Parallel Scalar Multiplication for Elliptic Curve Cryptosystems

Bijan Ansari  
Dept of ECE  
University of Waterloo  
Waterloo, Ontario, Canada  
Email: bansari@vlsi.uwaterloo.ca

Huapeng Wu  
Dept of ECE  
University of Windsor  
Windsor, Ontario, Canada  
Email: hwu@uwindsor.ca

Abstract—In this paper, a parallel elliptic curve scalar multiplication algorithm is proposed. The algorithm works at its highest efficiency when a dual-processor hardware system is utilised. The new method has an average computation time of \( \frac{n}{2} T_{\text{ECADD}} \) times of that of one elliptic curve addition (ECADD) on an \( n \)-bit scalar. The saving is \( n \) elliptic curve doublings (ECDBLs) compared to the standard (single processor) methods. When the ratio of the execution time of a point addition to that of a point double is 3, the proposed method is 90\% faster than the standard method.

Keywords: Elliptic curve, cryptosystem, scalar multiplication, side channel attack

I. INTRODUCTION

Security has become an increasingly important issue for network communications and especially for e-commerce. In order to provide common network cryptographic functions such as key exchange and digital signature, one of the two popular public key cryptosystems, namely, RSA and elliptic curve (EC) systems, must be adopted by the security system. EC cryptosystem is preferred to RSA in many important applications such as smart card, since the former uses much shorter key than RSA for the same level of security strength [7]. Shorter key means less memory for storing the key and saving on bandwidth use.

The major computation involved in an elliptic curve cryptosystem is scalar multiplication over an elliptic curve. Thus, research and development of fast scalar multiplication methods is vital to improvement of efficiency of elliptic curve systems and the security services provided for the Internet.

It is well known that a scalar multiplication can be realized by a series of point addition (ECADD) and point doublings (ECDBL), which is called double-and-add method. If we denote the times for point addition and doubling as \( T_{\text{ECADD}} \) and \( T_{\text{ECDBL}} \), respectively, then for a \( n \)-bit scalar \( k \), the average time for computing the scalar multiplication \( kP \), where \( p \) is a point on the elliptic curve, is \( \frac{n}{2} T_{\text{ECADD}} + nT_{\text{ECDBL}} \). A number of improvements based on the double-and-add method have been proposed [5], [1], which will be briefly reviewed in the next section.

In this paper, a fast parallel elliptic curve scalar multiplication algorithm based on a dual-processor hardware system is proposed. The new method has an average computation time of \( \frac{n}{3} T_{\text{ECADD}} \) on an \( n \)-bit scalar. When a proper coordinate system and binary representation for the scalar \( k \) is used, the average execution time will be as low as \( nT_{\text{ECDBL}} \), which proves this method to be about two times faster than conventional single processor multipliers using the same coordinate system.

The organization of the rest of the paper is as follows. A brief review of the related work is given in Section 2. In Section 3, a new parallel scalar multiplication algorithm is proposed. The security of the proposed method against the side channel attack is discussed in Section 4. A few conclusion remarks are given in Section 5.

II. RELATED WORK

Algorithm 1 shows the double-and-add algorithm for computing the scalar multiplication, \( kP \), where \( k \) is a scalar and \( P \) is a point on the elliptic curve. The bit examination can also be done from the most significant bit (MSB first method).

Algorithm 1: The double-and-add algorithm
Input: A point \( P \), an integer \( k = \sum_{i=0}^{n-1} k_i2^i, k_i \in \{0,1\} \)
Output: \( Q = kP \)

1. \( Q := P; R := O \)
2. FOR \( i := 0 \) TO \( n-l \)
   2.1. IF \( k_i = 1 \) THEN \( R := R + Q \)
   2.2. \( Q := 2Q; \)
RETURN \( R \)

Assume that the average number of 1’s in \( n \)-bit \( k \) is \( \frac{n}{2} \). The execution time for Algorithm 1 is \( \frac{n}{2} T_{\text{ECADD}} + nT_{\text{ECDBL}} \) [1]. If redundant representation (i.e., binary non-adjacent form or NAF) is used to represent the scalar \( k \), the average number of ones or minus ones in the binary representation of \( k \) will be reduced to \( \frac{n}{3} \). In this case the average execution time is about \( \frac{n}{3} T_{\text{ECADD}} + nT_{\text{ECDBL}} \) [1], [3]. Further improvement in the speed can be achieved by the window method at the cost of precomputation and memory size [1]. Table I summarizes the execution time for these methods. Note in Table I that \( w \) denotes the window size in the window method. Other serial methods include addition Chain methods and Comb methods, which are very effective when \( k \) or \( P \) is known in advance, respectively. In comparison Window methods are efficient for most cases.

Constraints in scalar multiplications are speed, memory usage, and security against side channel attack (SCA). Faster
scalar multiplication methods can be obtained by introducing parallel computation of point operations. Note in Algorithm 1 that steps 2.1 and 2.2 are independent and thus they can be executed in parallel. In [6] Moller proposes a parallel algorithm for scalar multiplication which is fast and secure against side channel attack.

In this paper we propose a method which uses two processors and a circular buffer. The buffer acts as a communication channel between the two processors to reduce the average time of the scalar multiplication to \( nT \). In this way the total time for ECADD is saved and the system can be as fast as a system using radic NAF for Koblitz curves [4].

### III. PROPOSED PARALLEL SCALAR MULTIPLICATION METHOD

#### A. System Architecture

The proposed method for calculating \( kP \) uses two processors, one for execution of ECDBL and one for ECADD. The two processors may operate asynchronously. The ECDBL processor calculates \( 2^iP \) and stores them to a circular buffer. The ECADD processor reads from the circular and performs the addition. Figures 1 depicts the configuration of the dual-processor system.

The two processors share the circular buffer and a counter. The buffer can be a standard circular buffer and should provide empty and full flags.

#### B. Algorithm and Operations for ECDBL Processor

The ECDBL processor initially read the point \( P \) and then starts to keep performing doubling operations. It fills up the buffer with \( 2^iP \) whenever a nonzero \( k_i \) is detected and the buffer is not full. If the data in the buffer are not consumed by the ECADD processor the buffer becomes full and the ECDBL processor needs to wait until there is free room in the buffer.

<table>
<thead>
<tr>
<th>Method</th>
<th>Average execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary [1]</td>
<td>((n - 1)T_{ECDBL} + \frac{n}{2}T_{ECADD})</td>
</tr>
<tr>
<td>Binary NAF [3]</td>
<td>((n - 1)T_{ECDBL} + \frac{n}{2}T_{ECADD})</td>
</tr>
<tr>
<td>Window [1]</td>
<td>(nT_{ECDBL} + \frac{n}{2}T_{ECADD})</td>
</tr>
</tbody>
</table>

**TABLE I**

On the other hand if there is not enough ones in the binary representation of \( k \), the buffer becomes empty after a while and ECADD processor needs to wait until data is put into the buffer by ECADD processor. Dual port RAM/registre should be used so that both processors can have simultaneous access to the buffer.

**Algorithm 2**: Parallel Scalar Multiplication: the Part for ECADD processor

Input: A point \( P \), an integer \( k = \sum_{i=0}^{n - 1} k_i2^i, k_i \in \{0, 1\} \)

Output: \( 2^iP \), stored in the buffer

Global: \( i, \) buffer

1. \( Q := P; i := 0; \)
2. WHILE \( i < n \)
   1. IF \( k_i = 1 \) THEN \( (\text{wait}) \)
      IF (buffer is full) THEN (wait)
      ELSE (write \( Q \) into buffer)
   2.2. \( Q := 2Q; i := i + 1 \);

#### C. Algorithm and Operations for ECADD Processor

The ECADD processor takes input only from the buffer. When there are multiple data in the buffer, it read the one that ECDBL put there first. Immediately after reading a data the ECADD processor erases the data from the buffer. The final scalar multiplication is achieved when ECDBL processor stops computing and writing into the buffer and ECADD reads all the data in the buffer.

**Algorithm 3**: Parallel Scalar Multiplication: the Part for ECADD processor

Input: \( 2^iP \), read from the buffer

Output: \( R = kP \)

Global: \( i, \) buffer

1. \( R := O; \)
2. WHILE \( i < n \)
   2.1. IF (buffer is not empty) THEN
       read \( 2^iP \) from the buffer;
       \( R := R + 2^iP; \)
       erase \( 2^iP \) from the buffer;

#### D. Performance of the Parallel Algorithm

 Apparently, by using the proposed method a scalar multiplication can be computed at the optimal speed, compared to any other methods that are based point addition and doubling operations. The performance of the algorithm depends on the ratio of \( r = T_{ECADD}/T_{ECDBL} \) and the occurrences of nonzero digits in the binary representation of the scalar \( k \). Assume that the NAF for \( k \) is used for both the standard single processor method and the proposed parallel method, simulation results of the algorithm are summarized in Table II. Note in the table that \( n \) is the binary length of \( k \), \( r = T_{ECADD}/T_{ECDBL} \).

\( T_{NAF} \) denotes the execution time for \( kP \) using the proposed method, and \( T_{NAF} \) denotes the execution time for \( kP \) using the standard NAF method based on single processor. Values of \( n \) are chosen as 150, 200, 250, and 300, which are considered in the range of current popular implementation of elliptic curve cryptosystems [2].
The results show that the maximal speed up is achieved when the execution time of a point addition is three times that of a point doubling operation. In this case, it requires that the circular buffer size is $4$, or the buffer is large enough to hold $4$ elliptic curve points.

### E. Security Against Side Channel Attack (SCA)

The execution time of the algorithm depends on the scalar integer $k$. For example if $k$ has a very low Hamming weight, the execution time for $kP$ will be close to $nT_{\text{ECDBL}}$. In case of $k = 101010\ldots101010$ the execution time will be about $(n/2)T_{\text{ECADD}}$. Therefore the algorithm cannot be immune to SCA. But, since the execution time depends on the total number of ones and on the distribution of ones, many values of $k$ will have the same execution time. Therefore the algorithm offers better security against SCA when compared to the standard double-and-add methods.

### IV. Conclusions

A parallel method for scalar multiplication is introduced which uses two processors to perform the $kP$ operation. Using proper implementation this method is $90\%$ faster than the single processor methods.

Since the ratio $r$ depends on the coordinate system in which the elliptic curve calculation is performed, further research might include application of the proposed methods to computation of scalar multiplication using different coordinate systems.

### Acknowledgments

This work was supported by the National Sciences and Engineering Research Council of Canada (NSERC). The major part of this work was part of Mr.Ansari’s Master thesis in the Department of Electrical and Computer Engineering, University of Windsor.

### References


<table>
<thead>
<tr>
<th>$n$</th>
<th>$r$</th>
<th>$T_{\text{new}}$ ($T_{\text{ECDBL}}$)</th>
<th>$T_{\text{NAF}}$ ($T_{\text{ECDBL}}$)</th>
<th>Speed up</th>
<th>Max # of points in buf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>1</td>
<td>150</td>
<td>200</td>
<td>1.3</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>200</td>
<td>266</td>
<td>1.3</td>
<td>1</td>
</tr>
<tr>
<td>250</td>
<td>1</td>
<td>250</td>
<td>335</td>
<td>1.3</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>1</td>
<td>300</td>
<td>400</td>
<td>1.3</td>
<td>1</td>
</tr>
<tr>
<td>150</td>
<td>2</td>
<td>130</td>
<td>250</td>
<td>1.7</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>2</td>
<td>200</td>
<td>335</td>
<td>1.7</td>
<td>1</td>
</tr>
<tr>
<td>250</td>
<td>2</td>
<td>250</td>
<td>416</td>
<td>1.7</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
<td>300</td>
<td>500</td>
<td>1.7</td>
<td>1</td>
</tr>
<tr>
<td>150</td>
<td>3</td>
<td>160</td>
<td>300</td>
<td>1.9</td>
<td>4</td>
</tr>
<tr>
<td>200</td>
<td>3</td>
<td>213</td>
<td>400</td>
<td>1.9</td>
<td>4</td>
</tr>
<tr>
<td>250</td>
<td>3</td>
<td>266</td>
<td>500</td>
<td>1.9</td>
<td>4</td>
</tr>
<tr>
<td>300</td>
<td>3</td>
<td>320</td>
<td>600</td>
<td>1.9</td>
<td>4</td>
</tr>
<tr>
<td>150</td>
<td>4</td>
<td>242</td>
<td>350</td>
<td>1.4</td>
<td>4</td>
</tr>
<tr>
<td>200</td>
<td>4</td>
<td>325</td>
<td>466</td>
<td>1.4</td>
<td>4</td>
</tr>
<tr>
<td>250</td>
<td>4</td>
<td>409</td>
<td>583</td>
<td>1.4</td>
<td>4</td>
</tr>
<tr>
<td>300</td>
<td>4</td>
<td>492</td>
<td>700</td>
<td>1.4</td>
<td>4</td>
</tr>
</tbody>
</table>

*TABLE II*  
Simulation results for the proposed parallel algorithm and the standard NAF method.