If $k_j \in \{3, 5, 6\}$ Then
        Step 1.2.1: $P_1 \leftarrow P_1 + (7 - k_j)P_{Acc}$;
        Step 1.2.2: $P_{Acc} \leftarrow 8P_{Acc}$; /* Prepare $P_{Acc}^{(j+1)}$ */
        Step 1.2.3: $P_{kP} \leftarrow P_{Acc} - P_1$;
Step 2: End For
Step 3: Return $(P_{kP})$;
```

We illustrate Algorithm 2 by showing an example of computing kP. Suppose that $k = 6644$ and has an octal representation of $(14764)_8$, which can be further represented as $(015\bar{1}64)_8$, where $\bar{1} = -1$, using the non-seven recoding method that is shown in Algorithm 1. Table 1 illustrates the process of computing kP by exploiting the proposed signed radix-8 scalar multiplication that is shown in Algorithm 2.

As shown in Table 1, the three registers P_{kP}, P_1, and P_{Acc} are initialized to 0, P, and P, respectively (see the Initialize step in Algorithm 2). The loop started in Step 1, is executed t times, that is $t = \lceil \log_8 6644 \rceil + 1 = 6$ in this example. As shown in Step 1 in Algorithm 2, the *for* loop iteration starts from the least significant octal value of k. This is shown in the third column of Table 1. If the octal digit, i.e., k_j in a column is $k_j \in \{-1, 0, 1, 2, 4\}$, then the operations in Steps from 1.1.1 to

TABLE 1
An Example of the Computation for $kP = 6644P$ Using the Proposed Signed Radix-8 Scalar Multiplication

k_j Groups	Initialization	(Iteration No.) , $k_j = k_j$ value					
		(0) , $k_0 = 4$	(1) , $k_1 = 6$	(2) , $k_2 = -1$	(3) , $k_3 = 5$	(4) , $k_4 = 1$	(5) , $k_4 = 0$
$k_j \in \{-1, 0, 1, 2, 4\}$.	$P_{kP} \leftarrow \mathcal{O}$ $P_{Acc} \leftarrow P$ $P_1 \leftarrow P$	$P_{kP} \leftarrow 4P$ $P_{Acc} \leftarrow 8P$ $P_1 \leftarrow 4P$		$P_{kP} \leftarrow -12P$ $P_{Acc} \leftarrow 512P$ $P_1 \leftarrow 524P$		$P_{kP} \leftarrow 6644P$ $P_{Acc} \leftarrow 32768P$ $P_1 \leftarrow 26124P$	$P_{kP} \leftarrow 6644P$ $P_{Acc} \leftarrow 262144P$ $P_1 \leftarrow 255500P$
$k_j \in \{3, 5, 6\}$.			$P_{kP} \leftarrow 52P$ $P_{Acc} \leftarrow 64P$ $P_1 \leftarrow 12P$		$P_{kP} \leftarrow 2548P$ $P_{Acc} \leftarrow 4096P$ $P_1 \leftarrow 1548P$		

TABLE 2
The 4 Stages of the Processes that Algorithm 2 Evaluates for Each Value of k_j

k_j	Processing Stages	k_j	Processing Stages
-1	$P_{Temp} \leftarrow 2P_{Acc}$. $P_{Temp} \leftarrow 2P_{Temp}$. $P_{kP} \leftarrow P_{kP} - P_{Acc}$,† $P_{Acc} \leftarrow 2P_{Temp}$. $P_1 \leftarrow P_{Acc} - P_{kP}$.	3	$P_{Acc} \leftarrow 2P_{Acc}$. $P_{Acc} \leftarrow 2P_{Acc}$. $P_1 \leftarrow P_{Acc} + P_1$, $P_{Acc} \leftarrow 2P_{Acc}$. $P_{kP} \leftarrow P_{Acc} - P_1$.
0	$P_{Temp} \leftarrow 2P_{Acc}$. $P_{Temp} \leftarrow 2P_{Temp}$. $P_{kP} \leftarrow P_{Acc} - P_1$,† $P_{Acc} \leftarrow 2P_{Temp}$. $P_1 \leftarrow P_{Acc} - P_{kP}$.	4	$P_{Acc} \leftarrow 2P_{Acc}$. $P_{Acc} \leftarrow 2P_{Acc}$. $P_{kP} \leftarrow P_{Acc} + P_{kP}$, $P_{Acc} \leftarrow 2P_{Acc}$. $P_1 \leftarrow P_{Acc} - P_{kP}$.
1	$P_{Temp} \leftarrow 2P_{Acc}$. $P_{Temp} \leftarrow 2P_{Temp}$. $P_{kP} \leftarrow P_{Acc} + P_{kP}$, $P_{Acc} \leftarrow 2P_{Temp}$. $P_1 \leftarrow P_{Acc} - P_{kP}$.	5	$P_{Acc} \leftarrow 2P_{Acc}$. $P_{Temp} \leftarrow 2P_{Acc}$. $P_1 \leftarrow P_{Acc} + P_1$, $P_{Acc} \leftarrow 2P_{Temp}$. $P_{kP} \leftarrow P_{Acc} - P_1$.
2	$P_{Acc} \leftarrow 2P_{Acc}$. $P_{Temp} \leftarrow 2P_{Acc}$. $P_{kP} \leftarrow P_{Acc} + P_{kP}$, $P_{Acc} \leftarrow 2P_{Temp}$. $P_1 \leftarrow P_{Acc} - P_{kP}$.	6	$P_{Temp} \leftarrow 2P_{Acc}$. $P_{Temp} \leftarrow 2P_{Temp}$. $P_1 \leftarrow P_{Acc} + P_1$, $P_{Acc} \leftarrow 2P_{Temp}$. $P_{kP} \leftarrow P_{Acc} - P_1$.

†The SUB operation can be easily obtained using the ADD operation.

1.1.3 are sequentially computed. On the other hand, if $k_j \in \{3, 5, 6\}$, then the operations in Steps from 1.2.1 to 1.2.3 in Algorithm 2 are sequentially computed. Eventually, the content of the P_{kP} register, at iteration $t - 1 = 5$ (initial iteration $= 0$), contains the desired computation of kP, i.e., in the rightmost column in Table 1.

4 Proposed Regular ECSM Scheme

In this section, we present a uniform addition chain scheme that is resistant to SSCA and safe-error fault attacks. The proposed radix-8 ECSM shown in Algorithm 2 is revised to behave in a highly regular manner; so that for any k_j digit, the computational cycle of the addition chain loop is evaluated using the same sequence of EC-operations.

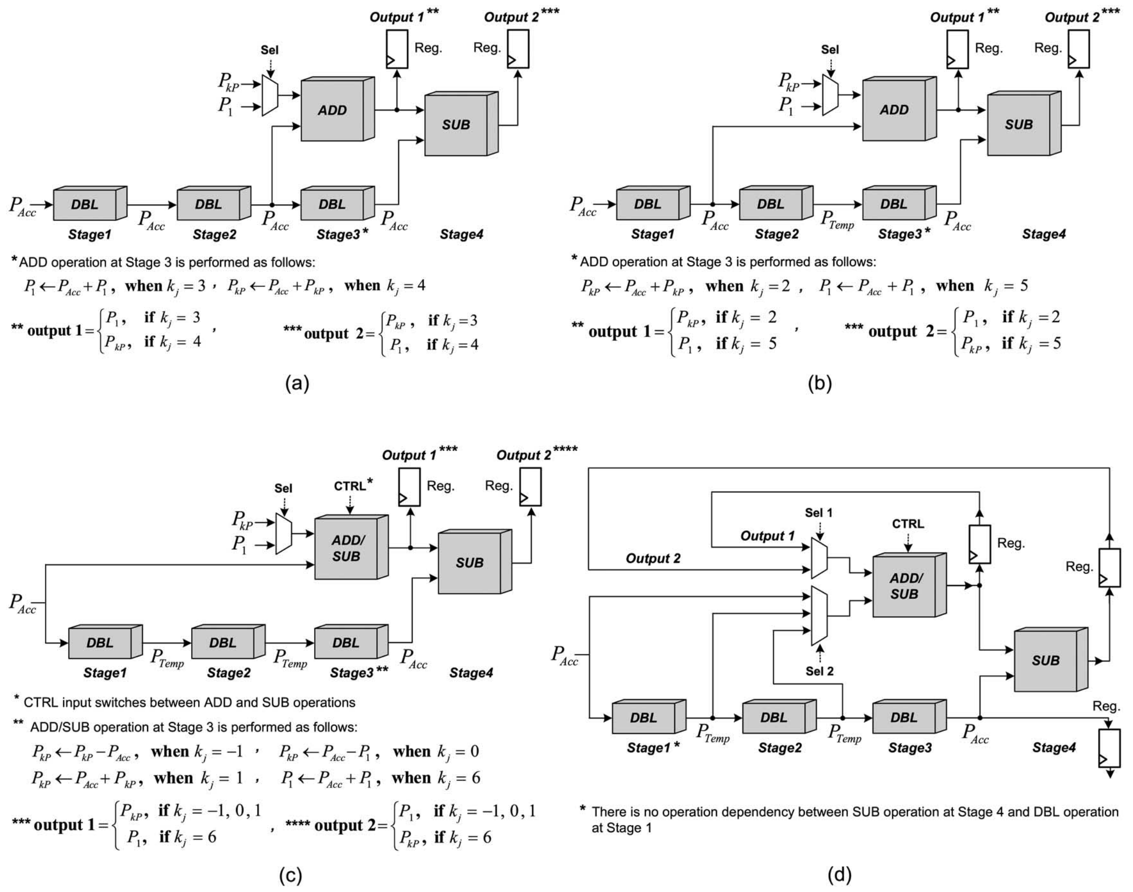
4.1 The Four-Stage Levels

In the following, it is assumed that a temporary register P_{Temp} is provided as part of the processor. It is also assumed that both EC-operations ADD and SUB are indistinguishable under SSCA attacks [50]–[52]. The latter assumption can be justified as follows. The cost of negation operation in $GF(p)$, i.e., mapping $x \rightarrow -x$, can be carried out by one non-modular subtraction (which has about half the cost of a modular addition/subtraction). Considering the extended twisted Edwards curve as an example, one can see from [25] that the cost of $\mathrm{ADD} = 8\mathbf{M} + 10\mathbf{A}$. Based on the experimental ratio of the cost of a modular addition by the one of a modulo multiplication, i.e., $\mathbf{A}/\mathbf{M}$ on the smart cards that is provided in [53], the average ratio is $\mathbf{A}/\mathbf{M} \cong 0.2$. Then, one can obtain the cost of ADD in term of $\mathbf{A}$ as $\mathrm{ADD} \cong 50\ \mathbf{A}$. The cost of SUB for this curve that is equal to the cost of ADD and the cost of modular negation operation, i.e., SUB $\mathrm{SUB} \cong 50.5\ \mathbf{A}$. We conclude that the ratio of cost of the point ADD to the cost of point SUB becomes $\mathrm{ADD}/\mathrm{SUB} \cong 0.99$.

Proposition 1. *For any value of k_j, Algorithm 2 would be evaluated in 4 stages as*

$$\begin{aligned} &Stage\ 1: DBL.\\ &Stage\ 2: DBL.\\ &Stage\ 3: DBL, ADD/SUB.\\ &Stage\ 4: SUB. \end{aligned} \tag{14}$$

Proof. Table 2 provides the evaluation sequence for each case of k_j values separately. Also in Figs. 2(a)–(c) it is shown how Algorithm 2 is evaluated at the EC-operations level for each case of k_j. We provide here a detailed analysis of the main two cases, i.e., when $k_j = -1$, and $k_j = 0$. Given that $k_j = -1$, the operations in Step 1.1 in Algorithm 2 are processed. In Step 1.1.1, the evaluation of P_{kP} requires processing $P_{kP} - P_{Acc}$; hence, the SUB operation that is very similar to the ADD operation is processed. So by shifting the evaluation of this operation, i.e., $P_{kP} = P_{kP} - P_{Acc}$ to



Output 1**
Output 2***
Sel
Reg.
ADD
SUB
DBL
P_{kP}
P_1
P_{Acc}
P_{Temp}
Stage1
Stage2
Stage3*
Stage4
*ADD operation at Stage 3 is performed as follows:
$P_1 \leftarrow P_{Acc} + P_1$, when $k_j = 3$, $P_{kP} \leftarrow P_{Acc} + P_{kP}$, when $k_j = 4$
** output 1 = $\begin{cases} P_1, & \text{if } k_j = 3 \\ P_{kP}, & \text{if } k_j = 4 \end{cases}$, *** output 2 = $\begin{cases} P_{kP}, & \text{if } k_j = 3 \\ P_1, & \text{if } k_j = 4 \end{cases}$
(a)
*ADD operation at Stage 3 is performed as follows:
$P_{kP} \leftarrow P_{Acc} + P_{kP}$, when $k_j = 2$, $P_1 \leftarrow P_{Acc} + P_1$, when $k_j = 5$
** output 1 = $\begin{cases} P_{kP}, & \text{if } k_j = 2 \\ P_1, & \text{if } k_j = 5 \end{cases}$, *** output 2 = $\begin{cases} P_1, & \text{if } k_j = 2 \\ P_{kP}, & \text{if } k_j = 5 \end{cases}$
(b)
Output 1***
Output 2****
CTRL*
ADD/SUB
Stage3**
* CTRL input switches between ADD and SUB operations
** ADD/SUB operation at Stage 3 is performed as follows:
$P_{kP} \leftarrow P_{kP} - P_{Acc}$, when $k_j = -1$, $P_{kP} \leftarrow P_{Acc} - P_1$, when $k_j = 0$
$P_{kP} \leftarrow P_{Acc} + P_{kP}$, when $k_j = 1$, $P_1 \leftarrow P_{Acc} + P_1$, when $k_j = 6$
*** output 1 = $\begin{cases} P_{kP}, & \text{if } k_j = -1, 0, 1 \\ P_1, & \text{if } k_j = 6 \end{cases}$, **** output 2 = $\begin{cases} P_1, & \text{if } k_j = -1, 0, 1 \\ P_{kP}, & \text{if } k_j = 6 \end{cases}$
(c)
Output 1
Output 2
Sel 1
Sel 2
CTRL
Stage1 *
Stage3
* There is no operation dependency between SUB operation at Stage 4 and DBL operation at Stage 1
(d)

Fig. 2. EC-operation dependency graph that shows the usage of both the ADD and the DBL blocks. (a) When $k_j = 3$ or 4. (b) When $k_j = 2$ or 5. (c) When $k_j = -1$, 0, 1, or 6. (d) For all cases of k_j, i.e., $k_j \in \{-1, 0, 1 \ldots, 6\}$. Notice that the SUB operation is used at stage 3 for both cases $k_j = -1$ and $k_j = 0$.

Stage 3 (see Fig. 2(c)), the three Steps: 1.1.1-1.1.3 are evaluated in 4 stages as follows:

$$Stage\ 1 : P_{Temp} \leftarrow 2P_{Acc}.$$
$$Stage\ 2 : P_{Temp} \leftarrow 2P_{Temp}.$$
$$Stage\ 3 : P_{kP} \leftarrow P_{kP} - P_{Acc}, P_{Acc} \leftarrow 2P_{Temp}.$$
$$Stage\ 4 : P_1 \leftarrow P_{Acc} - P_{kP}.$$

Given that $k_j = 0$, the operations in Step 1.1 in Algorithm 2 are processed. In Step 1.1.1, the evaluation of P_{kP} requires no processing. However, in order to keep the scheme consistent with the other cases, i.e., highly regular, we re-evaluate P_{kP} by performing the following operation $P_{kP} = P_{Acc} - P_1$ since the sum of the two points P_{kP} and P_1 are always preserved and are equal to the value of the point P_{Acc}. Notice that this operation affects the evaluation of kP, and, hence, it cannot be considered to be a dummy operation. Then the three Steps: 1.1.1 to 1.1.3 are evaluated in 4 stages as follows:

$$Stage\ 1 : P_{Temp} \leftarrow 2P_{Acc}.$$
$$Stage\ 2 : P_{Temp} \leftarrow 2P_{Temp}.$$
$$Stage\ 3 : P_{kP} \leftarrow P_{Acc} - P_1, P_{Acc} \leftarrow 2P_{Temp}.$$
$$Stage\ 4 : P_1 \leftarrow P_{Acc} - P_{kP}.$$

□

Fig. 2(d), shows the EC-operation dependency for all eight of the different combinations of k_j. An intriguing feature of this scheme is that for all cases of k_j, the same steps are performed, i.e., only the operands are transposed. This means that the cost per 3 bits is fixed at $3\text{DBLs} + 2\text{ADDs}$. It is worth mentioning here that in order to evaluate Steps 1.1.2 or 1.2.2 of Algorithm 2, 3 repeated DBL operations are necessary. Also, in (14), at stage 3 both the ADD/SUB and the DBL operations are evaluated in parallel (see Stage 3 in Fig. 2(a)–(d)).

4.2 The Three-Stage Levels

Based on Proposition 1, the proposed Algorithm 2 can be evaluated in a unified sequence of four stages. Analysing the generalised schedule scheme shown in Fig. 2(d), for the eight cases of k_j values, one can see that the DBL operation evaluated for all cases at Stage 1 has no operation dependency with the SUB operation being evaluated at Stage 4. Since there is no operation dependency between the two EC-operations, the SUB operation that is evaluated at Stage 4 can be rearranged to be performed at Stage 1 of the next iteration. Therefore, the SUB operation of the previous iteration and the first DBL operation of the current iteration can be evaluated in parallel. The sequence order of the EC-operations is then adjusted as shown in Fig. 3(c); hence, a total of 3 stages would be used at each iteration. In this case, the proper initialization of the registers has to be considered, i.e., initially, $P_{Acc} = 8P$, and based on the value of k_0 either

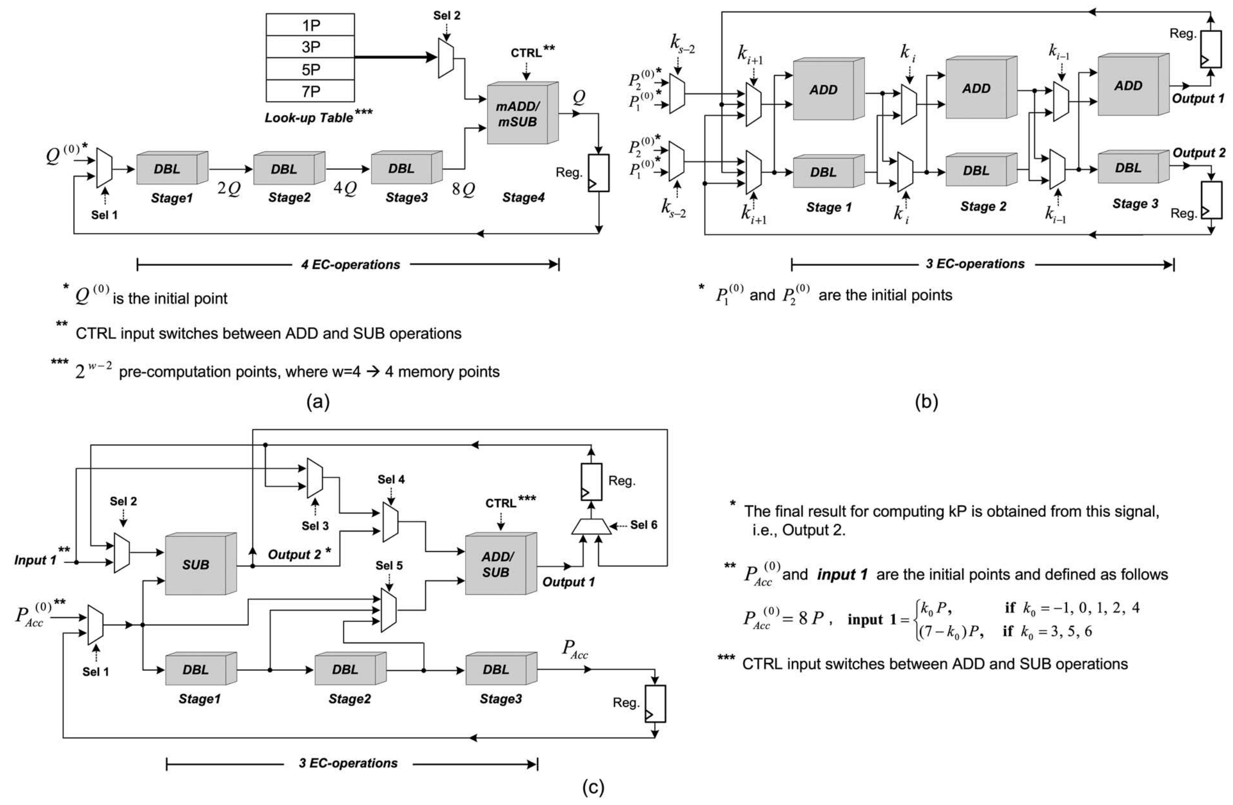

1P
3P
5P
7P
Look-up Table***
Sel 2
CTRL**
mADD/ mSUB
Q
$Q^{(0)*}$
DBL
DBL
DBL
Stage1
2Q
Stage2
4Q
Stage3
8Q
Stage4
Sel 1
Reg.
4 EC-operations
* $Q^{(0)}$ is the initial point
** CTRL input switches between ADD and SUB operations
*** 2^{w-2} pre-computation points, where w=4 → 4 memory points
(a)
k_{s-2}
$P_2^{(0)*}$
$P_1^{(0)*}$
k_{i+1}
k_i
k_{i-1}
ADD
ADD
ADD
DBL
DBL
DBL
Stage 1
Stage 2
Stage 3
Output 1
Output 2
Reg.
3 EC-operations
* $P_1^{(0)}$ and $P_2^{(0)}$ are the initial points
(b)
Input 1**
$P_{Acc}^{(0)**}$
Sel 1
Sel 2
Sel 3
Sel 4
Sel 5
Sel 6
SUB
Output 2 *
CTRL***
ADD/ SUB
Output 1
Reg.
DBL
DBL
DBL
Stage1
Stage2
Stage3
P_{Acc}
3 EC-operations
* The final result for computing kP is obtained from this signal, i.e., Output 2.
** $P_{Acc}^{(0)}$ and input 1 are the initial points and defined as follows
$P_{Acc}^{(0)} = 8P$, input 1 $= \begin{cases} k_0 P, & \text{if } k_0 = -1, 0, 1, 2, 4 \\ (7-k_0)P, & \text{if } k_0 = 3, 5, 6 \end{cases}$
*** CTRL input switches between ADD and SUB operations
(c)

Fig. 3. EC-operations dependency graphs for the SSCA secured methods that show the total memory points required, the total EC-operations costs per 3 scalar bits, and the total computational time complexity per 3 scalar bits at the EC-operations level: (a) For width-4 Okeya method [22]. (b) For the Montgomery Ladder and Joye's binary methods [44], [47]. (c) The proposed radix-8 ESCM method.

$P_1 = (7 - k_0)P$, or $P_{kP} = k_0 P$. We also note that the temporary register P_{Temp} can be omitted in the proposed scheme shown in Fig. 3(c). Let us consider the following two possible scenarios:

1) The first scenario involves the serial implementation design of Fig. 3(c), i.e., one ADD and one DBL are implemented in parallel. In this case, it takes 3 clock cycles to complete one iteration of the *for* loop in Algorithm 2, i.e., processing 3 scalar bits. As one can see from Fig. 3(c), only one DBL operation is required to be executed at clock cycle 2. Then, during the clock cycle 2, the additional temporary registers used to compute the ADD operation become idle and it becomes possible to reuse them to store the contents of P_{Temp}.
2) The second scenario, which is considered in this paper, involves a parallel implementation design of Fig. 3(c), i.e., a total of two ADDs and three DBLs are implemented. In this case, three bits of the scalar (one digit of k_j) are processed at every clock cycle, and the contents of P_{Temp} will be no longer needed to be stored. Furthermore, for hardware resource efficiency in this scenario, a single register can be shared between the two points P_{kP}, and P_1. The strategy is to store one point in the register, and to obtain the result of the second point at the end of the ADD operation at the end of Stage 1 in every iteration (see Fig. 3(c)).

Since all the k_j cases use the same set of EC-operations, ADD and DBL do not have to be indistinguishable. Also, as no dummy operations are introduced, the risk posed by the adaptive fault analysis is minimal [35].

5 Performance Analysis of the Proposed ECSM Scheme

As shown in Fig. 3(c), the power consumption of the proposed scalar multiplication scheme is fixed. This indicates that the proposed scheme is intrinsically protected against SSCA because every iteration in the main loop involves 3 DBLs and 2 ADDs. Furthermore, since no dummy operation is used, any fault introduced into any operation will result in an incorrect scalar multiplication result, which makes it resistant to safe-error fault attacks.

In the following, we evaluate and analyse the efficiency of the proposed ECSM scheme (Fig. 3(c)) and compare it to the other well known ECSM schemes at the EC-operations level. To compare fairly, the proposed scheme evaluates 3 bits of the scalar, and, hence, the comparisons are made corresponding to the 3 bits of the scalar k. First, we compare it to two well-known binary methods: the Double-and-Add [37], and the signed binary methods [14], [19], [39], [40]. Second, we compare it to the non-secure width-4 [21], and the non-secure radix-8 NAF schemes [20]. Third, we compare it to the SSCA aware width-4 window-based methods, i.e., the width-4 Möller [43], and the width-4 Okeya windows schemes (Fig. 3(a)) [22]. Fourth, we compare it to the SSCA aware binary methods: the Montgomery Ladder [44]–[46], and Joye's binary methods (Fig. 3(b)) [47]. In our analysis, we assume that the recoding is secure against SSCA, and has a negligible computational cost.

Table 3 summarizes the comparison of the different ECSM schemes. In this table, the memory consumption is the sum of

TABLE 3
Comparison of Related Binary, and Width-4 Window-Based ECSM Schemes with the Proposed Radix-8 Scheme (Fig. 3(c)) in Terms of Memory Register Space Used, Total EC-Operations Cost, and Computation Time Complexity at the EC-Operations Level Per 3 Scalar Bits Evaluations

Method	Memory Points	Total EC-operations Cost /3 Scalar Bits	Computational Time Complexity/3 Scalar Bits [a]
Non-Secure ECSM Methods			
Double-and-Add [37]	$2 \rightarrow [P, Q]$	4.5 uADD or Atomic Structure [b]	3 EC-operations (Fix)
Signed Binary [14], [19], [39], [40]	$2 \rightarrow [P, Q]$	4 uADD or Atomic Structure [c]	3 EC-operations (Fix)
Width-4 NAF [21]	$4 \rightarrow [P, 3P, 5P, Q]$ [d]	3.67 uADD or Atomic Structure [e]	3.67 EC-operations (Av.)
Radix-8 NAF [20]	$4 \rightarrow [P, 3P, 5P, Q]$ [d]	3.67 uADD or Atomic Structure [e]	3.67 EC-operations (Av.)
Secure ECSM Methods			
Width-4 Möller [43]	$6 \rightarrow [P, 3P, 5P, 7P, 8P, Q]$ [f]	3 DBL & 1 mADD	4 EC-operations (Fix)
Width-4 Okeya [22]	$5 \rightarrow [P, 3P, 5P, 7P, Q]$ [g]	3 DBL & 1 mADD	4 EC-operations (Fix)
Montgomery Ladder [44], [45], [46]	$2 \rightarrow [P_1, P_2]$ [h]	3 DBL & 3 ADD	3 EC-operations (Fix)
Joye's Binary Method [47]	$2 \rightarrow [R_0, R_1]$ [h]	3 DBL & 3 ADD	3 EC-operations (Fix)
Proposed Radix-8 Scheme Fig. 3c	$2 \rightarrow [P_{Acc}, \textit{output 1}]$ [i]	3 DBL & 2 ADD	3 EC-operations (Fix)

[a]Note that the terms Av. and Fix stand for the average and fix measurements of the computation complexity.
[b]Utilizing the atomicity principle, on average, the computation complexity is 3 $\mathrm{DBLs} + 1.5$ mADDs.
[d] $(2^{w-2} - 1)$ pre-computation points, where $w = 4$, and another EC-point is used in the evaluation process.
[e]Utilizing the atomicity principle, on average, the computation complexity is 3 $\mathrm{DBLs} + 0.67$ mADDs.
[f] $(2^{w-2} + 1)$ pre-computation points, where $w = 4$, and another EC-point is used in the evaluation process.
[g] (2^{w-2}) pre-computation points, where $w = 4$, and another EC-point is used in the evaluation process.
[h]If only the x*-coordinates* of the EC-points are computed, then the initial (base) EC-point, i.e., P will be reserved and used to obtain the ADD operation and the y*-coordinate* from the x*-coordinates*. Hence, total memory points would become 3.
[i]If one ADD and one DBL are implemented in parallel to design Fig. 3(c), then the total of the registers would become 3.

the look-up table and the registers required during the evaluation stage. We note that in order to compute the ECSM in a non-secure width-w NAF, a total of $2^{w-2} - 1$ pre-computation points including base point P is required. The width-w of the Möller method is based on $(2^{w-2} + 1)$ pre-computation look-up tables and, hence, for $w = 4$, the total memory consumption in this ECSM scheme is 5 pre-computation points and 1 for the evaluation stage. Also, the SSCA aware width-w NAF method presented by Okeya and Takagi in [22], has more recoding overhead; but, as shown in Table 3, it has 1 memory reduction in the size of the look-up table as compared to the width-4 Möller method in [43]. Hence, a total of 5 memory registers including the register for the evaluation stage are required (see Fig. 3(a)). It can be seen from this table that the secure width-4 window-based ECSM methods requires the highest amount of memory, and that it used at least 40% of the memory registers more compared to the proposed ECSM scheme shown in Fig. 3(c).

The Double-and-Add, signed binary, Radix-8 NAF, and width-4 NAF methods are prone to SSCA. In order to withstand SSCAs, the methods should either use the unified operation approach (cf., [25]) or the atomicity principle (cf. [30] and [24]). The first approach uses an indistinguishable addition, i.e., a uADD that is when the formulas used for both the ADD and the DBL are the same; however, the implementation of such a formula for different models of elliptic curves would suffer from huge area complexity. The atomic structure approach is usually implemented with DBLs and a Jacobian projective-affine mADD operation. It should be noted that the atomic structure schemes are only provided to a few projective coordinates, that is, they are not generalized to all of the elliptic curve models. Further, the architecture design in the atomic schemes is very restricted; hence, the architecture design is restricted to performing a specific number of arithmetic multiplication and squaring operations per each clock cycle.

The SSCA aware binary methods, i.e., the Montgomery Ladder, and Joye's binary methods, require a total of $3\ \mathrm{DBLs} + 3\ \mathrm{ADDs}$ for every 3 bits of the scalar k_j. The proposed scheme requires a total of $3\ \mathrm{DBLs} + 2\ \mathrm{ADDs}$ for every 3 bits of the scalar k_j. This indicates that $1/3$ of the computation of the ADD operations in the proposed ECSM scheme shown in Fig. 3(c) decreases when compared to the SSCA-protected binary methods. It is noted that in those SSCA aware binary methods, the computation of the scalar multiplication can be enhanced at the arithmetic field level. For instance, in the Montgomery Ladder method on the Montgomery curve, only the x*-coordinates* of the EC-points are computed in the EC-operations. As will be shown in Section 6, utilizing the proposed ECSM scheme in a parallel environment, one can gain a significant performance improvement that yields a faster performance time than do the optimized binary ECSM schemes.

The secure width-4 window-based methods require a total of $3\ \mathrm{DBLs} + \mathrm{mADD}$. Assuming that their pre-computed points are kept in affine coordinates. However, as seen in Fig. 3(a)–(c), in terms of computational time complexity, the proposed method along with all other binary methods reveal themselves to be more efficient by observing that in each stage both EC-operations, DBL and ADD, are independent and can be evaluated in parallel. Whereas, the non-secure window-based and secure window-based methods are performed sequentially. Hence, their computational complexity becomes 3.67, and 4 EC-operations, respectively. In order to make these window-based methods, which involve pre-computations with the base point P, feasible for implementations supporting parallel processing of EC-operations, i.e., their computational time complexity becomes 3 EC-operations, all the

TABLE 4
Comparison of the Proposed Radix-8 Scheme (Fig. 3(c)) with the Unified Operation Technique and with Different ECSM Schemes that Are Resist Against Side Channel Attacks in Term of Total Field Arithmetic Operations Per 3 Scalar Bits on the Weierstraß Elliptic Curve

Security Method	Point Operation Cost [a]	ECSM Method	Field Arithmetic Complexity/3 Scalar Bits	
			In Terms of **M** and **S**	When **S**=0.8**M**,
Projective Coordinates Representation [56]				
Unified Operation Techniques	ADD $\rightarrow 12\mathbf{M}+2\mathbf{S}$ DBL [b] $\rightarrow 7\mathbf{M}+3\mathbf{S}$ uADD [c] $\rightarrow 13\mathbf{M}+3\mathbf{S}$	Double-and-Add [37]	$58.5\mathbf{M}+13.5\mathbf{S}$	$69.3\mathbf{M}$
		Signed Binary [14], [19], [39], [40]	$52\mathbf{M}+12\mathbf{S}$	$61.6\mathbf{M}$
		Non-Secure Width-4 NAF [21], [20]	$47.71\mathbf{M}+11.01\mathbf{S}$	$56.51\mathbf{M}$
		Proposed ECSM Scheme Fig. 3c	$45\mathbf{M}+13\mathbf{S}$	$55.4\mathbf{M}$
SSCA Secured ECSM Methods	ADD $\rightarrow 12\mathbf{M}+2\mathbf{S}$ DBL [b] $\rightarrow 7\mathbf{M}+3\mathbf{S}$ mADD $\rightarrow 9\mathbf{M}+2\mathbf{S}$	Secure Width-4 Möller Scheme [43]	$30\mathbf{M}+11\mathbf{S}$ [d]	$38.8\mathbf{M}$
		Secure Width-4 NAF Scheme [22]		
		Montgomery Ladder [44], [45], [46]	$57\mathbf{M}+15\mathbf{S}$	$69\mathbf{M}$
		Joye's Binary Method [47]		
		Proposed ECSM Scheme Fig. 3c	$45\mathbf{M}+13\mathbf{S}$	$55.4\mathbf{M}$
Jacobian Projective Coordinates Representation [57]				
SSCA Secured ECSM Methods	ADD $\rightarrow 12\mathbf{M}+4\mathbf{S}$ DBL [b] $\rightarrow 4\mathbf{M}+4\mathbf{S}$ mADD $\rightarrow 8\mathbf{M}+3\mathbf{S}$	Secure Width-4 Möller Scheme [43]	$20\mathbf{M}+15\mathbf{S}$ [d]	$32\mathbf{M}$
		Secure Width-4 NAF Scheme [22]		
		Montgomery Ladder [44], [45], [46]	$48\mathbf{M}+24\mathbf{S}$	$67.2\mathbf{M}$
		Joye's Binary Method [47]		
		Proposed ECSM Scheme Fig. 3c	$36\mathbf{M}+20\mathbf{S}$	$52\mathbf{M}$

[a] We follow most of the literature in ignoring the cost of $\mathbf{A}$.
[b] It is assumed that $a=-3$.
[c] It is assumed that $a=-1$.
[d] Additional computation of $3(k-1)\mathrm{M}+1\mathbf{I}$, where k is the total pre-computation points in lookup table, is required for the transformation of points to the affine coordinate in the pre-computation stage, i.e., preparing the points in lookup table.

pre-computed points need to be doubled $w-1$ times at each iteration [54].

We apply the proposed ECSM scheme to two well-known Weierstraß elliptic curve models. Table 4 reports the total field arithmetic operations for computing the scalar multiplication using Double-and-Add, signed binary, and non-secure width-4 NAF algorithms with unified addition-or-doubling formulas. A comparison of the proposed ECSM scheme, i.e., Fig. 3(c), with the other secured ECSM methods is also provided in this table. From Table 4, one can see that the secured width-4 methods require less amount of field arithmetic operations. It must be noted however, that the secured width-4 methods impose additional memory registers for the pre-computed points.

In the following section, we take advantage of the ECSM scheme we proposed, i.e., Fig. 3(c), with the objective of deriving faster ECC formulae for parallel architectures. For the comparison with other parallel environment systems, we decided to choose the prime extended twisted Edwards coordinates for the curves defined over $GF(p)$.

6 Parallel Architectures

In this section, we explain how a protected scalar multiplication using the proposed scheme for the prime extended twisted Edwards model can be performed faster than all of the parallel and SSCA-protected schemes over prime fields reported in the literature including the fast Montgomery Ladder method on the Montgomery curve.

The objective of using the proposed scheme, i.e., Fig. 3(c), is to achieve the fastest scalar multiplication result. Note for simplicity purpose, the required auxiliaries (or registers) in the ECSM schemes are not discussed or analysed. Also in the parallelization process, we impose the restriction that the architectures can only be based on SIMD (single instruction multiple data) operations.

The total field arithmetic operations cost of the Montgomery curve is the least among the existing elliptic curve models over prime fields [55]. We recall [44], that an elliptic curve produced by a Montgomery equation is of the form

$$\mathcal{E}_M : By^2 = x^3 + Ax^2 + x,$$

where $A,\ B \in GF(p)$ with $(A^2-4)\ B \neq 0$. Let $P_m(X_m, Z_m)$, and $P_n(X_n, Z_n)$, be two arbitrary points on this curve, and $P_{m-n}(X_{m-n}, Z_{m-n})$ be another point that is equal to the difference between the two points, i.e., $P_{m-n} = P_m - P_n$. Assuming that $Z_{m-n} = 1$, then the coordinates of the point $P_{m+n}(X_{m+n}, Y_{m+n}) = P_m + P_n$ are given as follows [44]

$$\begin{aligned} X_{m+n} &= ((X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n))^2, \\ Z_{m+n} &= X_{m-n}((X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n))^2, \end{aligned}$$

and the coordinates of the doubling formulae, i.e., $P_{2m}(X_{2m}, Z_{2m}) = 2P_m$ are given in [44] by

$$\begin{aligned} 4X_mZ_m &= (X_m + Z_m)^2 - (X_m - Z_m)^2, \\ X_{2m} &= (X_m + Z_m)^2 (X_m - Z_m)^2, \\ Z_{2m} &= (4X_mZ_m)((X_m - Z_m)^2 + ((A+2)/4)(4X_mZ_m)). \end{aligned}$$

A $5\mathbf{M}+4\mathbf{S}+1\mathbf{D}+8\mathbf{A}$, Montgomery Ladder ADDDBL algorithm is given in [55], and a parallel algorithm for the

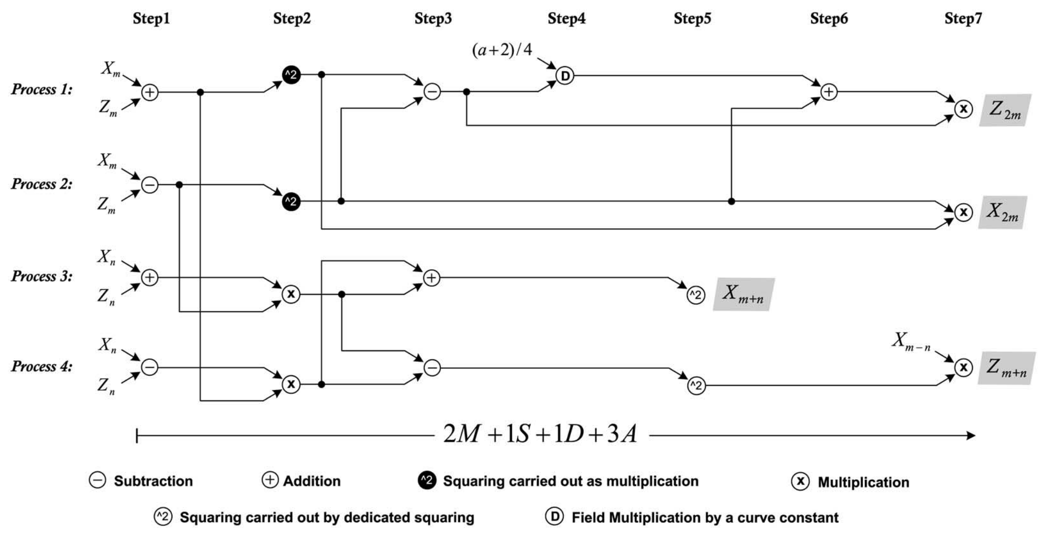


Fig. 4. Data dependency graph for parallel computing of the ADDDBL operation for the x-*coordinates only* Montgomery Ladder method on the Montgomery curve.

Montgomery Ladder is given in [25] at an effective time cost of $2\mathbf{M} + 2\mathbf{S} + 3\mathbf{A}$ using 4-processors. As a point of comparison, in Fig. 4, we derived the fastest timings for the Montgomery Ladder's ADDDBL operation in the parallel strategies. In our scheme, we assumed that the two $\mathbf{S}$ operations performed in Step 2 in Fig. 4 are carried out as $\mathbf{M}$ operations. Then, all of the four operations executed in Step 2 are performed at the same time with a delay of $\mathbf{M}$. We note that the two $\mathbf{S}$ operations

TABLE 5
Comparison of the Proposed Radix-8 ECSM Scheme (Fig. 3(c)) with Different Scalar Multiplication Schemes that Offers Resistance Against Side-Channel Attacks Using Parallel Environments with Respect to the Computation Time Complexity

Scheme - Processors[a]	EC Model - Coordinates	ECSM Method	Comput. Time Complexity/3 Scalar Bits[b]	
			In Terms of **M**, **S** and **D**	**S**=0.8**M** and **D**=0
[26] - 2 Processors[c]	Jacobian Proj. Coordinates	Width-4 Möller Scheme [43]	10**M**+8**S**	16.4**M**
[27] - 2 Processors[d]	PL with *x-coordinate only* [36]	Montgomery Curve [44]	30**M**	30**M**
[28] - 2 Processors[e]	Modified Jacobian Coordinates	Width-4 Möller Scheme [43]	11**M**+10**S**	19**M**
[28] - 3 Processors[f]	Modified Jacobian Coordinates	Width-4 Möller Scheme [43]	10**M**+3**S**	12.4**M**
[29] - 3 Processors[g]	Hessian Proj. Curves	Width-4 Möller Scheme [43]	10**M**+3**S**	12.4**M**
[24] - 2 Processors[h]	Jacobian Proj. Coordinates (atomic)	Non-Secure Width-4 NAF [21], [20]	8.7**M**+8.7**S** (Av.)	15.66**M**
[24] - 3 Processors[i]	Jacobian Proj. Coordinates	Width-4 Möller Scheme [43]	6**M**+8**S**	12.4**M**
[24] - 4 Processors[j]	Jacobian Proj. Coordinates	Width-4 Möller Scheme [43]	3**M**+10**S**	11**M**
[25] - 2 Processors[k]	Extended Twisted Edwards (unified)	Non-Secure Width-4 NAF [21], [20]	14.68**M**+3.67**D** (Av.)	14.68**M**
[25] - 4 Processors[l]	Extended Twisted Edwards (unified)	Non-Secure Width-4 NAF [21], [20]	7.34**M**+3.67**D** (Av.)	7.34**M**
[25] - 4 Processors[m]	Extended Twisted Edwards	Width-4 Möller Scheme [43]	5**M**+3**S**	7.4**M**
[25] - 4 Processors	PL with *x-coordinate only* [36]	Montgomery Curve [44]	6**M**+6**S**	10.8**M**
Fig 4 - 4 Processors	PL with *x-coordinate only* [36]	Montgomery Curve [44]	6**M**+3**S**+3**D**	8.4**M**
Proposed 8 Processors[n] Fig. 5	Extended Twisted Edwards	proposed radix-8 scheme (Fig. 3c)	5**M**+1**S**	5.8**M**

[a]Processors are based on the number of parallel field multipliers $\mathbf{M}$. The effects of the number of auxiliaries (or registers) to the area is not discussed here.
[b]We follow most of the literature in ignoring the cost of $\mathbf{A}$. The experimental ratio $\mathbf{A}/\mathbf{M}$ on the smart cards is provided in [53].
[c]A sequence of 3 DBLs, i.e., $3(2\mathrm{M} + 2\mathrm{S})$ followed by the mADD, i.e., $(4\mathbf{M} + 2\mathbf{S})$.
[d]A sequence of 3 parallel computing of [ADD & DBL], i.e., $3(10\mathrm{M}) = 30\mathbf{M}$.
[e]A sequence of 3 DBLs, i.e., $3(2\mathbf{M} + 3\mathbf{S})$ followed by the mADD, i.e., $(5\mathbf{M} + 1\mathbf{S})$.
[f]A sequence of 3 DBLs, i.e., $3(2\mathbf{M} + 1\mathbf{S})$ followed by the mADD, i.e., $(4\mathbf{M})$.
[g]A sequence of 3 DBLs, i.e., $3(2\mathbf{M} + 1\mathbf{S})$ followed by the ADD, i.e., $(4\mathbf{M})$.
[h]Each point is represented by sextuplet coordinates. An average of 3 $\mathrm{DBLs} + 0.67$ mADDs, i.e., $3(2\mathbf{M} + 2\mathbf{S}) + 0.67(4\mathbf{M} + 4\mathbf{S})$.
[i]Each point is represented by the sextuplet coordinates. A sequence of 3 DBLs, i.e., $3(1\mathbf{M} + 2\mathbf{S})$ followed by the mADD, i.e., $(3\mathbf{M} + 2\mathbf{S})$.
[j]Each point is represented by the sextuplet coordinates. A sequence of 2 special DBLs, i.e., 2(3S) followed by a generalized DBL, i.e., $1\mathbf{M} + 2\mathbf{S}$ followed by the mADD, i.e., $2\mathbf{M} + 2\mathbf{S}$.
[k]Each point is represented by the quadruple coordinates. A sequence of 3.67 uDBLs, i.e., $3.67(4\mathbf{M} + 1\mathbf{D})$.
[l]Each point is represented by the quadruple coordinates. A sequence of 3.67 uDBLs, i.e., $3.67(2\mathbf{M} + 1\mathbf{D})$. Stated in [59] that it is the fastest known approach to performing elliptic curve point operations.
[m]Each point is represented by the quadruple coordinates. A sequence of 3 DBLs, i.e., $3(1\mathbf{M} + 1\mathbf{S})$ followed by the ADD, i.e., $(2\mathbf{M})$.
[n]Each point is represented by the quadruple coordinates. A sequence of ADDDBL, DBL, and ADDDBL. As shown in Fig. 5, the ADDDBL operation can be performed at an effective cost of $2\mathbf{M}$, and from [25], the DBL operation can be performed at an effective cost of $1\mathbf{M} + 1\mathbf{S}$.

performed in Step 4 are carried out by dedicated squaring. From this figure, one can see that the ADDDBL operation for the Montgomery Ladder can be performed with an effective time of $2\mathbf{M} + 1\mathbf{S} + 1\mathbf{D} + 3\mathbf{A}$ for each bit of the scalar. It is worth noting that dependencies restrict us from achieving further reductions with more processes. Consequently, for the Montgomery Ladder algorithm, the computation time complexity per each of the 3 bits of the scalar as shown in Table 5 is $6\mathbf{M} + 3\mathbf{S} + 3\mathbf{D}$.

We now investigate the 8-processor implementation of the ADDDBL operation for the prime extended twisted Edwards curve. The twisted Edwards curve is a generalization of the Edwards curve [56] and has the equation [58]

$$\mathcal{E}_T : ax^2 + y^2 = 1 + dx^2y^2,$$

where $a, d \in GF(p)$, with $ad(a-d) \neq 0$. To develop a faster way of performing the DBL and the ADD operations, in [25], an additional auxiliary coordinate was added to the twisted Edwards coordinates. It is observed in [25] that the extended twisted Edwards curves are represented by the quadruple coordinates, and for the special case $a = -1$, the DBL and the ADD operations can be performed at a computation cost of $4\mathbf{M} + 4\mathbf{S} + 6\mathbf{A}$, and $8\mathbf{M} + 10\mathbf{A}$ operations, respectively, assuming that the field arithmetic addition and subtraction are equal [25].

Let $P_1(X_1, Y_1, T_1, Z_1)$, and $P_2(X_2, Y_2, T_2, Z_2)$, be two distinct points on $\mathcal{E}^e$, where $\mathcal{E}^e$ denotes the extended twisted Edwards coordinates, with $Z_1 \neq 0$ and $Z_2 \neq 0$, then the coordinates of the point $P_3(X_3, Y_3, T_3, Z_3) = P_1 + P_2$ are given as follows [25]

$$\begin{aligned}
X_3 &= (X_1Y_2 - Y_1X_2)(T_1Z_2 + Z_1T_2),\\
Y_3 &= (Y_1Y_2 - X_1X_2)(T_1Z_2 - Z_1T_2),\\
T_3 &= (T_1Z_2 + Z_1T_2)(T_1Z_2 - Z_1T_2),\\
Z_3 &= (Y_1Y_2 - X_1X_2)(X_1Y_2 - Y_1X_2), \qquad (15)
\end{aligned}$$

and the coordinates of the doubling formulae, i.e., $P_4(X_4, Y_4, T_4, Z_4) = 2P_1$ are given in [25] by

$$\begin{aligned}
X_4 &= 2X_1Y_1(2Z_1^2 - Y_1^2 + X_1^2),\\
Y_4 &= (Y_1^2 - X_1^2)(Y_1^2 + X_1^2),\\
T_4 &= 2X_1Y_1(Y_1^2 + X_1^2),\\
Z_4 &= (Y_1^2 - X_1^2)(2Z_1^2 - Y_1^2 + X_1^2). \qquad (16)
\end{aligned}$$

It was shown in [25], that both the ADD and the DBL operations can be performed utilizing 4-processors with an effective time of $2\mathbf{M} + 3\mathbf{A}, 1\mathbf{M} + 1\mathbf{S} + 3\mathbf{A}$, respectively. We propose a composite ADDDBL operation for this curve by splitting the computational task of both the ADD and the DBL operations into 5 steps with the utilization of 8-processors. The data dependency graph of both (15) and (16) is presented in Fig. 5, which shows that combining these two equations requires a computation cost of $12\mathbf{M} + 4\mathbf{S} + 15\mathbf{A}$ (1 field addition operation is saved). According to this figure, the effective time can be reduced to $2\mathbf{M} + 3\mathbf{A}$ operations with 8 processes. As shown in Fig. 5, the ADDDBL operation scheme consists of eight independent processing elements, i.e., process 1 to process 8. A finite-field arithmetic operation is represented by a circle and it is labelled according to the type of action it performs. In our scheme, we assumed that the $\mathbf{S}$ operations performed in Step 2 are carried out as $\mathbf{M}$ operations. The interconnections among the eight processing elements are needed because of the data dependency in the operation in each processing element. For instance, when arriving at Step 2, process 5 needs the output data generated by process 4 in Step 1. Thus, an interconnection between process 4 and process 5 is needed to support such data dependency. Similarly, other necessary interconnections should also be obtained. From this figure, and the effective time cost of DBL operation for the prime extended twisted Edwards curve that is obtained from [25], we conclude that one round of computing 3 bits of the scalar in the proposed scheme (Fig. 3(c)), which requires a sequence of ADDDBL, DBL, and ADDDBL, can be completed in an effective time of $5\mathbf{M} + 1\mathbf{S} + 9\mathbf{A}$. Table 5 shows the computation time complexity of the different scalar multiplication schemes that offer resistance against side-channel attacks in the parallel environments.

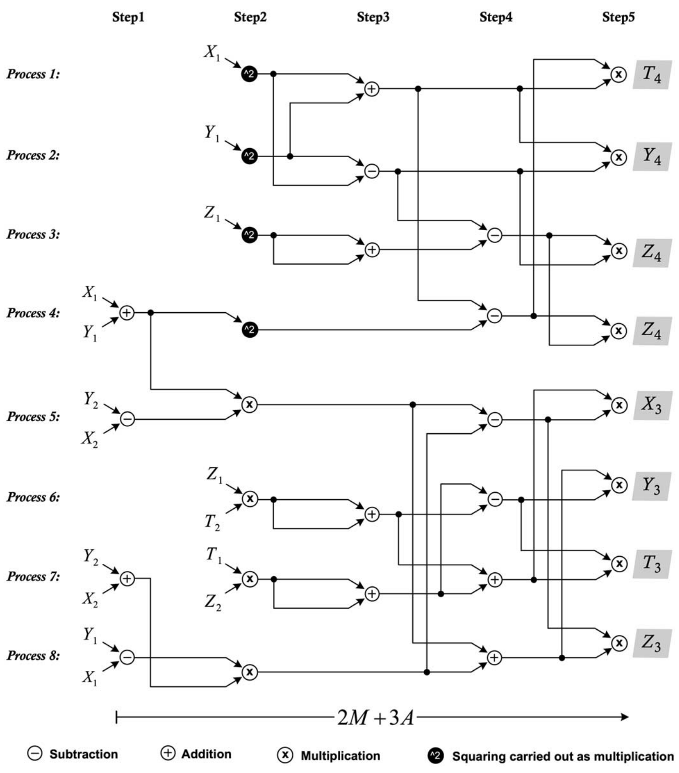


Fig. 5. Data dependency graph for parallel computing of the proposed ADDDBL operation for the prime extended twisted Edwards curve.

In general, for an s-bit scalar multiplication, the Montgomery Ladder method shown in Fig. 4 requires $6\frac{(s-1)}{3}\mathbf{M} + 3\frac{(s-1)}{3}\mathbf{S}$, whereas the extended twisted Edwards curve in the proposed ECSM method requires $\frac{5(s)}{3}\mathbf{M} + \frac{(s)}{3}\mathbf{S}$. Table 6 shows the comparison of the 4-processor scheme for the Jacobian projective coordinates presented in [24], the 4-processor scheme for the extended twisted Edwards curve presented in [25], the 4-processor Montgomery Ladder method on the Montgomery curve that is obtained from [25], the 4-processor Montgomery Ladder method on the

TABLE 6
Comparison of Related Parallel Schemes with the Proposed 8-Processor Scheme for the Extended Twisted Edwards Curve over Prime Fields, Which Is Shown in Fig. 5, with Respect to the Computational Time Complexities for the Bit Lengths of the Underlying Fields of NIST Recommended Curves [60]

Prime Field Size $GF(p)$	Scheme - Processors	Computational Time Complexities
$s = 192$	4 Processors for Jacobian Projective Coordinates [24]	191**M**+637**S**
	4 Processors for Extended Twisted Edwards [25]	319**M**+191**S**
	Montgomery Ladder method on the Montgomery curve [25]	382**M**+382**S**
	Montgomery Ladder method on the Montgomery curve (Fig. 4)	382**M**+191**S**
	Proposed 8 processors scheme (Fig. 5, and DBL operation obtained from [25])	320**M**+64**S**
$s = 224$	4 Processors for Jacobian Projective Coordinates [24]	223**M**+744**S**
	4 Processors for Extended Twisted Edwards [25]	372**M**+223**S**
	Montgomery Ladder method on the Montgomery curve [25]	446**M**+446**S**
	Montgomery Ladder method on the Montgomery curve (Fig. 4)	446**M**+223**S**
	Proposed 8 processors scheme (Fig. 5, and DBL operation obtained from [25])	374**M**+75**S**
$s = 256$	4 Processors for Jacobian Projective Coordinates [24]	255**M**+850**S**
	4 Processors for Extended Twisted Edwards [25]	425**M**+255**S**
	Montgomery Ladder method on the Montgomery curve [25]	510**M**+510**S**
	Montgomery Ladder method on the Montgomery curve (Fig. 4)	510**M**+255**S**
	Proposed 8 processors scheme (Fig. 5, and DBL operation obtained from [25])	427**M**+86**S**
$s = 384$	4 Processors for Jacobian Projective Coordinates [24]	383**M**+1277**S**
	4 Processors for Extended Twisted Edwards [25]	639**M**+383**S**
	Montgomery Ladder method on the Montgomery curve [25]	766**M**+766**S**
	Montgomery Ladder method on the Montgomery curve (Fig. 4)	766**M**+383**S**
	Proposed 8 processors scheme (Fig. 5, and DBL operation obtained from [25])	640**M**+128**S**
$s = 521$	4 Processors for Jacobian Projective Coordinates [24]	520**M**+1734**S**
	4 Processors for Extended Twisted Edwards [25]	867**M**+520**S**
	Montgomery Ladder method on the Montgomery curve [25]	1040**M**+1040**S**
	Montgomery Ladder method on the Montgomery curve (Fig. 4)	1040**M**+520**S**
	Proposed 8 processors scheme (Fig. 5, and DBL operation obtained from [25])	869**M**+174**S**

Montgomery curve that is shown in Fig. 4, and the 8-processor scheme for the extended twisted Edwards curve that is shown in Fig. 5 in terms of the computational time complexities for the prime fields that are recommended by NIST.

7 Conclusion

In this paper, a new radix-8 scalar multiplication scheme is introduced that can be used for any elliptic curve model. It allows one to compute each of the three bits of the scalar with five point arithmetic operations in a unified sequence. We showed that the properties of the proposed scheme enhance parallelism at both the point arithmetic, and the field arithmetic levels. Further, it implicitly provides resistance against certain implementation attacks.

We applied the proposed scheme to the prime extended twisted Edwards curves for the computation of a scalar multiplication in an 8-processor environment. We then provided the performance estimates and presented the comparisons between the proposed scheme and the other known parallel schemes. We further showed that to the best of the authors' knowledge, the 8-processor scheme provided in this work is the fastest SSCA protected scalar multiplication scheme over prime fields in the parallel environment. The proposed 8-processor scheme provided in this work can be applied to all of the parallel hardware implementations and also to parallel software environments such as a Cell multiprocessor [61], and ePUMA [12].

References

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.
[2] M. Abdelguerfi, B. S. Kaliski, Jr., and W. Patterson, "Public-Key security systems," *IEEE Micro*, vol. 16, no. 3, pp. 10–13, Jun. 1996.
[3] L. Batina, S. B. Örs, B. Preneel, and J. Vandewalle, "Hardware architectures for public key cryptography," *Integr. VLSI J.*, vol. 34, no. 1, pp. 1–64, May 2003.
[4] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
[5] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.
[6] D. Boneh, "Twenty years of attacks on the RSA cryptosystem," *Notices Amer. Math. Soc.*, vol. 46, no. 2, pp. 203–213, Feb. 1999.
[7] Y. Y. Song, *Cryptanalytic Attacks on RSA*. New York, NY: Springer-Verlag, 2008.
[8] D. R. Hankerson, A. J. Menezes, and S. A. Vanstone, *Guide to Elliptic Curve Cryptography*. New York, NY: Springer-Verlag, 2004.
[9] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. Adv. Cryptology (CRYPTO'85)*, Aug. 1985, pp. 417–426.
[10] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, Jan. 1987.
[11] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *J. Cryptology*, vol. 14, no. 4, pp. 255–293, Aug. 2001.
[12] J. Tolunay, "Parallel gaming related algorithms for an embedded media processor," Master's thesis, Linköping Univ., Linköping, Sweden, 2012.

[13] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*. Boston, MA: Kluwer, 1993.

[14] IEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques, IEEE Std. 13632–2008, Jan. 2009.

[15] E. W. Knudsen, "Elliptic scalar multiplication using point halving," in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Security: Adv. Cryptology (ASIACRYPT'99)*, Nov. 1999, pp. 135–149.

[16] V. Dimitrov, L. Imbert, and P. K. Mishra, "Efficient and secure elliptic curve point multiplication using double-base chains," in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Security: Adv. Cryptology (ASIACRYPT'05)*, Dec. 2005, pp. 59–78.

[17] M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery, "Trading inversions for multiplications in elliptic curve cryptography," *Des. Codes Cryptography*, vol. 39, no. 2, pp. 189–206, May 2006.

[18] P. Longa and A. Miri, "New composite operations and precomputation scheme for elliptic curve cryptosystems over prime fields," in *Proc. Int. Workshop Practice Theory Public-Key Cryptography (PKC'08)*, Mar. 2008, pp. 229–247.

[19] G. W. Reitwiesner, "Binary arithmetic," in *Advances in Computers*, vol. 1. New York, NY: Academic Education Press, 1960, pp. 231–308.

[20] S. Arno and F. S. Wheeler, "Signed digit representations of minimal hamming weight," *IEEE Trans. Comput.*, vol. 42, no. 8, pp. 1007–1010, Aug. 1993.

[21] J. A. Solinas, "Efficient arithmetic on Koblitz curves," in *Des. Codes Cryptography*, vol. 19, no. 2–3, pp. 195–249, Mar. 2000.

[22] K. Okeya and T. Takagi, "The width-w NAF method provides small memory and fast elliptic scalar multiplications secure against side channel attacks," in *Proc. Cryptographers Track RSA Conf.: Topics Cryptology (CT-RSA'03)*, Apr. 2003, pp. 328–343.

[23] N. Koblitz, "CM-Curves with good cryptographic properties," in *Proc. Adv. Cryptology (CRYPTO'91)*, Aug. 1991, pp. 279–287.

[24] P. Longa and A. Miri, "Fast and flexible elliptic curve point arithmetic over prime fields," *IEEE Trans. Comput.*, vol. 57, no. 3, pp. 289–302, Mar. 2008.

[25] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson, "Twisted Edwards curves revisited," in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Security: Adv. Cryptology (ASIACRYPT'08)*, Dec. 2008, pp. 326–343.

[26] T. Izu and T. Takagi, "Fast elliptic curve multiplications with SIMD operations," in *Proc. Int. Conf.: Inf. Commun. Security (ICICS'02)*, Dec. 2002, pp. 217–230.

[27] W. Fischer, C. Giraud, E. W. Knudsen, and J.-P. Seifert, Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against non-differential side-channel attacks, IACR, Cryptology ePrint Archive, 2002/007 [Online]. Available: http://eprint.iacr.org/2002/007.

[28] K. Aoki, F. Hoshino, T. Kobayashi, and H. Oguro, "Elliptic curve arithmetic using SIMD," in *Proc. Int. Conf.: Inf. Security (ISC'01)*, Oct. 2001, pp. 235–247.

[29] N. P. Smart, "The Hessian form of an elliptic curve," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst. (CHES'01)*, May 2001, pp. 118–125.

[30] P. K. Mishra, "Pipelined computation of scalar multiplication in elliptic curve cryptosystems (extended version)," *IEEE Trans. Comput.*, vol. 55, no. 8, pp. 1000–1010, Aug. 2006.

[31] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves using Gaussian normal basis," *IEEE Trans. VLSI Syst.*, vol. 20, no. 8, pp. 1453–1466, Aug. 2012.

[32] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proc. Int. Cryptology Conf.: Adv. Cryptology (CRYPTO'96)*, Aug. 1996, pp. 104–113.

[33] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst. (CHES'99)*, Aug. 1999, pp. 292–302.

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

[35] R. Avanzi. (2005). Side channel attacks on implementations of curve-based cryptographic primitives, IACR, Cryptology ePrint Archive, 2005/017 [Online]. Available: http://eprint.iacr.org/2005/017/.

[36] J. López and R. Dahab, "Fast multiplication on elliptic curves over GF (2^m) without precomputation," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst. (CHES'99)*, Aug. 1999, pp. 316–327.

[37] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2, 1st ed. Reading, MA: Addison Wesley, May 1969.

[38] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography*. Cambridge, MA: Cambridge Univ. Press, Jul. 1999.

[39] A. D. Booth, "A signed binary multiplication technique," *Q. J. Mech. Appl. Math.*, vol. 4, no. 2, pp. 236–240, Aug. 1951.

[40] K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi, "Signed binary representations revisited," in *Proc. Int. Cryptology Conf.: Adv. Cryptology (CRYPTO'04)*, Aug. 2004, pp. 123–139.

[41] Y. Sung-Ming, S. Kim, S. Lim, and S. Moon, "A countermeasure against one physical cryptanalysis may benefit another attack," in *Proc. Int. Conf.: Inf. Security Cryptology (ICISC'01)*, Dec. 2001, pp. 414–427.

[42] C. Clavier and M. Joye, "Universal exponentiation algorithm: A first step towards provable SPA-resistance," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst. (CHES'01)*, May 2001, pp. 300–308.

[43] B. Möller, "Securing elliptic curve point multiplication against side-channel attacks," in *Proc. Int. Conf.: Inf. Security (ISC'01)*, Oct. 2001, pp. 324–334.

[44] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Math. Comput.*, vol. 48, no. 177, pp. 243–264, Jan. 1987.

[45] K. Okeya, H. Kurumatani, and K. Sakurai, "Elliptic curves with the montgomery-form and their cryptographic applications," in *Proc. Int. Workshop Practice Theory Public Key Cryptosystems (PKC'00)*, Jan. 2000, pp. 238–257.

[46] M. Joye and S.-M. Yen, "The montgomery powering ladder," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst. (CHES'02)*, Aug. 2002, pp. 291–302.

[47] M. Joye, "Highly regular right-to-left algorithms for scalar multiplication," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst. (CHES'07)*, Sep. 2007, pp. 135–147.

[48] C. Vuillaume and K. Okeya, "Flexible exponentiation with resistance to side channel attacks," in *Proc. Int. Conf.: Appl. Cryptography Netw. Security (ACNS'06)*, Jun. 2006, pp. 268–283.

[49] B. Parhami, *Computer Arithmetic Algorithms and Hardware Designs*, 2nd ed. London, U.K.: Oxford Univ. Press, 2000.

[50] A. Kargl and G. Wiesend, "On randomized addition-subtraction chains to counteract differential power attacks," in *Proc. Int. Conf.: Inf. Commun. Security (ICICS'04)*, Oct. 2004, pp. 278–290.

[51] N. Thériault, "SPA resistant left-to-right integer recordings," in *Proc. Int. Workshop: Select. Areas Cryptography (SAC'05)*, Aug. 2005, pp. 345–358.

[52] D.-G. Han and T. Takagi. (2005). Some analysis of radix-r representations, IACR, Cryptology ePrint Archive, 2005/402 [Online]. Available: http://eprint.iacr.org/2005/402.

[53] C. Giraud and V. Verneuil, "Atomicity improvement for elliptic curve scalar multiplication," in *Proc. IFIP WG 8.8/11.2 Int. Conf.: Smart Card Res. Adv. Appl. (CARDIS'10)*, Apr. 2010, pp. 80–101.

[54] K. Järvinen, "Optimized FPGA-based elliptic curve cryptography processor for high-speed applications," *Integr. VLSI J.*, vol. 44, no. 4, pp. 270–279, Sep. 2011.

[55] D. J. Bernstein and T. Lange. (2012). Explicit-Formulas database, Joint Work by D. J. Bernstein, and T. Lange, Building on Work by Many Authors [Online]. Available: http://www.hyperelliptic.org/EFD/.

[56] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Security: Adv. Cryptology (ASIACRYPT'07)*, Dec. 2007, pp. 29–50.

[57] H. Cohen, A. Miyaji, and T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates," in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Security: Adv. Cryptology (ASIACRYPT'98)*, Oct. 1998, pp. 51–65.

[58] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards curves," in *Proc. Int. Conf. Cryptology Africa: Progress Cryptology (AFRICACRYPT'08)*, Jun. 2008, pp. 389–405.

[59] W. B. Joppe, "On the cryptanalysis of public-key cryptography," PhD dissertation, Univ. École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, 2012.

[60] Digital Signature Standard (DSS), Fed. Inf. Processing Standard, Nat'l Inst. of Standards and Technology Std, FIPS PUB 186-3, Jun. 2009.

[61] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: Built for speed," *IEEE Micro*, vol. 26, no. 3, pp. 10–23, May 2006.

Ebrahim A. H. Abdulrahman received the BSc degree in computer science and engineering from Qatar University, Doha, in 2002, with the first rank, and the MSc degree in information technology (networking) from James Cook University, Townsville, Australia, in 2005. He is currently pursuing the PhD degree with the Department of Electrical and Computer Engineering, Western University, London, Canada. In February 2002, he joined the Department of Computer Engineering, University of Bahrain, Zallaq, as a graduate teaching and research assistant, where he was awarded a master's degree and a PhD scholarship.

Arash Reyhani-Masoleh received the BSc degree in electrical and electronic engineering from Iran University of Science and Technology, Tehran, in 1989, the MSc degree in electrical and electronic engineering from the University of Tehran, Iran, in 1991, both with the first rank, and the PhD degree in electrical and computer engineering from the University of Waterloo, Ontario, Canada, in 2001. From 1991 to 1997, he was with the Department of Electrical Engineering, Iran University of Science and Technology. From June 2001 to September 2004, he was with the Centre for Applied Cryptographic Research, University of Waterloo, where he was awarded a Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowship in 2002. In October 2004, he joined the Department of Electrical and Computer Engineering, University of Western Ontario, London, Canada, where he is currently a tenured associate professor. His current research interests include algorithms and VLSI architectures for computations in finite fields, fault-tolerant computing, and error-control coding. He has been awarded an NSERC Discovery Accelerator Supplement (DAS) in 2010. Currently, he serves as an associate editor for Integration, the *VLSI Journal* (Elsevier). He is a member the IEEE Computer Society.