Journal of Computers Vol. 28, No. 6, 2017, pp. 64-78
doi:10.3966/199115992017122806006

Flexible Parallel Computing for Elliptic Curve Scalar Multiplication with Resistance against Simple Side-Channel Attacks

Keke Wu

School of Computer, Shenzhen Institute of Information Technology
Shenzhen 518172, China
wukk@sziit.edu.cn

Received 9 July 2016; Revised 14 February 2017; Accepted 14 March 2017

Abstract. In this paper, we proposed a flexible parallel method of scalar multiplication for elliptic curve cryptosystems (ECC) using our proposed partition and integration models. In contrast to previous parallel scalar multiplication methods, our method can be implemented into various parallel systems with resistance against simple side-channel attacks (SSCA). Implementation results indicate that our method can be implemented into various parallel systems, which provide a speedup ratio of 1.55 relative to the binary method on a generic curve over F_p. Furthermore, for resisting against SSCA, previous secure scalar multiplication methods can be integrated into our method. Experimental results indicate that our SSCA-protected version can resist against SSCA.

Keywords: elliptic curve cryptosystems, parallel computing, scalar multiplication, simple side-channel attacks

1 Introduction

Elliptic curve cryptosystems (ECC) were independently introduced by Miller [1] and Koblitz [2] in the 1980s, which attracted increasing attention in recent years due to the requirement of shorter private key length with the same security level, in comparison with other public-key cryptosystems. In ECC, the scalar multiplication is the dominant and time-consuming operation. Since then, acceleration of scalar multiplication in ECC has become an important issue. Given a scalar d and an elliptic curve point P, a scalar multiplication dP is computed by a series of addition operations (A) and doubling operations (D) of the point P, depending upon the bit sequence representing of d. The structure of scalar multiplication involves three computational levels: scalar multiplication algorithm, point arithmetic, and field arithmetic. We will mainly focus on improvements at the scalar multiplication algorithm level to speed up the ECC scalar multiplication. In this regard, traditionally, a scalar multiplication dP is computed through the binary method [3], whereas the operation time is long due to the sequential process. For the scalar d with k bit length, the time complexity of the binary method reaches $kD + (k/2)A$. Enormous research has been conducted to speed up the binary method through reducing the number of elementary point addition operations, such as non-adjacent form (NAF) method, window NAF method, and sliding window method etc. Detailed analyses of scalar multiplication techniques are given in [4].

However, these sequential scalar multiplication methods are not suitable for high-performance end servers because of the sequential computation. To adapt to high-performance implementations, parallel scalar multiplication methods are needed. Various parallelisms for scalar multiplication have been proposed [5-13]. These methods can efficiently speed up scalar multiplication, and some of them [7-11] can further secure scalar multiplication against so-called simple side-channel attacks (SSCA) [14-16] that have aroused widespread attentions. However, most of them are limited to parallel systems with fixed number of processors (normally 2, 3 or 4 processors) due to using the point or field arithmetic level parallelism.

In this paper, based on our previous work of scalable parallel processing of scalar multiplication [17], we propose a flexible parallel scalar multiplication method to be implemented into various parallel systems that are not limited to fixed number of processors. Based on our previous work, furthermore, we give the parallelization implementation for proposed parallel scalar multiplication method in a cluster system, and we give the side-channel analysis experiment to verify the side-channel security of our proposed method. Usually, various parallel systems with flexible number of processors involve symmetric multiprocessor, massively parallel processor, distributed shared memory multiprocessor, cluster of workstations, grid computing environment, and even cloud computing environment etc.

In contrast to previous scalar multiplication parallelisms, we focus our efforts to parallelize ECC scalar multiplication operations at the scalar multiplication algorithm level. We first iteratively partition the scalar d (with k bit length) in half by n times into 2^n equal length sub-scalar. Then the 2^n sub-scalar multiplications are distributed onto 2^n processors in a parallel system. Next, we compute the 2^n sub-scalar multiplications with the binary method in parallel and then obtain 2^n elliptic curve points concurrently. Finally, based on our proposed scalar integration model, we compute additions of every two points the same way recursively in parallel, until we only need to compute a single addition of two points. The division is flexible and on the task level parallelism, which is naturally adapted for execution in various parallel systems. The time complexity is only $kD + (k/2p + \log p)A$. With the optimal partition times $n = \log k - 2$, the time complexity is reduced to $kD + (\log k)A$. Moreover, our technique does not aim at any special structure of the elliptic curve due to using the scalar multiplication algorithm level parallelism, and it covers the case where curve points are not known a priori. Thus, our method is on an arbitrary elliptic curve in Weierstrass form as specified [18-20].

On the other hand, since our method is at the highest operation level, previous SSCA-protected scalar multiplication methods [16, 22-28] can be integrated into our method for resisting against SSCA. Performance analysis results of our parallel SSCA-protected version indicate that our flexible parallel scheme can be more suitable for parallelization of expensive SSCA-protected scalar multiplication methods.

Finally, we implement our proposed parallel method in a Chinese DAWNING TC5000 cluster system, which is a flexible hyper parallel processing architecture. The maximal speedup ratio reaches 1.55 relative to the sequential binary method on a generic curve over $\mathbb{F}_p$ with 256 bits for the prime p and a scalar d of size 256 bits. Meanwhile, we implement the binary method and double-and-add-always method in smart cards and present side-channel experimental results to prove our SSCA-protected version is secure against SSCA.

The rest of the paper is organized as follows. In Section 2, we introduce the background in ECC scalar multiplication and side-channel attacks. In Section 3, we propose the scalar partition and integration models. In Section 4, we propose a flexible parallel scalar multiplication method based on the scalar partition and integration models. In Section 5, we analyze the time complexity for our method. In Section 6, we present a simple example for the flexible method. In Section 7, we propose an SSCA-protected version for our parallel method. In Section 8, we present detailed comparison between previous parallel scalar multiplication methods and our proposed flexible parallel method. In Section 9, we implement our method in a parallel system and present implementation results. In Section 10, we implement the binary method and double-and-add-always method in smart cards and present side-channel experimental results. Finally, we conclude in Section 11.

2 Background

In this section, we first present the classic binary method of elliptic curve scalar multiplication, and then discuss the current state of affairs in scalar multiplication countermeasures against simple side-channel attacks (SSCA).

2.1 Classic Binary Method

The classic method to compute a scalar multiplication dP is the straightforward binary method [3] based on the binary expansion of a scalar $d = (d_k d_{k-1} ... d_2 d_1)_2$, where d_k and d_1 is the most significant bit (MSB)

and the least significant bit (LSB) of d, respectively, and k is the bit length of d. This method includes two different implementation types [4]. One is the right to left binary method and the other is the left to right binary method. We describe one version (left to right) of them as follows.

Algorithm 1 (Classic Binary Method).

Input: $d = (d_k d_{k-1} ... d_2 d_1)_2$, an elliptic curve point P

Output: dP

```
Q=O;
fori = k to 1 do          /* scan d from MSB to LSB */
{
   Q= 2Q;                 /* point doubling */
   if (d_i = 1) then Q = Q + P;  /* point addition */
}
Return Q;
```

On average, the expected number of bit "1" in the binary representation of d is $k/2$, thus the expected time complexity of the binary method is approximately $k/2$ point additions (A) plus k point doublings (D), denoted as $kD + (k/2)A$.

2.2 Side-channel Attacks

Side-channel attacks (SCA) [14-16] exploit leaked side-channel information, such as power consumption, electromagnetic emanations and running time etc., from running cryptographic devices to reveal the secret keys. Generally, simple SCA (SSCA) and differential SCA (DSCA) are the two main attack techniques. The goal of SSCA is to learn the secret key using the information obtained through carefully observing the side-channel leakage. For example, the binary method (Algorithm 1) consists of a point doubling followed by a point addition if the key bit is "1", and a point doubling if the key bit is "0". Since the side-channel pattern of point addition is different from that of point doubling, binary implementation operations are key-dependent executions that attackers can easily reveal the secret key from a single side-channel trace. The more sophisticated DSCA that perform the programs multiple times and manipulate the results with statistical tools are not really a threat to ECC since they are easily avoided by randomizing the inputs [16]. Therefore, in this paper we concentrate on SSCA.

Great research efforts [16, 21-27] have been invested to protect scalar multiplication against SSCA. These countermeasures can be generally classified into the following three groups. The first group is to make the processing of bits "1" and bits "0" of multiplier d indistinguishable, typically by means of double-and-add-always method [16], Montgomery method [21], and partition method [26] where scalar multiplication operations appear as a point doubling followed by a point addition regularly. The second group is to make the point addition and the point doubling indistinguishable by means of unifying the addition formulas [22-24]. The third group is to insert dummy operations and divide each process into atomic blocks so that it can be expressed as the repetition of instruction blocks which appear equivalent by SSCA [22, 27]. All these SSCA-protected methods can be integrated into our proposed parallel method for resisting against SSCA.

3 Proposed Partition and Integration Models

In this section, we propose an elliptic curve cryptographic scalar partition and integration technique. The basic idea is to iteratively partition the binary bit string of a scalar in half by several times, and then these sub-scalars are integrate into the original decimal scalar recursively based on the conversion rule between binary and decimal. Consequently, a scalar multiplication can be divided into several sub-scalar multiplications along the scalar partition procedures. The products of these sub-scalar multiplications can then be accumulated into the final product of the scalar multiplication along the scalar integration procedures. The scalar partition and integration techniques are well suited to parallelism.

Let a k bit length scalar d be iteratively partitioned in half by n times, i.e. d is divided into 2^n equal length sub-scalars. Let each sub-scalar of d be denoted as B_i^n for $i = 1, 2, ..., 2^n - 1, 2^n$, i.e. $d = B_{2^n}^n \,||\, B_{2^n-1}^n \,||\, ... \,||\, B_2^n \,||\, B_1^n$, where '||' is the concatenation operator. The depth of this full binary tree

is the partition times n. The number of leaves is 2^n that is the number of sub-scalars. To better illustrate the scalar integration procedures, we first present a definition as follows.

Definition 1. Let x, y be positive integers, and let $|y|$ denote the bit length of the binary y. We define the following function:

$$f(x,y) = 2^{|y|} \cdot x + y \,. \tag{1}$$

Using the definition, we can quickly obtain a lemma as follows.

Lemma 1. Let B_j^{i+1} and B_{j-1}^{i+1} denote the j^{th} and $(j\text{-}1)^{\text{th}}$ sub-scalars after partition d in half by i+1 times, respectively, where $i \in \{0, 1, ..., n-1\}$ and $j \in \{2^{i+1}, 2^{i+1}-2, ..., 4, 2\}$. We refer to them as $(i\text{+}1)^{\text{th}}$-partition sub-scalars. Then we have the i^{th}-partition sub-scalar as follows:

$$B_{j/2}^{i} = f(B_j^{i+1}, B_{j-1}^{i+1}) \,. \tag{2}$$

The equation (2) also can be expressed by a function $f_{j/2}^{i} \Leftrightarrow f(B_j^{i+1}, B_{j-1}^{i+1})$.

Proof. Because B_j^{i+1} and B_{j-1}^{i+1} denote the j^{th} and $(j\text{-}1)^{\text{th}}$ sub-scalars after partition d in half by i+1 times, respectively, we have the i^{th}-partition sub-scalar $B_{j/2}^{i} = B_j^{i+1} \,||\, B_{j-1}^{i+1}$, where $i \in \{0, 1, ..., n-1\}$ and $j \in \{2^{i+1}, 2^{i+1}-2, ..., 4, 2\}$. According to the conversion rule between binary and decimal, and Definition 1, we can conclude the following equation:

$$B_{j/2}^{i} = B_j^{i+1} \,||\, B_{j-1}^{i+1} = 2^{|B_{j-1}^{i+1}|} \cdot B_j^{i+1} + B_{j-1}^{i+1} = f(B_j^{i+1}, B_{j-1}^{i+1}) \,. \tag{3}$$

According to Lemma 1, we can integrate every two sub-scalars among the 2^n sub-scalars the same way recursively in parallel until we only need to integrate two sub-scalars into the initial scalar d. The scalar integration procedures are the procedures of converting binary to decimal, which is the reverse scalar partition model.

4 Flexible Parallel Computations

In this section, we first depict the flexible parallel implementation model based on our proposed scalar partition and integration models. Then we propose a flexible parallel scalar multiplication method.

4.1 Flexible Parallelization Model

Based on the scalar partition model, the scalar multiplication dP can be divided into 2^n sub-scalar multiplications that can be computed by the binary method in parallel as follows.

$$\begin{aligned} dP &= (B_{2^n}^{n} \,||\, B_{2^n-1}^{n} \,||\, \ldots\ldots \,||\, B_2^n \,||\, B_1^n) \cdot P \\ &\Rightarrow (B_{2^n}^{n} \cdot P), (B_{2^n-1}^{n} \cdot P), \ldots\ldots, (B_2^n \cdot P), (B_1^n \cdot P) \\ &\Rightarrow Q_{2^n}^{n}, Q_{2^n-1}^{n}, \ldots\ldots, Q_2^n, Q_1^n, \end{aligned} \tag{4}$$

where Q_j^n = Binary Method (B_j^n, P), i.e. we obtain 2^n elliptic curve point outputs concurrently.

To better depict the procedures of accumulating these sub-scalar multiplications into the final product of the scalar multiplication dP in parallel, we present the following theorem.

Theorem 1. Let Q_j^{i+1} and Q_{j-1}^{i+1} denote the products of sub-scalar multiplications $B_j^{i+1} \cdot P$ and $B_{j-1}^{i+1} \cdot P$, respectively, where $i \in \{0,1,...,n-1\}$ and $j \in \{2^{i+1}, 2^{i+1}-2, ..., 4, 2\}$. We refer to them as $(i\text{+}1)^{\text{th}}$-partition sub-scalar multiplications. Thus, the product of the i^{th}-partition sub-scalar multiplication is as follows:

$$Q_{j/2}^{i} = f(Q_j^{i+1}, Q_{j-1}^{i+1}) \,. \tag{5}$$

Herein, we assume that $|Q_{j-1}^{i+1}|$ gets the bit length of correspondent sub-scalar B_{j-1}^{i+1}. The equation (5) also

can be expressed by a function $f^i_{j/2} \Leftrightarrow f(Q^{i+1}_j, Q^{i+1}_{j-1})$.

Proof. According to Lemma 1 and Definition 1 presented above, we have the following equation:

$$B^i_{j/2} = f(B^{i+1}_j, B^{i+1}_{j-1}) = 2^{|B^{i+1}_{j-1}|} \cdot B^{i+1}_j + B^{i+1}_{j-1}. \quad \textbf{(6)}$$

The equation (6) is multiplied by an arbitrary elliptic curve point P on both sides of the equation. Thus we can conclude the following equation:

$$\begin{aligned} B^i_{j/2} \cdot P = Q^i_{j/2} &= 2^{|B^{i+1}_{j-1}|} \cdot B^{i+1}_j \cdot P + B^{i+1}_{j-1} \cdot P \\ &= 2^{|B^{i+1}_{j-1}|} \cdot Q^{i+1}_j + Q^{i+1}_{j-1} \\ &= f(Q^{i+1}_j, Q^{i+1}_{j-1}). \end{aligned} \quad \textbf{(7)}$$

According to the scalar integration model and Theorem 1, we compute additions of every two points among the 2^n points the same way recursively in parallel, until we only need to compute a single addition of two points.

4.2 Proposed Method

According to the parallelization procedures introduced above, we develop a flexible parallel scalar multiplication method as follows.

Algorithm 2 (Flexible Parallel Method).

Input: $d = B^n_{2^n} \,||\, B^n_{2^n-1} \,||\, \ldots\ldots \,||\, B^n_1$, an elliptic curve point P

Output: dP

$Q = O$;

for $j = 1$ to 2^n do in parallel

{

Q^n_j = Binary Method (B^n_j , P);

}

for $i = n-1$ downto 0 do

for $j = 2^{i+1}$ downto 2 by 2 do in parallel

$Q^i_{j/2} = f(Q^{i+1}_j, Q^{i+1}_{j-1})$;

Return Q^0_1;

In the first "for" loop of Algorithm 2, these 2^n sub-scalar multiplications $B^n_j \cdot P$ are independently executed in parallel by calling the binary method, and generate 2^n elliptic curve points as outputs concurrently. The scalar partition times n is flexible. The maximum of n is determined by the available number of processors p in a parallel system, where $n = \lfloor \log p \rfloor$. In the second and third "for" loops, these 2^n points are recursively accumulated into the final solution in parallel.

Our technique works by iteratively dividing a task (scalar multiplication) into several sub-tasks (sub-scalar multiplications) of the same type, until these sub-tasks are distributed into the available processors in a parallel system. The results of the sub-tasks are then recursively accumulated into the final solution to the original task. The division of the task is flexible and closure. Therefore, our method is flexible and on the task level parallelism (i.e. scalar multiplication operation level parallelism) that are adapted to various parallel systems with flexible number of processors.

However, previous parallelisms of scalar multiplication are on the point or field operation level parallelism (i.e. instruction level parallelism), which are not on the task's closure. Hence, previous parallelisms of scalar multiplication only can be adapted to parallel systems with fixed number of processors. Moreover, we do not exploit any special structure of the curve for the parallelization of the scalar multiplication, and we cover the case where elliptic curve points are not known a priori. Thus, our method works for an arbitrary Weierstrass curve.

5 Performance Analysis

Generally, we use the numbers of point additions (A) and point doublings (D) to evaluate the time complexity in scalar multiplication methods. In the first "for" loop of Algorithm 2, we use the binary method to compute every sub-scalar multiplication $B_j^n \cdot P$ in parallel, where $j \in \{1, 2, 3, ..., 2^n\}$. The product is denoted as Q_j^n. Thus, the time complexity of the first loop is as follows:

$$t_{B_j} = \frac{k}{2^n} \cdot D + \frac{1}{2} \cdot \frac{k}{2^n} \cdot A . \tag{8}$$

According to Definition 1, the function $f(Q_j^{i+1}, Q_{j-1}^{i+1})$ takes $(k/2^{i+1})D + A$ in parallel in the third "for" loop that is embedded in the second "for" loop that executes n time iterations in sequence. Thus the time complexity of Algorithm 2 is as follows:

$$t_p = t_{B_j} + (\frac{k}{2^n} + \frac{k}{2^{n-1}} + ... + \frac{k}{2})D + nA = kD + (\frac{k}{2^{n+1}} + n)A . \tag{9}$$

In terms of the equation (9), the problem to solve the minimum of t_p is the problem to solve the minimum of the following function:

$$g(x) = \frac{k}{2^{x+1}} + x . \tag{10}$$

When $g'(x) = 0$, i.e. $x = \log k + \log(\ln 2) - 1$, $g(x)$ reaches the minimum. Thus, the optimal partition times n_{opt} are:

$$n_{opt} = \begin{cases} \log k + \lfloor \log(\ln 2) \rfloor - 1 = \log k - 2, \\ \log k + \lceil \log(\ln 2) \rceil - 1 = \log k - 1. \end{cases} \tag{11}$$

For simplicity, we get the optimal partition times $n_{opt} = \log k - 2$. When the available number of parallel processors p is no less than $2^{\log k - 2} = k/4$, we obtain the best case time complexity $t_{p\min} = kD + (\log k)A$. The time complexity of Algorithm 1 also can be given as follows:

$$t_p = \begin{cases} kD + (\frac{k}{2p} + \log p)A, \ when \ \ p < k/4, \\ kD + (\log k)A, \ when \ \ p >= k/4, \end{cases} \tag{12}$$

which are significantly less than $kD + (k/2)A$ of the binary method when the available number of parallel processor p is relative large.

For example, if the bit length of a scalar d is k = 256, then optimal partition times $n = \log 256 - 2 = 6$. When $p < 2^6$, we should partition the scalar d in half by $\log_2 p$ times into p parts for obtaining the time complexity $t_p = 256D + (256/2p + \log p)A$. When $p \geq 2^6$, however, we should partition the scalar d in half by 6 times for obtaining the best case time complexity $t_{p\min} = 256D + 8A$.

6 A Simple Example

Herein, we give an example to depict the process of the proposed technique. For the sake of simplicity, we only assume the scalar $d = (38749)_{10} = (1001011101011101)_2$ the bit length of d is k = 16.

6.1 Partition and Integration Models

According to the equation (11), the optimal partition times are $n_{opt} = \log 16 - 2 = 2$. We partition the scalar d in half by 2 times as shown in Fig. 1.

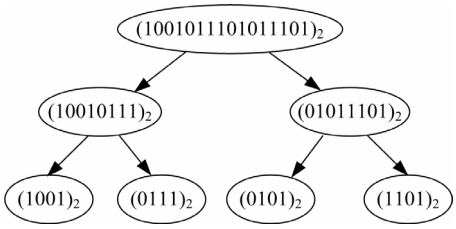

Fig. 1. Partition a scalar in half by 2 times

According to Lemma 1, we can integrate every two sub-scalars among the 4 sub-scalars the same way recursively in parallel until we only need to integrate two sub-scalars into the initial scalar d. The scalar integration model is shown in Fig. 2.

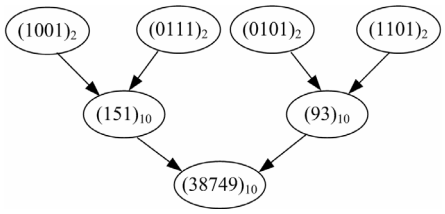

Fig. 2. Integrating four sub-scalars into initial scalar

6.2 Parallelization Procedures

Based on the scalar partition model, the scalar multiplication $dP = 38749P$ can be divided into four sub-scalar multiplications that can be computed by the binary method in parallel as follows.

$$38749P = (1001 \| 0111 \| 0101 \| 1101)P$$
$$\Rightarrow (1001)P,\ (0111)P,\ (0101)P,\ (1101)P.$$

By calling the binary method for computing the four sub-scalar multiplications, we obtain four elliptic curve point outputs concurrently.

According to the scalar integration model and Theorem 1, we compute additions of every two points among the four points the same way recursively in parallel, until we only need to compute a single addition of two points. We present the parallelization procedures of the scalar multiplication $dP = 38749P$ as following Fig. 3.

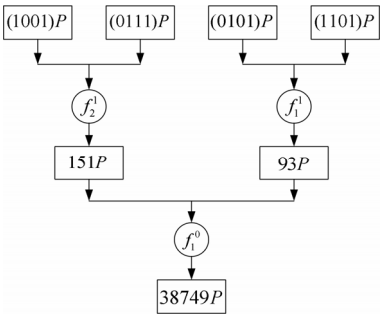

Fig. 3. Parallelization procedures of a scalar multiplication

The four sub-scalar multiplications $1001P$, $0111P$, $0101P$ and $1101P$ are executed in four parallel processors concurrently by calling the binary method. According to Theorem 1, the former two points

and the latter two points are accumulated into one point $151P$ and one point $93P$ in two processors concurrently by means of f_2^1 and f_1^1, respectively. Then the two points $151P$ and $93P$ are accumulated into the final one point $38749P$ in one processor by means of f_1^0.

According to the equation (12), the best case time complexity of the example is about $t_p = 16D + 4A$. However, the time complexity of the example implemented by the sequential binary method reaches about $t_s = 16D + 8A$.

7 SSCA-Protected Version

In Algorithm 2, the 2^n sub-scalar multiplications of our method are executed independently in parallel by calling the binary method, thus the kinds of variants of the binary method can be applied to our method by means of substituting the binary method. For example, the three group SSCA-protected scalar multiplication methods mentioned above can be integrated into our method for resisting against SSCA by means of substituting the binary method.

7.1 Proposed Method

Algorithm 2 mainly consists of two "for" loops. The second "for" loop is to continuously execute a serious of point doubling operations followed by one point addition operation, and the number of point doublings is dependent upon the bit length not the bit value of sub-scalars. Hence, the second "for" loop is inherently key-independent execution that is secure for resisting against SSCA. The first "for" loop is key-dependent execution due to the unprotected binary method. Thus, previous SSCA-protected scalar multiplication methods (double-and-add-always method [16], Montgomery method [21], partition method [26], unifying the addition formulas [22], [23], [24], and side-channel atomicity [22], [27] etc.) can be still applied to the first "for" loop of Algorithm 2 for resisting against SSCA by means of substituting the unprotected binary method.

For example, we substitute the binary method with double-and-add-always method to compute these sub-scalar multiplications in parallel for side-channel security as shown in Algorithm 3.

Algorithm 3 (Flexible Parallel Secure Method).

Input: $d = B_{2^n}^n \,||\, B_{2^n-1}^n \,||\, \ldots\ldots \,||\, B_1^n$, an elliptic curve point P

Output: dP

$Q = O$;

for $j = 1$ to 2^n do in parallel

{

Q_j^n = Double-and-add-always Method (B_j^n , P);

/* substitute the binary method with an SSCA-protected scalar multiplication method */

}

for $i = n-1$ downto 0 do

for $j = 2^{i+1}$ downto 2 by 2 do in parallel

$Q_{j/2}^i = f(Q_j^{i+1}, Q_{j-1}^{i+1})$;

Return Q_1^0;

Similarly, we can substitute the binary method with other SSCA-protected scalar multiplication methods, which also can achieve the same secure effect.

7.2 Time Complexity

The complexity of double-and-add-always method is about $kD + kA$, thus the time complexity of Algorithm 3 is as follows:

$$\begin{aligned} t_p &= (\frac{k}{2^n}D + \frac{k}{2^n}A) + (\frac{k}{2^n} + \frac{k}{2^{n-1}} + ... + \frac{k}{2})D + nA \\ &= kD + (\frac{k}{2^n} + n)A. \end{aligned} \tag{13}$$

Similarly, we solve the optimal partition times $n_{opt} = \log k - 1$ (or $\log k$). The time complexity of Algorithm 3 also can be given as follows:

$$t_p = \begin{cases} kD + (\frac{k}{p} + \log p)A, \ when \ \ p < k/4, \\ kD + (1 + \log k)A, \ when \ \ p >= k/4, \end{cases} \tag{14}$$

which are significantly less than $kD + kA$ of double-and-add-always method when the available number of parallel processor p is relative large.

8 Comparison

Similarly to other systems, ECC can be adapted to parallel architectures at different algorithmic levels. Aoki, Hoshino and Kobayashi [5], Izu and Takagi [6] introduced efficient parallel point operations targeting SIMD-based processors. Aoki et al. [5] introduced modified Jacobian coordinates (X, Y, Z, Z^2) to develop fast parallel formulas for platforms that can execute two and three operations simultaneously. Izu and Takagi [6] presented formulas for two-processor architectures. Longa and Miri [11] introduced the replacing multiplications methodology that allows the development of superior parallel operations that are more efficient for multiprocessor/parallel execution. In this regard, Longa and Miri [11] proposed faster parallel formulas that are able to execute three and four operations simultaneously. However, the limitation of these works is that they rely on traditional point operation formulas, which are restricted to a fixed number of squarings and multiplications over finite fields.

The previous methods target unprotected implementations where simple side-channel analysis (SSCA) is not a concern. However, Fisher, Giraud, Knudsen and Seifet, Izu and Takagi [7-8] presented efficient parallel schemes on generic curves over prime fields using the Montgomery Ladder, which is intrinsically protected against SSCA because every iteration in the main loop involves one doubling and one addition. An advantage of this method is that the formulas involve computation with the x-coordinate only. In particular, Fischer et al. [7] presented a more attractive scheme since it parallelizes doublings and additions at the field operation level, whereas Izu and Takagi [8] proposed parallel method in point operation level in the main loop. The latter has the limitation that the cost of the every iteration is determined by the most costly point operation, namely, point doubling and addition. Later, Izu and Takagi [9] improved the previous proposals and introduced a unified Doubling-Addition formula for the Montgomery Ladder method. The composite formula was then efficiently parallelized. Mishra proposed a pipelined approach for generic curves over prime fields using the standard point arithmetic [10]. In this scheme, each point operation is protected against SSCA using atomicity and the atomic block execution is done through a pipeline, where up to two atomic blocks can be computed simultaneously. Because a pipelined atomic operation can begin its execution before the previous atomic operation is complete, the total throughput is significantly reduced to a few atomic blocks. Longa and Miri [11] propose a faster two-processor SSCA-protected scheme that introduces further cost reductions by using the enhanced atomic structure with squarings. The atomic structure not only offers true protection against SSCA by distinguishing multiplications from squarings but also allows us to pack more field operations in each block. Ahmed, Turki and Paul [12] computed scalar multiplication more efficiently using the post-computation method in the more typical case where the elliptic curve point is variable. Reza and Arash [13] proposed efficient and high speed architectures to implement point multiplication on binary Edwards and generalized Hessian curves. They perform a data-flow analysis and investigate maximum number of parallel multipliers to be employed to reduce the latency of point multiplication on these curves.

In a word, previous parallel scalar multiplication methods are on point or field arithmetic level parallelism that only can be adapted to parallel systems with fixed number of processors (normally 2, 3 or 4). In this paper, we focus our efforts to parallelize scalar multiplication at the scalar multiplication

algorithm level for the implementation in various parallel systems with flexible number of processors. On the other hand, the binary method embedded in our method can be substituted with previous SSCA-protected scalar multiplication methods [16, 22-28] to resist against SSCA.

In contrast to previous methods, our method is on scalar multiplication algorithm level (task level) parallelism that prompts good scalability. Consequently, our method can be implemented into various parallel systems with flexible number of parallel processors.

On the other hand, since our method is on the highest operation level, previous parallel scalar multiplication techniques [5-13] can be integrated into our method for further speedup. These parallel techniques usually involve efficient elliptic curve point or field arithmetic. In a sense, our method is downward compatible with previous parallel scalar multiplication methods, which are on the point or field operation level parallelism. Consequently, there is no need to compare the time complexity between our method and previous parallel scalar multiplication methods.

9 Parallelization Implementations

In this section, we implement our method (Algorithm 2) on generic curves in a parallel system. For the implementation of Algorithm 2 we exploit a Chinese DAWNING TC5000P cluster system (Fig. 4), which is a flexible hyper parallel processing architecture: Generation IV Blade Server, 7U/40PCS architecture, 4-way AMD 1.9GHz master frequency 4-core Bacelona CPU, 4-way AMD Shanghai 8374HE 2.2GHz processor, 4GB Reg. per processor, 64GB solid state disk, dual 1000M network adapter.

Fig. 4. Chinese DAWNING TC5000 cluster system

We choose a generic curve $E: y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ with 256 bits for the prime p, a generating point P of this group and a scalar d of size 256 bits. The running time of the binary method (Algorithm 1) is about 7.13ms when choosing a single processor from the cluster system. Table 1 summarizes our running time (ms) and speedup ratio (S_p) results for various implementations of Algorithm 2 on a generic curve in parallel systems with flexible number of parallel processors.

Table 1. Speedup ratio and running time t (ms) for partition times n and parallel processor number p on a generic curve when $k = 256$.

n	1	2	3	4	5	6	7	8
p	2	4	8	16	32	64	128	256
t	5.77	5.19	4.91	4.73	4.64	4.61	4.82	5.25
S_p	1.24	1.37	1.45	1.51	1.54	1.55	1.48	1.36

Table 1 can be clearly described by Fig. 5. The lower curve is the implementation results. Similarly to the performance analysis results, the S_p of the implementation is increasing when $p < 64$. However, the difference is that S_p is decreasing when $p > 64$ (not $p > 128$). When $p = 64$, the maximal speedup ratio ($S_{p\max}$) reaches 1.55. We also can find the slope is decreasing with the increase of the number of parallel processors when $p < 64$. Moreover, every S_p and slope of the implementation results are less than that every correspondent S_p and slope of the performance analysis results when $p < 64$. These variations indicate that implementation results are slower than performance analysis results, which results from communication delays between units. The communication delay is increasing with the increase of the number of parallel processors. Thus, when the number of parallel processors p reaches 128 even 256, S_p is decreasing significantly. Nevertheless, the $S_{p\max}$ of the implementation still reaches 1.55, which is only

less about 17.6% than the $S_{p\max} = 1.88$ of the performance analysis. We also can get a suitable parallel processor number $p = 16$ where $S_p = 1.51$ that is only less about 2.6% than the $S_{p\max} = 1.55$.

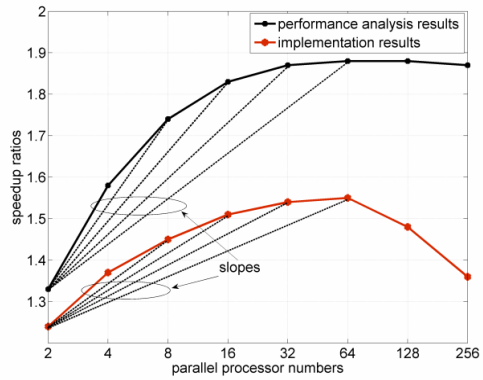

Fig. 5. Comparison between implementation results and performance analysis results of our method on a generic curve

10 Side-Channel Experiments

In this section, we use power measurement experiments to evaluate side-channel security of the binary method and double-and-add-always method to prove our proposed SSCA-protected version (Algorithm 3) can resist against simple side-channel attacks (SSCA).

10.1 Experimental Setup

Our experimental setup consists of a PC, a power tracer (embedded a resistor), a digital oscilloscope, a USB line, a network cable, a BNC trigger line, a BNC signal line and three smart cards. The experimental setup is as shown in Fig. 6. The two smart cards with 8 bit 10MHz microprocessors are implemented with the binary method, double-and-add-always method, respectively, where EC-operations of scalar multiplication methods are implemented in software implementation style except the hardware implementation of modulo multiplication operation in prime field. The sampling frequency of power traces is 781.2K samples per second in our experiments.

Fig. 6. Side-channel experimental setup

10.2 Side-channel Models

Simple side-channel attacks (SSCA) attackers try to derive the key more or less directly from one sample. This often requires detailed knowledge about the implementation of the cryptographic algorithm under attack. Generally, the principles of SSCA are: (1) the attackers need to be able to monitor the side-channel leakage of the device under attack; and, (2) in the attacked device, the key must have a significant impact on the side-channel leakage within a trace or very few traces. We define following

SSCA model and security model of ECC based on the principles of SSCA and the scalar multiplication implemented rule.

Definition 2. S*SCA Model*: SSCA on scalar multiplication implementations exploits: (1) the different patterns between the side-channel features of point addition (ADD) and point doubling (DBL); and, (2) the different patterns depend upon the key bits due to conditional branch statements. We refer to this SSCA model as *key-dependent pattern*.

According to the Definition 2, knowledge of how the algorithm is used and implemented facilitates SSCA. Any implementation where the execution path is determined by the key bits has a potential vulnerability.

Definition 3. S*SCA-Resistant Security Model*: The SSCA-resistant security is to remove the dependence between the different side-channel patterns of ADD and DBL and the key bits. We refer to this security model as *key-independent pattern*.

According to the Definition 3, the three group secure methods mentioned above are to remove the key-dependent patterns by means of indistinguishing the processing of bits "1" and bits "0" of multiplier *d*, indistinguishing the ADD and DBL, and dividing each process into atomic blocks by inserting dummy operations so that it can be expressed as the repetition of instruction blocks which appear equivalent by SSCA, respectively.

10.3 Unprotected Binary Method

SSCA are made easier for unprotected scalar multiplication algorithms because the field operations of ADD have a different side-channel pattern than that of point doubling DBL. We do power experiment of the binary method implementation (as shown in Fig. 7): the peaks in ADD trace are 3 clusters, but several slim clusters in DBL trace, and the running time of ADD is obviously longer than DBL. Obviously, there are different features of side-channel traces between ADD and DBL. Therefore, the power trace satisfies the first condition of Definition 2.

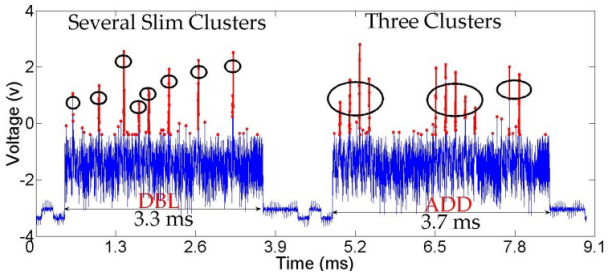

Fig. 7. The different power patterns between a DBL and an ADD

We cut one section power trace randomly from the sampled power trace as the following Fig. 8. We note that the conditional branch "if statement" in the binary method and the different power patterns of EC-operations depend on the key bits. Hence, the power trace also satisfies the second condition of Definition 1. Thus, according to Definition 1, recovering the key is feasible in this naive implementation of the binary method. Fig. 8 shows that key bit is "1" when ADD operation is followed by one DBL operation in one iteration between two dashed lines, whereas key bit is "0" when only one DBL operation in one iteration.

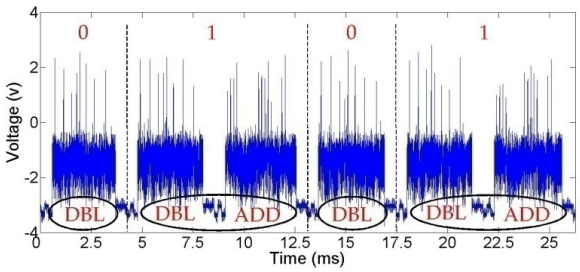

Fig. 8. Key-dependent pattern of binary method

10.4 Secure Double-and-add-always Method

According to the side-channel security analysis of Algorithm 2 in Section 6, we only need to substitute the unprotected binary method with side-channel secure scalar multiplication methods to resist SSCA. Taking double-and-add-always method for example, please see Algorithm 3, we only need to prove the side-channel security of double-and-add-always method. Please see Algorithm 4, according to the Definition 3, double-and-add-always method consists of regular operation iterations, which are key-independent executions.

Algorithm 4 (Double-and-add-always Method).

Input: $d = (d_k d_{k-1} ... d_2 d_1)_2$, an elliptic curve point P

Output: dP

$Q_0 = P$;

for$i = k$ - 1 to 1 do /*Scan d from MSB to LSB*/

{

$Q_0 = 2Q_0$; /* DBL */

$Q_1 = Q_0 + P$; /* ADD */

$Q_0 = Q_{d_i}$;

}

Return Q_0;

Experimental results are shown in Fig. 9, which presents the regular "DBL-ADD" power trace iterations of double-and-add-always method. Hence, the experiment result proves that double-and-add-always method is secure against SSCA.

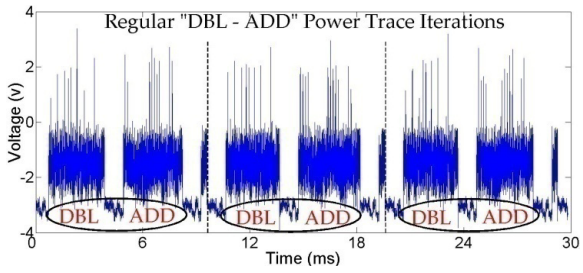

Fig. 9. Key-independent pattern of double-and-add-always method

11 Conclusions

In this paper we presented a fast and flexible parallel scalar multiplication method for the encryption and decryption of ECC on an arbitrary elliptic curve in Weierstrass form. Sequential scalar multiplication methods are too slow for parallel systems, and previous parallel scalar multiplication methods are limited to parallel systems with fixed number of processors. For these reasons, herein we proposed a fast and flexible parallel scalar multiplication method that can be adapted to various parallel systems. Implementation results indicate that our method can be implemented into various parallel systems and that can provide the better speedup effect. Side-channel experiments prove that our SSCA-protected version is secure against SSCA.

Since our proposed parallelization technique is on the scalar multiplication algorithm level not the elliptic curve group law, they also can be applicable for parallelization of exponentiation using partition and integration exponent in other public-key cryptosystems, such as RSA cryptosystem.

Acknowledgements

This work was supported by the Guangdong Natural Science Foundation (Grant No. 2014A030310299), and the Basic Research Project of Shenzhen (Grant No. JCYJ20160415113927863).

References

[1] V.S. Miller, Use of elliptic curves in cryptography, in: Proc. International Cryptology Conference on Advances in Cryptology, 1985.

[2] N. Koblitz, Elliptic curve cryptosystems, Mathematics of Computation 48(177)(1987) 203-209.

[3] D.E. Knuth, The Art of Computer Programming Volume 2: Seminumerical Algorithms (3nd edn), Tsinghua University Press, Beijing, 2002.

[4] D. Hankerson, A. Menezes, S. Vanstone, Guide to Elliptic Curve Cryptography, Springer-Verlag, London, 2004.

[5] K. Aoki, F. Hoshino, T. Kobayashi, H. Oguro, Elliptic curve arithmetic using SIMD, in: Proc. Fourth International Conference on Information Security, 2001.

[6] T. Izuand, T. Takagi, Fast elliptic curve multiplications with SIMD Operations, in: Proc. Fourth International Conference on Information Security, 2002.

[7] W. Fischer, C. Giraud, E.W. Knudsen, J.P. Seifert, Parallel scalar multiplication on general elliptic curve over Fp hedged against non-differential side-channel attacks, Cryptology ePrint Archive 2002(7) IACR, 2002. <http://eprint.iacr.org/2002/007/> (accessed 09.01.02).

[8] T. Izu, T. Takagi, A fast parallel elliptic curve multiplication resistant against side channel attacks, in: Proc. International Workshop on Practice and Theory in Public Key Cryptosystems, 2002.

[9] T. Izu, T. Takagi, Fast elliptic curve multiplication resistant against side channel attacks, IEICE Transactions on Fundamentals E88-A(1)(2005) 161-171.

[10] P.K. Mishra, Pipelined computation of scalar multiplication in elliptic curve cryptosystems, IEEE Transactions on Computers 55(8)(2006) 1000-1010.

[11] P. Longa, A. Miri, Fast and flexible elliptic curve point arithmetic over prime fields, IEEE Transactions on Computers 57(3)(2008) 289-302.

[12] A.O. Ahmed, F.A.S. Turki, B. Paul, Efficient elliptic curve parallel scalar multiplication methods, in: Proc. 8th International Conference on Computer Engineering & Systems (ICCES), 2014.

[13] A. Reza, R.M. Arash, Parallel and high-speed computations of elliptic curve cryptography using hybrid-double multipliers, IEEE Transactions on parallel and distributed systems 26(6)(2015) 1668-1677.

[14] P.C. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, in: Proc. International Cryptology Conference on Advances in Cryptology, 1996.

[15] P.C. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: Proc. International Cryptology Conference on Advances in Cryptology, 1999.

[16] J.S. Coron, Resistance against differential power analysis for elliptic curve cryptosystems, in: Proc. Workshop on Cryptographic Hardware and Embedded Systems, 1999.

[17] K. Wu, H. Li, D. Zhu, Fast and scalable parallel processing of scalar multiplication in elliptic curve cryptosystems, Security and Communication Networks 5(6)(2012) 648-657.

[18] ANSI, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), Accredited Standards Committee X9 Incorporated, Annapolis, 1999.

[19] IEEE, IEEE Standard Specifications for Public Cryptography, IEEE Std 1363-2000, IEEE Computer Society, District of Columbia, 2000.

[20] NIST, Digital Signature Standard (DSS), FIPS PUB 186-2, National Institute of Standards and Technology, Gaithersburg, 2000.

[21] N. Koblitz, CM-curves with good cryptographic properties, in: Proc. International Cryptology Conference on Advances in Cryptology, 1991.

[22] P.L. Montgomery, Speeding the pollard and elliptic curve algorithms for factorizations, Mathematics of Computation 48(177)(1987) 243-264.

[23] M. Joye, J.J. Quisquater, Hessian elliptic curves and side-channel attacks, in: Proc. Workshop on Cryptographic Hardware and Embedded Systems, 2001.

[24] E. Brier, M. Joye, Weierstrass elliptic curves and side-channel attacks, in: Proc. Public Key Cryptography 2002, LNCS 2274, 2002.

[25] O. Billet, M. Joye, The Jacobi model of an elliptic curve and side-channel analysis, in: Proc. International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 2003.

[26] B. Chevallier-Mames, M. Ciet, M. Joye, Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity, IEEE Trans. Computers 53(6)(2004) 760-768.

[27] K.K. Wu, H.Y. Li, D.J. Zhu, F.Q. Yu, Efficient solution to secure ECC against side-channel attacks, Chinese Journal of Electronics 20(3)(2011) 471-475.

[28] C.Y. Lu, S.M. Jen, C.S. Laih, A general framework of side-channel atomicity for elliptic curve scalar multiplication, IEEE Transactions on Computers 62(3)(2015) 428-438.