Addition-Subtraction Chain for 160 bit Integers by using
2’s Complement

Mohamed M. Abd-Eldayem
Associate Professor,
Department of Computer Engineering
College of Computer and Information Sciences
King Saud University
Riyadh, Saudi Arabia
mdayem@ksu.edu.sa

Ehab T. Alnrawy
Assistant Lecturer,
Computer Science and
Information Systems Department
Sadat Academy, Alexandria branch, Egypt
alnrawy@gmail.com

Aly A. Fahmy
Professor,
Computer Science Department
Faculty of Computers and Information
Cairo University, Giza, Egypt
a.fahmy@fci-cu.edu.eg

Abstract

Speeding up the scalar multiplication k.P is the major point in elliptic curve cryptosystem efficiency. The Addition-Subtraction (AS) Chain is efficient tool to do that where integer k is fixed and Point P is variable. Since the optimization problem to find the shortest AS Chain is NP-hard, many algorithms to get sub-optimal AS Chain in polynomial time are proposed. The window method is a class of theses algorithms. In this paper an algorithm for finding a shorter AS Chain is proposed. This algorithm is based on window method with small width by using 2’s Complement for 160 bit integers. It is shown practically that proposed algorithm more efficient than the last result in the subject with 20% improvement.

Keywords: Addition-Subtraction Chain, Window Method, Scalar Multiplication, Elliptic Curve Cryptosystems.

1. Introduction

Elliptic Curve Cryptography (ECC) was proposed independently by Koblitz [1] and Miller [2] in 1985. It facilitates two parties to generate a secret key for communication across an insecure channel. The most fascinating feature of Elliptic Curve Cryptography is that it utilizes much smaller key of size 160 bits to provide the same level of security as other cryptographic standards such as RSA of 1024 bits [3]. The strength of Elliptic Curve Cryptosystems (ECC) is based on the complexity of the elliptic curve discrete logarithm problem. The major building block of ECC is the computation of the form k.P known as the scalar multiplication or exponentiation, where k is a positive long integer (a secret scalar) and P is a point on an elliptic curve. This curve defined over a finite field. Therefore, efficient and secure exponentiation methods are vital in ECC. The techniques and algorithms for exponentiations in the context of elliptic curve cryptography are described in [4].

An addition chain is used to represent the computation sequence of modular multiplications for a modular exponentiation. A shorter addition chain means faster execution of the corresponding modular exponentiation, because there is one-to-one
relationship between an element of the addition chain and a modular multiplication in the process of the modular exponentiation. In addition, the addition chain is used to reduce the execution time of k.P, on an elliptic curve cryptosystem. Calculation of k.P is composed of repetition of additions and computation sequence of additions can be represented by an addition-chain. On an elliptic curve cryptosystem, the subtraction of a point from another point on the curve requires the same execution time as the corresponding addition, because negation of a point is very easy. Therefore, it is possible to use subtractions to find the computation sequence of a scalar multiplication k.P over elliptic curves. An excellent technical and historical account of exponentiation and the addition chain problem is given by Knuth [5] who traces the problem back to 200 BC. A survey carried by Gordon [6] describes various fast methods, including some specialized to elliptic curve groups.

Our contribution will deal with using 2’s Complement in constructing Addition-Subtraction Chain. This is not the first time for using 2’s Complement in this manner. It used before in [7] to construct Addition-Subtraction chain for 512 bit integers. We developed a new algorithm for 160 bit integers to get more better efficiency scalar multiplication and also with sufficient security[3].

The rest of this paper is organized as follows. In Section 2 we give the brief survey for related works of A and AS Chains. In section 3 the algorithm, its running time analysis, the proof of chain length, and the correctness proof are provided. In Section 4 Implementation conditions, a discussion of our experimental results, and a comparisons are explained. In Section 5 the final conclusion of this paper is established.

2. Related Work

In [7], the authors introduce a new lower length AS chain for 512 bit integers by using 2’s complement.
In [8], the authors a new result about finding shorter AS chain for 160 bit integers by using Fibonacci sequences and golden ratio.
In [9], the authors obtain the minimal AS Chain by using ant colony
In [10], the authors obtain the minimal AS Chain by using genetic algorithms.
Practically, we cannot use the minimal AS Chain evaluation in cryptography. This because the main using of ECC is in verification manner. Which means we need to store the result chain for using it again. This not practically in multi user verification system. In [11], generalize the Bruaer method for AS Chain
In [12], Canonical Signed Bits representation used to obtain AS Chain
In [13], using the idea of substitution long block of 1’s by the same length block of 0’s and one subtraction.
Together with these studies, there are many proposals for AS algorithms. The intuitive binary method has been used for a long time [5], and the modified binary method using modulo inverse of a number was proposed in [14] and [15]. Also, the simple and efficient
small-window method is presented in [5]. Bos and Coster proposed the large-window method using some heuristics [16]. Yacobi proposed a modified m-ary algorithm which uses similarities between data compression and operation that deals with large numbers in [17]. Koyama and Tsuruoka proposed a signed binary window method in [18]. There is performance comparison of these various algorithms in Table 2.

3 The Proposed Addition-Subtraction (AS) Chain Algorithm

3.1 Background

We will introduce the required definitions and facts to use it in our algorithm.

Definition 3.1 An addition chain computing an integer $k$ is given by two sequences $v = (v_0, ..., v_l)$ and $w = (w_1, ..., w_l)$ such that $v_0 = 1$, $v_l = k$, $v_i = v_r + v_s$, for all $1 \leq i \leq l$ with respect to $w_i = (r, s)$ and $0 \leq r, s \leq l - 1$. The length of the addition chain is $l$.

Definition 3.2 An Addition-Subtraction (AS) chain is similar to an addition chain except that the coordinate $v_i = v_r + v_s$ is replaced by $v_i = v_r + v_s$ or $v_i = v_r - v_s$.

Definition 3.3 $T$ will be 2’s complement of integer $t$ i.e. it is equal to the subtraction of $t$ from $w$ power of 2 since $w$ is the binary length of $t$ ($T = 2^w - t$). This for any value of $t$ except zero. For $t = 0$, $T = 0$.

3.2 The proposed algorithm

The core idea of this algorithm is using 2’s complement in building Addition-Subtraction (AS) chain.

Firstly, the algorithm pre-compute the elements of the target chain $1, 2, 3, 5, ..., 2^{w-1}$ (only odd numbers in addition to 2).

Secondly, Constructing the windows: We scan the binary representation of $k$ from right to left. We construct each window with using sub-string of width $w$ (constant) from the whole integer representation. During this stage, if we find the following window (the next $w$ bits i.e the left $w$ of the current window) starts with zero from right then:

1. we substitute the current window with its 2’s Complement value of it,
2. we add 1 to the decimal value of the following window, and
3. and put a flag with 1 for subtraction.

Thirdly, Appending these windows values in the target chain.

Fourthly, Scanning the created windows from left to right by the following manner:
Algorithm 1: AlnASC : Using 2’s Complement To Produce Addition-Subtraction (AS) Chain

**Input**: \(k = (k_{n-1}, k_{n-2}, ..., k_1, k_0)\) binary representation of \(n\) bit integer, \(w\) the width of used window

**Output**: Chain = 1, 2, 3, ..., \(2^w - 1\), ..., \(k\)

1. Procedure of calculating the difference between addition and doubling operation, let \(d = \text{addition} - \text{doubling} \); 
2. \(\text{windex} = 0\); \(\text{index} = 0\); \(\text{cindex} = 1\);
3. Chain[0] = 1;
4. Chain[cindex] = Chain[cindex - 1] + Chain[cindex - 1]; cindex = cindex + 1;
5. Chain[cindex] = Chain[cindex - 1] + Chain[cindex - 2]; cindex = cindex + 1;
6. while Chain[cindex] \(\leq 2^w - 1\) do 
7. \hspace{1em} Chain[cindex] = Chain[cindex - 1] + Chain[1]; cindex = cindex + 1;
8. end
9. while index \(\leq n - 1\) do 
10. \hspace{1em} \(\text{wv}[\text{windex}] = \text{decimal value of w bits} (k_{\text{index}+w-1}, ..., k_{\text{index}}); 
11. \hspace{1em} \text{if} \ k_{\text{index}+w} = 0 \text{ then} 
12. \hspace{2em} \text{wv}[\text{windex} + 1] = \text{wv}[\text{windex} + 1] + 1;
13. \hspace{1em} \text{wv}[\text{windex}] = 2^w - \text{wv}[\text{windex}]; 
14. \hspace{1em} \text{sub}[\text{windex}] = 1;
15. end
16. \hspace{1em} \text{Procedure of insert} \text{wv}[\text{windex}] \text{ into the Chain by insertion sort and put the position of it in wpos[wind ex]} ; 
17. \hspace{1em} \text{index} = \text{index} + w ; \text{windex} = \text{windex} + 1;
18. end
19. llwi = windex - 1 ;
20. Chain[cindex] = Chain[wpos[llwi]] + Chain[wpos[llwi]]; cindex = cindex + 1;
21. for \(t = 2\) to \(w\) do 
22. \hspace{1em} Chain[cindex] = Chain[cindex] + Chain[cindex]; cindex = cindex + 1;
23. \hspace{1em} delay(d);
24. end
25. for \(i = llwi - 1\) down to 0 do 
26. \hspace{1em} \text{if} \ sub[i] = 0 \text{ then} 
27. \hspace{2em} Chain[cindex] = Chain[cindex - 1] + Chain[wpos[i]]; cindex = cindex + 1;
28. \hspace{1em} \text{else} 
29. \hspace{2em} Chain[cindex] = Chain[cindex - 1] - Chain[wpos[i]]; cindex = cindex + 1;
30. \hspace{1em} end
31. \hspace{1em} for \(t = 1\) to \(w\) do 
32. \hspace{2em} Chain[cindex] = Chain[cindex - 1] + Chain[cindex - 1]; cindex = cindex + 1;
33. \hspace{2em} delay(d);
34. \hspace{1em} end
35. end
1. Doubling the current window w times. with each doubling appending the new element to the target chain

2. check if the flag of the next window is 0 then we add the next window to the last element created in the chain. Otherwise we subtract the next window from the last created element in the chain. Then go to the previous step with next window as new current window.

3.3 Analysis of proposed algorithm

Theorem 3.1 The algorithm 1 is correct.

Proof: Let y be a binary string of the long integer representation k. This means y = (y_i, y_{i-1}, ... y_j) since y_i = k_i \in (k_{n-1}, k_{n-2}, ..., k_1, k_0). We can easily deduce from the above algorithm that we substitute each y by \(2^k - \tilde{y}\) since \(\tilde{y}\) is the two’s complement of y. We add one to the left window instead of \(2^k\). And put flag subtract to indicate subtraction.

Theorem 3.2 Algorithm 1 is in O((n(2^{w-1})/w) + 3/2).

Proof: We will concentrate on the main loops to find the complexity of this algorithm and neglect the computation on the regular integers and we will concern with long integer computation. According to the algorithm, It contain the following main steps

1. In line 6 We have a loop of computing the start part of the Chain. This loop will repeated \(2^{w-1}\) iterations. Each iteration include n bit additions. This means that this loop will cost n.2^{w-1} bit operations.

2. In line 9, the loop for creating the windows is provided. This loop will repeated n/w times. Each iteration will include 2 assignment statement with w addition bit operations. So, the loop will cost 2n/w operations. We note here the small operations are neglected like addition by one.

3. In line 21, loop has w − 1 repetitions.

4. In line 25, loop has n/w repetitions.
   
   • In Line 26, consume n operations.
   • In Line 31, loop has w repetitions.

5. insertion the result windows in the chain, will take (n/w)((2^w − 1/2) + 1) assignment operations in worst case by insertion sort.

6. loop of concatenation the windows in each others, This loop will repeated n/w times.
   Each iteration will take w operations. Then this loop totally takes n operations.
The main component in the running time is \((n/w) \times ((2^w - 1/2) + 1) = (n(2^{w-1})/w) + 3/2\) 

Theorem 3.3 Algorithm 1 produce Addition Subtraction Chain of length \((n/w)(w + 2) + 2^{w-1} + w + 3/2\) in worst case.

Proof:
From the algorithm:

- the precomputed chain part contains \((2^w - 1/2) + 1\) elements.
- After composing the windows, we must insert \(n/w\) elements into the chain in the worst case.
- Before the last loop in line ??, we will add \(1 + w\) elements.
- In each iteration of the loop, we will add \(1 + w\) elements. This loop will repeated \(n/w\) times.

So we get the target expression in the theorem.

<table>
<thead>
<tr>
<th>Window length</th>
<th>Addition</th>
<th>Subtraction</th>
<th>Chain length</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>238</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>215</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>203</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>215</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>241</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>427</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>678</td>
</tr>
</tbody>
</table>

4. Experimental Results

To compare our result, we implement them by using a notebook with Intel (R) Core (TM)2 Duo CPU T6570 @2.1GHz 1.19 GHz, 1.92 GB of RAM. And the platform is Windows XP Service Pack 3.

We use library Miracl[19] in the implementation. MIRACL is a Big Number Library which implements all of the primitives necessary to design Big Number Cryptography into your real-world application. It is primarily a tool for cryptographic system implementors. RSA public key cryptography, Diffie-Hellman Key exchange, DSA digital signature, they are all just a few procedure calls away. Support is also included for even more esoteric Elliptic Curves and Lucas function based schemes. The latest version offers
full support for Elliptic Curve Cryptography. It has two options use C or C++. For efficiency reasons, we use C option in the library.

We compare our result here by result in [8] because its last good known result in application to find Addition-Subtraction Chain for 160 bit integers. The introduced algorithm produce a chain of length 203 elements. Which is shorter than 258 in [8]. Table 2 shows the result of our implementation with comparison to other algorithms, We get that our improvement is about 20% since the last shorter known published length in [8] is 258.

In the table we refer to the source of implementation and algorithm itself. For non-referred items, this means the algorithm or implementation created by this work. Algorithms of GRASC, Window Fibonacci and Add, Signed Fibonacci and Add, Fibonacci and Add and EAC different from the others because its doubling free algorithms. Our Trend using a dummy operation to erase the different between addition and doubling which is the major aim of SPA attack.

### 4.1 Selecting window width

To implement the AlnASC algorithm, we must select the width of the window used in the algorithm implementation. In our experiments we seeking the best window to get the shortest addition subtraction chain. In Table 1, We provide the results of different window width for long integers of length 160 bits.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Length of Chain</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlnASC Addition-Subtraction Chain</td>
<td>203</td>
</tr>
<tr>
<td>Slide Window Addition Chain (constant window)</td>
<td>214</td>
</tr>
<tr>
<td>Slide Window Addition Chain (variable window)</td>
<td>223</td>
</tr>
<tr>
<td>Binary Addition Chain [4]</td>
<td>239</td>
</tr>
<tr>
<td>GRASC[8]</td>
<td>258 [8]</td>
</tr>
<tr>
<td>EAC[21]</td>
<td>320 [21]</td>
</tr>
<tr>
<td>Fibonacci and Add[21]</td>
<td>358 [8]</td>
</tr>
</tbody>
</table>

### 5. Conclusions

Table 2 shows the result of our implementation with comparison to other algorithms, We get that our improvement is about 21% since the last shorter known published length in [8] is 258.

We can direct the research in more than direction. Find a relation between the positions of zeros and ones and its numerical value to use them in scanning the integer
and decide the addition and subtraction positions. Also using the variable length window may will provide good improvement.

References


