Expansion of Sliding Window Method for Finding Shorter Addition/Subtraction-Chains

Younho Lee, Heeyoul Kim, Seong-Min Hong, and Hyunsoo Yoon
(Corresponding author: Seong-Min Hong)

Division of Computer Science, Korea Advanced Institute of Science and Technology
373-1 Guseng-dong, Yuseong-gu, Daejeon 305-701, Rep. of Korea (Email: smhong@camars.kaist.ac.kr)

(Received July 3, 2005; revised and accepted Aug. 2, 2005)

Abstract

Finding a shorter addition/subtraction-chain for an integer is an important problem for many cryptographic systems based on number theory. Especially, execution time of multiplication on an elliptic curve cryptosystem is directly proportional to the length of the addition/subtraction-chain. In this paper, we propose an algorithm to find an addition/subtraction-chain. The proposed algorithm is based on the small-window method, and reduces the number of windows by using subtractions. We show the proposed algorithm finds the shorter addition/subtraction-chain than what can be found by any other previous algorithm.

Keywords: Addition/subtraction-chain, elliptic curve cryptosystem, public key cryptosystem

1 Introduction

Since Diffie and Hellman had proposed public-key cryptography in 1976, many cryptosystems based on number theory have been developed [3, 4]. They have to deal with large numbers as 512-bit integers to gain acceptable security. As it takes much time to deal with such large numbers on current computer systems, we need algorithms which can execute cryptographic functions fast. Modular exponentiation in RSA and multiplication on an elliptic curve cryptosystem are ones of such operations [6, 9, 10].

A modular exponentiation is composed of many modular multiplications. 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.

Also, an addition-chain is used to reduce the execution time when we calculate the d multiple of a point P, that is $d \cdot P$, on an elliptic curve cryptosystem. Calculation of $d \cdot P$ is composed of repetition of additions and computation sequence of additions can be represented by an addition-chain. On an elliptic curve cryptosystem, subtraction of a point from another point requires the same time as the corresponding addition, because negation of a point is very easy. Therefore, we can use subtractions when we find the computation sequence of a multiplication over elliptic curves. That computation sequence can be represented as an addition/subtraction-chain. An *addition/subtraction-chain* of length l for an integer n is a sequence of integers $a_0, a_1, ..., a_l$ satisfying $a_0 = 1$, $a_l = n$, and $a_i = \pm a_j \pm a_k$, where $0 \leq j \leq k < i \leq l$.

Many researchers have studied about addition(/subtraction)-chains for their cryptographic importance [1, 2, 7, 9, 13, 14, 15]. In these researches there are some remarkable investigations. Downey et al. proved that finding the shortest addition-chain containing a set of integers is an NP-complete problem in [5]. Schonhage revealed the fact that the lower bound of the shortest length of an addition-chain for an integer a is $\log a + \log \nu(a) - 2.13$ in [11] ($\nu(a)$ is the number of 1's in the binary representation of a). Together with these studies, there are many proposals for addition-chain algorithms. The intuitive binary method has been used for a long time [8], and the modified binary method using modulo inverse of a number was proposed in [7] and [12]. Also, the simple and efficient small-window method is presented in [8]. Bos and Coster proposed the large-window method using some heuristics [1]. Yacobi proposed a modified m-ary algorithm which uses similarities between data compression and operation that deals with large numbers in [15]. Koyama and Tsuruoka proposed a signed binary window method in [9]. There is performance comparison of these various algorithms in Table 1.

In this paper, we propose an algorithm to find an addition/subtraction-chain. The proposed algorithm is

based on the small-window method, and reduces the number of windows by using subtractions. We show the proposed algorithm finds the shorter addition/subtraction-chain than what can be found by any other previous algorithm.

Composition of this paper is as follows. In Section 2, we propose an addition/subtraction-chain algorithm. In Section 3, we calculate the length of the chain which is found by the proposed algorithm. In Section 4, we compare the performance of the proposed algorithm with those of previous addition/subtraction-chain algorithms. Finally, we conclude in Section 5.

2 Algorithm

In this section, we propose an addition/subtraction-chain algorithm. First, we explain the small-window method [8] briefly, because our algorithm is based on it. And then, we explain our algorithm.

2.1 Small-Window Method

With the small-window method, we write a number in the binary scale, and split it in many small pieces(windows). After that, we find the addition-chain by merging those windows.

For example, we find the addition-chain for a small integer "3584965235_{10}" with the small-window method. The binary representation of "3584965235_{10}" is as follows:

$$11010101101011100011101001110011.$$

We can divide it into the following partition with windows of which maximum size is 4.

$$\underline{1101}\ 0\ \underline{1011}\ 0\ \underline{1011}\ \underline{1}\ 000\ \underline{111}\ 0\ \underline{1001}\ \underline{11}\ 00\ \underline{11}$$

First, all values that a window may have should be included in the addition-chain. That is, $\{1, 2, 3, 5, 7, ..., 13, 15\}$ should be included. Note that 2 must be included in the set, since we are not able to calculate 3,5,7,...,13,15 without it.

We start with the first(from the left) window "$\underline{1101}$". First, we shift "$\underline{1101}$" five times to the left. Then, it becomes "$\underline{1101}\ 00000$". In this process, $\{13 \times 2, 13 \times 2^2, 13 \times 2^3, 13 \times 2^4, 13 \times 2^5\}$ is inserted into the addition-chain. Second, we add the second(from the left) window "$\underline{1011}$" to "$\underline{1101}\ 00000$". Then, it becomes "$\underline{1101}\ 0\ \underline{1011}$", which corresponds to the most significant 9-bit-string of the number of which addition-chain we want to find. The decimal number equivalent to it, $427_{10}(= 13 \times 2^5 + 11)$, is inserted into the addition-chain.

We call the process explained in the above paragraph *window merge*. Merging all eight nonzero windows, we can get an addition-chain. This procedure of finding an addition-chain with the small-window method is described in *C-like* pseudo-code in Algorithm 2.1.

We calculate the length of the addition-chain found above by counting the elements in the addition-chain. First, $9(= 2^{4-1} + 1)$ elements are inserted into the chain, because a window of which maximum size is 4 may have an odd integer smaller than $16(= 2^4)$ and 2 should be included in the addition-chain to calculate these integers. Second, $28(= 32 - 4)$ elements are inserted into the chain, because the first window should be shifted 28 times to the left and one shift means that one element should be inserted into the chain. Third, $7(= 8 - 1)$ elements are inserted into the chain, because the number of windows is 8 and merging two windows means that one element should be inserted into the chain. The number of all elements in the addition-chain is $44(= 9 + 28 + 7)$, and the length of chain is the number of elements in the chain minus one [8]. Therefore, the length of the addition-chain found in the example is 43.

Algorithm 2.1. *Let* $a(= e_{n-1}e_{n-2}\cdots e_1e_0,\ e_i = 0$ *or* $1,\ 0 \le i < n)$ *be the number of which addition-chain we want to find, and let* k *be the window size. In this algorithm,* α *is the addition-chain represented as a set,* w_j *means each window, '|' concatenation.*

```
1   Small_Window( a,k )
2   {
3       α = {1, 2, 3, 5, 7, 9, ..., 2^k − 1};
4       i = n − 1; p = 0;
5       while( i ≥ 0 ) {
6           w_p = e_i e_{i−1} e_{i−2} ··· e_{i−k+1};
7           i = i − k;
8           for( ; e_i == "0"; i − − )      w_p = w_p|e_i;
9           p + +;
10      }

    /* Now, a = w_0 w_1 ··· w_{p−2} w_{p−1}, w_i = e_{i,s_i−1} e_{i,s_i−2}
    ··· e_{i,1} e_{i,0}, 0 ≤ i < p,k ≤ s_i */

11      while( w_0 > 2^k )      w_0 = w_0 / 2;
12      if( w_0 == even )
13          α = α∪{w_0 − 1, w_0, 2^1 × w_0, 2^2 × w_0,
                                        ···, 2^{s_0−k} × w_0};
14      else
15          α = α∪{w_0, 2^1 × w_0, 2^2 × w_0, ..., 2^{s_0−k} × w_0};
16      ac = 2^{s_0−k} × w_0
17      for( i = 1; i < p; i + + ) {
18          while( w_i == even )      w_i = w_i / 2;
19          α = α∪ {2^1 × ac, 2^2 × ac, ..., 2^{⌊log w_i⌋+1} × ac};
20          ac = 2^{⌊log w_i⌋+1} × ac + w_i;
21          α = α∪ {ac, 2^1 × ac, 2^2 × ac,
                                        ···, 2^{s_i−⌊log w_i⌋−1} × ac};
22          ac = 2^{s_i−⌊log w_i⌋−1} × ac
23      }
24  }
```

2.2 Proposed Algorithm

In this section, we explain our addition/subtraction-chain algorithm. It is based on the small-window method, and reduces the number of windows.

2.2.1 Basic Idea

We show the basic idea of the proposed algorithm with a very small example. Let the window size 3. To find an addition-chain for an integer "1387_{10}", we write it in the binary scale and split it in many windows. The result is "$\underline{101}\ 0\ \underline{11}\ 0\ \underline{101}\ \underline{1}$", and can be represented as follows:

$$((\underline{101} \times 2^3 + \underline{11}) \times 2^4 + \underline{101}) \times 2^1 + \underline{1}.$$

8 shifts and 3 window merges are required to find the chain, and the length of the resulting chain is $15(= 5 + 8 + 3 - 1)$. The resulting chain can be enumerated as follows:

$$\underline{1, 2, 3, 5, 7}, 10, 20, 40, 43, 86, 172, 344, 688, 693, 1386, 1387.$$

The underlined elements in the above chain are ones which are prepared in advance, because the window may have those values. Although some of them are needless in this example, they must be included since we have to deal with large numbers as 512-bit integers in real environment.

If subtractions are permitted, we can obtain a shorter chain(addition/subtraction-chain). "1011"(the least significant 4 bits of the binary representation of "1387_{10}") equals to "$10000 - 101$". Therefore, the window partition can be converted into "$\underline{101}\ 0\ \underline{111}\ 0\ /\underline{101}$" ("/" means that we must subtract the value of the succeeding window of the slash from that of the preceding window when we merge windows later). Note that the third window(from the left) starts from the fourth bit from the leftmost bit of the second window, and that the value of the second window is incremented by one. Also, note that three bits from the start bit of the third window are 2's complemented. It can be represented as follows:

$$(\underline{101} \times 2^4 + \underline{111}) \times 2^4 - \underline{101}.$$

The number of window merges required is reduced by one, and the length of the resulting chain becomes $14(= 5+8+2-1)$. The resulting chain can be enumerated as follows:

$$\underline{1, 2, 3, 5, 7}, 10, 20, 40, 80, 87, 174, 348, 696, 1392, 1387.$$

Now, we can formulate our idea, which is as follows:

> If the $(k+1)$-th bit from the leftmost bit of the i-th window is 1, the $(i+1)$-th window can start from the $(k+2)$-th bit, where k is the window size.

This observation can be generalized easily.

> If all bits from the $(k+1)$-th bit(from the leftmost bit of the i-th window) to the $(k+j)$-th bit are 1, the $(i+1)$-th window can start from the $(k+j+1)$-th bit, where k is the window size.

2.2.2 Small Example

We explain our algorithm with the example that we used in Section 2.1 for comparison. Let the window size 4. We split the binary representation of "3584965235_{10}" in windows. The intermediate result is as follows:

$$\underline{1101}\ 0\ \underline{1011}\ 0\ \underline{1011}\ 100011101001110011.$$

Here, the 5-th bit from the leftmost bit of the third window is 1. Therefore, the value of the third window is incremented by one, and the fourth window starts from the 6-th bit from the leftmost bit of the third window. Four bits from the start bit are 2's complemented. The resulting partition is as follows:

$$\underline{1101}\ 0\ \underline{1011}\ 0\ \underline{11}\ 000\ /\underline{1111}\ 1101001110011.$$

Again, the 5-th bit and 6-th bit from the leftmost bit of the fourth window are all 1. Therefore, the value of the fourth window(equals to -15) is incremented by one, and the fifth window starts from the 7-th bit from the leftmost bit of the fourth window. Bits from the 7-th to the 10-th from the leftmost bit of the fourth window are 2's complemented. The resulting partition is as follows:

$$\underline{1101}\ 0\ \underline{1011}\ 0\ \underline{11}\ 000\ /\underline{111}\ 000\ /\underline{11}\ 001110011.$$

The same process makes the following result:

$$\underline{1101}\ 0\ \underline{1011}\ 0\ \underline{11}\ 000\ /\underline{111}\ 000\ /\underline{1011}\ 000\ /\underline{1101}.$$

The method of obtaining an addition/subtraction-chain from this window partition is much the same as that of the small-window method. The only difference is that when a slash appears in the process of window merge, we subtract the value of the succeeding window instead of adding it. The procedure explained until now is described in Algorithm 2.2 in *C-like* pseudo-code.

We calculate the length of the addition/subtraction-chain found above, and compare it with that of the small-window method in Section 2.1. The number of elements prepared in advance and the total number of shifts are the same as those of the small-window method, so these are 9 and 28 respectively. The number of windows is different. It is 6 which is smaller than that of the small-window method by 2. Therefore, the total length of the addition/subtraction-chain obtained above is 41(=9+28+5-1). It is smaller than that of the small-window method by the difference in the number of windows.

2.2.3 Complexity

We briefly show the complexity of the proposed algorithm. In Algorithm 2.2, our pseudocode scans the input bit string a once(lines from 5 to 18), and then scans windows once(lines from 25 to 33). The number of bits in a is $\lfloor \log a \rfloor + 1$, and the number of windows is less than that. Therefore, the complexity of the proposed algorithm is $O(\log a)$, which is the same as that of Algorithm 2.1.

Algorithm 2.2. *Let* $a(= e_{n-1}e_{n-2}\cdots e_1e_0,\ e_i = 0$ *or* $1,\ 0 \le i < n)$ *be the number of which an addition/subtraction-chain we want to find, and let* k *be the window size. In this algorithm,* α *is the addition/subtraction-chain represented as a set,* w_j *means each window, and '|' concatenation.*

```
1   Proposed( a,k )
2   {
3       α = {1, 2, 3, 5, 7, 9, ..., 2^k − 1};
4       i = n − 1; p = 0;
5       while( i ≥ 0 ) {
6           w_p = e_i e_{i−1} e_{i−2} ··· e_{i−k+1};
7           i = i − k; sub[p]=FALSE;
8           if( e_i == "1" ) {
9               w_p = w_p + 1;
10              if( w_p ≤ 2^{k−1} )
                    { w_p = 2's complement of w_p;
                      sub[p] = TRUE; }
11              for( ;e_i == "1"; i − − )    w_p = w_p|0;
12          }
13          else {
14              if( w_p < 2^{k−1} )
                    { w_p = 2's complement of w_p;
                      sub[p] = TRUE; }
15              for( ;e_i == "0"; i − − )    w_p = w_p|e_i;
16          }
17          p + +;
18      }

    /*Now, a = w_0 w_1 ··· w_{p−2} w_{p−1}, w_i = e_{i,s_i−1} e_{i,s_i−2}
        ··· e_{i,1} e_{i,0}, 0 ≤ i < p,k ≤ s_i*/

19      while( w_0 > 2^k )    w_0 = w_0 / 2;
20      if( w_0 == even )
21          α = α∪{w_0 − 1, w_0, 2^1 × w_0, 2^2 × w_0,
                                    ··· , 2^{s_0−k} × w_0};
22      else
23          α = α∪{w_0, 2^1 × w_0, 2^2 × w_0, ..., 2^{s_0−k} × w_0};
24      ac = 2^{s_0−k} × w_0
25      for( i = 1; i < p; i + + ) {
26          while( w_i == even )    w_i = w_i / 2;
27          α = α∪ {2^1 × ac, 2^2 × ac, ..., 2^{⌊log w_i⌋+1} × ac};
28          ac = 2^{⌊log w_i⌋+1} × ac
29          if( sub[i]==TRUE )    ac = ac − w_i;
30          else    ac = ac + w_i;
31      α = α∪{ac, 2^1 × ac, 2^2 × ac, ..., 2^{s_i−⌊log w_i⌋−1} × ac};
32          ac = 2^{s_i−⌊log w_i⌋−1} × ac
33      }
34  }
```

3 Performance Analysis

In this section, we calculate the length of the addition/subtraction-chain which can be obtained with the proposed algorithm for both average-case and worst-case, where a, n, and k have the following meanings.

- a: an integer of which addition/subtraction-chain we want to find.
- n: the number of bits required for binary representation of a, that is $\lfloor \log a \rfloor + 1$
- k: the window size

3.1 Average-Case

Elements in the addition/subtraction-chain obtained with the proposed algorithm are partitioned into the following three classes.

1) A set of integers which are prepared in advance: $1, 2, 3, 5, 7, ..., 2^k - 1$.
2) A set of integers obtained by doubling elements in the addition/subtraction-chain found already, and the integer made by adding two elements in the first class.
3) A set of integers obtained by adding an element in the second class and one in the first class, or by subtracting an element in the first class from one in the second class.

Clearly, the number of elements in the first class is $2^{k-1}+1$. The number of elements in the second class is the same as the number of shifts needed during finding the chain. Therefore, we count the number of shifts. If the k-th bit from the leftmost bit is 1, any extra addition is not needed because the value of the leftmost window is an odd integer larger than 2^{k-1} and it is included in the first class already. So, shifts are needed $n-k$ times during finding the addition/subtraction-chain. However, if the bit is 0, extra one addition is needed because the corresponding value of the first k-bit-string is an even integer and it can be obtained by adding one to an element of the first class. So, shifts are needed $n-k+1$ times. As both probabilities that the k-th bit from the leftmost bit is 1 and 0 are $\frac{1}{2}$ respectively, shifts are needed $n-k+0.5$ times on the average. It is the average number of elements in the second class.

We count the average number of windows, because the number of elements in the third class equals to the number of windows minus one. First, we consider the interval between the i-th window and the $(i+1)$-th window. Whether the $(k+1)$-th bit from the leftmost bit of the i-th window is 0 or 1, $(i+1)$-th window can start from the $(k+2)$-th bit. Therefore, the probability that the interval between the i-th window and the $(i+1)$-th window is larger than $k+1$ is 1. It can be generalized. If bits from the $(k+1)$-th to the $(k+j+1)$-th are all 1s or all 0s, the $(i+1)$-th window can start from the $(k+j+2)$-th bit from the leftmost bit of the i-th window, where $j \ge 0$. Therefore, the probability that the interval between the i-th window and the $(i+1)$-th window is larger than $k+j+1$ is 2^{-j}. The expected value of the interval between the i-th window and the $(i+1)$-th window is $k+2(=k+1+\sum_{j=1} 2^{-j})$. This means that each window occupies $(k+2)$ bits on the average. Therefore, the average number of windows in the partition made by the proposed method is $n/(k+2)$, and the number of elements in the third class is $n/(k+2)-1$.

We have counted the number of elements in each class. As those three classes are disjoint, we can calculate the length of addition/subtraction-chain obtained with the proposed algorithm by simply summing those three numbers. It is as follows:

$$2^{k-1} + n - k - 0.5 + \frac{n}{k+2}.$$

3.2 Worst-Case

We consider the worst-case input when we find an addition/subtraction-chain for it. The number of elements in the first among three classes that we classified in Section 3.1 has no relation with the input integer. The number of elements in the second class is determined by only the size of the input integer. When the k-th bit from the leftmost bit is 0, one additional element is required.

The number of elements in the third class is largely influenced by an input integer. The case that the bit string of the input integer is divided into the largest number of windows is the worst-case. As we explained in Section 3.1, the interval between the i-th window and the $(i+1)$-th window is $k+1$ at least. Therefore, if every window occupies $k+1$ bits, there are the largest number of windows. In this case, the average number of windows is $n/(k+1)$.

Now, we can compute the length of the addition/subtraction-chain found with the proposed algorithm in the worst-case. The number of elements in the first class is $2^{k-1}+1$, and in the second class is $n-k+1$. Because the number of elements in the third class equals to the number of windows minus one, it is $n/(k+1)-1$. Therefore, the length of the addition/subtraction-chain is as follows:

$$2^{k-1} + n - k + \frac{n}{k+1}.$$

4 Comparison

We compare the length of the addition/subtraction-chain found by the proposed algorithm with those that can be obtained by existing several methods, where a, n, and k in this section have the same meanings as those used in Section 3.

First, we examine the performance of existing several methods. The average length of the addition-chain obtained with the binary method [8] is $\frac{3}{2}n - 1.5$, and it is not longer than $2n-2$ even in the worst-case. If we use the small-window method [2], we can obtain an addition-chain whose average length is $2^{k-1}+n-k-0.5+\frac{n}{k+1}$, which is not longer than $2^{k-1}+n-k+\frac{n}{k}$ even in the worst-case. Bos and Coster state that they can obtain the addition-chain whose average length is 605 for a 512-bit integer with their heuristic algorithm in [1]. But, it is difficult to analyze the performance of their algorithm systematically because it uses heuristics. The addition/subtraction-chain obtained with the modified binary method [7, 12] guarantees that its length is not longer than $\frac{5}{3}n$, and has the average length of $\frac{4}{3}n$. If we use the algorithm that Yacobi proposed in [15] we can obtain an addition-chain whose average length is $n-(\log n-\log\log n)+1.5(\frac{n}{\log n}+o(\frac{n}{\log n}))$. The average length of the addition/subtraction-chain that can be obtained by Koyama and Tsuruoka's method [9] is $2^{k-1}+n-k+\frac{3}{4}+\frac{n+1/4}{k+3/2}$

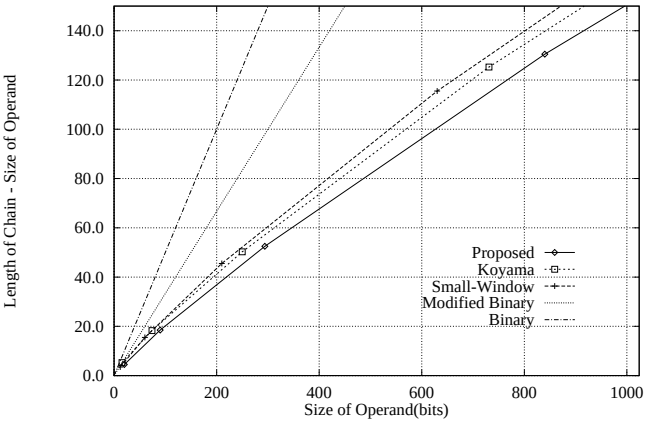


Figure 1: **Comparison of lengths of addition/subtraction-chains**

The length of the addition/subtraction-chain found with the proposed algorithm and the lengths of addition/subtraction-chains obtained with methods examined above are appeared in Figure 4. X-axis indicates the size of an operand, n, and Y-axis indicates the length of an addition/subtraction-chain.

Table 1: Lengths of addition/subtraction-chains, when $n = 512$

Algorithm	Length
binary [8]	766.5(1022)
modified binary [7, 12]	681.7(768)
Yacobi's [15] [9]	635.1(−)
small-window(5) [8]	607.8(625.4)
large-window(11) [1]	605(−)
Koyama's(5) [9]	602.6(629)
proposed (5)	**595.6(608.3)**

The concrete values for a 512-bit integer corresponding to Figure 4 are appeared in Table 1. The number between the parentheses on the algorithm column in Table 1 means the size of windows used, and that on the length column means the worst-case length. As we can see in Figure 4 and Table 1, the proposed algorithm finds the shorter addition/subtraction-chain than what can be found by any other existing method.

5 Conclusion

In this paper, we proposed an addition/subtraction-chain algorithm. The proposed algorithm is based on the clas-

sical small-window method, and uses a new window splitting mechanism.

Let a be an integer of which addition/subtraction-chain we want to find, and k be the window size. The average length of the addition/subtraction-chain which can be found with the proposed algorithm is as follows:

$$2^{k-1} + \lfloor \log a \rfloor - k + 0.5 + \frac{\lfloor \log a \rfloor + 1}{k+2}.$$

In the worst case, we guarantee the following length of the addition/subtraction-chain with our algorithm.

$$2^{k-1} + \lfloor \log a \rfloor - k + 1 + \frac{\lfloor \log a \rfloor + 1}{k+1}$$

These are shorter average-case and worst-case lengths than those of any other existing addition /subtraction-chain algorithm.

Acknowledgements

This work was supported by the KOSEF (Korea Science and Engineering Foundation) through the AITrc (Advanced Information Technology Research Center) and the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment)

References

[1] Jurjen Bos and Matthijs Coster, "Addition chain heuristics," in *Crypto'89*, pp. 400–407, 1989.

[2] M. J. Coster, *Some algorithms on addition chains and their complexity.* CWI Report CS-R9024, 1990.

[3] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Computers*, vol. IT-22, pp. 644–654, June 1976.

[4] Whitfield Diffie, "The first ten years of public-key cryptography," in *Proceeding of The IEEE*, vol. 76,NO.5, pp. 560–576, May 1988.

[5] Peter Downey, Benton Leong, and Rave Sethi, "Computing sequences with addition chains," *SIAM J. Comput.*, vol. 10, pp. 638–646, August 1981.

[6] T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. IT-31, no. 4, pp. 469–472, 1985.

[7] J. Jedwab and C. J. Mitchell, "Minimum weight modified signed-digit representations and fast exponentiation," *Electronics Letters*, vol. 25, pp. 1171–1172, 1989.

[8] D. E. Knuth, *The art of computer programming.* Addison-Wesley,Inc., 1981.

[9] Kenji Koyama and Yukio Tsuruoka, "A signed binary window method for fast computing over elliptic curves," *IEICE Transactions on Fundamentals*, vol. E76-A, pp. 55–62, 1993.

[10] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key crytosystems," *Communications of ACM*, vol. 21, pp. 120–126, 1978.

[11] A. Schonhage, "A lower bound on the length of addition chains," *Theoretical Computer Science*, vol. 1, pp. 1–12, 1975.

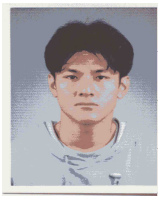
[12] A. Selby and C. Mitchell, "Algorithms for software implementations of RSA," *IEE Proceedings - Computer & Digital Technology*, vol. 136, pp. 166–170, MAY 1989.

[13] Y. Tsai and Y. Chin, "A study of some addition chain problems," *International Journal of Computer Mathematics*, vol. 22, pp. 117–134, 1987.

[14] Hugo Volger, "Some results on addition/subtraction chains," *Information Processing Letters*, vol. 20, pp. 155–160, 1985.

[15] Y. Yacobi, "Exponentiating faster with addition chains," in *Eurocrypt'90*, pp. 222–229, 1991.

Younho Lee received the B.E. degree in computer science from Korea Advance Institute of Science and Technology (KAIST), South Korea, in 2000, the M.S. degree in computer science from KAIST, in 2002. He is currently working toward the Ph.D. degree at the Division of Computer Science, KAIST.



Heeyoul Kim received the B.E. degree in computer science from Korea Advance Institute of Science and Technology (KAIST), South Korea, in 2000, the M.S. degree in computer science from KAIST, in 2002. He is currently working toward the Ph.D. degree at the Division of Computer Science, KAIST.

Seong-Min Hong received the B.E. degree in computer science from KAIST, South Korea, in 1994. He also received the M.S. degree and Ph.D degree in computer engineering from KAIST in 1996 and 2000, respectively. He is currently an research professor in the division of Computer Science at KAIST.

Hyunsoo Yoon received the B.E. degree in electronics engineering from SNU, South Korea, in 1979, the M.S. degree in computer science from KAIST, in 1981, and the Ph.D. degree in computer and information science from the Ohio State University, Columbus, Ohio, in 1988. From 1988 to 1989, with the AT & T Bell Labs. as a Member of Technical Staff. Since 1989 he has been a faculty member of Division of Computer Science at KAIST.