PERGAMON Computers and Mathematics with Applications 38 (1999) 187-196

An International Journal
computers & mathematics with applications

www.elsevier.nl/locate/camwa

A Method for Computing Lucas Sequences

CHING-TE WANG AND CHIN-CHEN CHANG
Institute of Computer Science and Information Engineering
National Chung Cheng University, Chiayi, Taiwan 62107, R.O.C.
<wct><ccc>@cs.ccu.edu.tw

CHU-HSING LIN*
Department of Computer and Information Sciences
Tunghai University, Taichung, Taiwan 407, R.O.C.
chlin@mail.thu.edu.tw

(Received January 1999; accepted February 1999)

Abstract—Most of public-key cryptosystems rely on one-way functions, which can be used to encrypt and sign messages. Their encryption and signature operations are based on the computation of exponentiation. Recently, some public-key cryptosystems are proposed and based on Lucas functions, and the Lucas sequences are performed as $S = V(d) \bmod N$. In this paper, we will transform the concept of addition chains for computing the exponentiation evaluations to the Lucas chains for computing the Lucas sequences. Theoretically, the shorter Lucas chain for d is generated, the less computation time for evaluating the value $V(d)$ is required. Therefore, we proposed a heuristic algorithm for evaluating a shorter Lucas chain and then use it to compute the Lucas sequence with less modular multiplications. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords—Addition chain, Cryptosystem, Signature scheme, Lucas chain.

1. INTRODUCTION

In 1976, Diffie and Hellman proposed the pioneering concept of public-key cryptosystems [1] and the key exchange scheme based on discrete logarithms over a Galois field $GF(P)$. After the idea was introduced, Rivest *et al.* proposed the most promising public-key cryptosystem [2]. The RSA scheme is based on the difficulty of factorization problem. These methods are required to compute the modular exponentiations.

Recently, Smith and Lennon proposed a new public-key system, the Lucas cryptosystem [3,4], which uses a new one-way function based on Lucas functions. The Lucas system applies a special type of second-order linear recurrences to cryptosystems and signature schemes. For example, the ElGamal's public-key cryptosystem and its digital signature [5] can be implemented by the Lucas function [6,7]. Further, Diffie-Hellman key distribution [1] can also be constructed based on the Lucas functions. Smith and Lennon pointed out that the reason of using Lucas functions instead of exponentiations is its cryptographical strength. It is much stronger than or at least as strong as the exponentiation-based systems. In 1995, Bleichenbacker *et al.* published some remarks on Lucas-based cryptosystems [8] and pointed out that some Lucas systems are vulnerable to subexponential time attacks. Of course, it is likely to spark a substantial debate concerning the

*Author to whom all correspondence should be addressed.

security of their schemes. At the same year, Laih *et al.* analyzed the security of Lucas function [9], and showed that its security is polynomial-time equivalent to the generalized discrete logarithm problems.

On the other hand, several researches on fast exponentiation evaluation have been proposed. Knuth presented a simple square-and-multiply method [10] based on the binary representation of the exponent. Yacobi uses addition chains [11] to reduce the number of multiplications. However, to find the shortest addition chain has been shown to be an NP-complete problem [12]. Similarly, some researches are concerning the fast computation of Lucas functions. To efficiently evaluate the Lucas values, it is important to find the shortest Lucas chain, which is the same as addition chains and is also a hard problem. In 1995, Yen and Laih proposed two algorithms by scanning the binary form of the exponent and sequentially evaluate the Lucas sequence [13]. Now, we will propose a heuristic method for finding a Lucas chain and then compute the Lucas sequence. The computation of modular multiplication is reduced and the Lucas values can be evaluated efficiently.

The proposed paper is organized as follows. In Section 2, we review the Lucas functions and the relative computations. In Section 3, we develop a method to generate Lucas chains and use it to compute the Lucas sequences. The discussions and applications will appear in Section 4. Finally, we make some conclusions in the last section.

2. REVIEW OF LUCAS FUNCTIONS

2.1. General Lucas Functions

Lucas functions can be seen as generalized linear recurrences. Assume that $P_1, P_2, \dots, P_m$ are integers, a Lucas function is a sequence of integers $\{V(n)\}$ defined as follows:

$$V(n) = P_1 V(n-1) + P_2 V(n-2) + \cdots + P_m V(n-m),$$

where $V(0), V(1), V(2), \dots, V(m-1)$ are given independently before using the equation. This equation is referred as an m^{th} order linear recurrence. Obviously, to find a value $V(n)$, we have to evaluate the previous $m-1$ values of $V(i)$'s and that needs enormous computations. The sequence defined by the first-order linear recurrence $V(n) = PV(n-1)$ is the multiplication of P with the power n and the initial $V(0)$. Under this case, the Lucas function is reduced to an exponentiation operation.

The general second-order linear recurrences are usually used in the recent cryptosystems [3,4]. Here, two functions $U(n), V(n)$ in Lucas sequences are defined as follows:

$$U(0) = 0, \quad U(1) = 1, \quad U(n) = P_1 \times U(n-1) - P_2 \times U(n-2), \quad \text{for } n \geq 2, \tag{1}$$

$$V(0) = 2, \quad V(1) = P_1, \quad V(n) = P_1 \times V(n-1) - P_2 \times V(n-2), \quad \text{for } n \geq 2, \tag{2}$$

where P_1, P_2 are two relatively prime numbers. In equation (1), if the parameters are selected as $P_1 = 1, P_2 = -1$, the sequence derived by the recursive is the well-known Fibonacci sequence. In equation (2), the recursive function $V(n)$ with $P_2 = 1$ is usually used to devise cryptosystems by cryptographers.

2.2. Computations of Lucas Functions

For the RSA cryptosystem, several methods, such as "addition chain" [11,14] or "square-and-multiply" [10,15] algorithms, are given to speed up the computations for exponentiations [16–18]. For the Lucas system, it is infeasible to evaluate $V(n) \bmod N$ in a recursive manner. In [3], a method is proposed by directly combining formulas of the Lucas function. Recently, Yen and

Laih [13] proposed two algorithms to evaluate the Lucas function $V(n) = MV(n-1) - V(n-2)$, which is based on the following property of the function:

$$V(x+y) = V(x)V(y) - V(x-y), \qquad \text{for } x \geq y \geq 0. \tag{3}$$

This property describes the relationship between two numbers x and y in a Lucas sequence. In their schemes, a value d has to be expressed in its binary form as:

$$d = (d_{t-1}, d_{t-2}, \dots, d_1, d_0)_2, \qquad \text{where } d_i \in \{0,1\}, \quad i = 0, 1, \dots, t-1.$$

The first algorithm is an approach of left-to-right scanning. It scans the binary form step by step and evaluates the partial value of $V(d)$ from left to right by the following formula:

$$\begin{aligned} T_{i-1} &= (\dots(d_{t-1} \times 2 + d_{t-2}) \times 2 \cdots + d_i) \times 2 + d_{i-1} \\ &= T_i \times 2 + d_{i-1} = T_i + d_{i-1} + T_i, \end{aligned}$$

and

$$V(T_{i-1}) = V(T_i + d_{i-1})V(T_i) - V(d_{i-1}).$$

On the contrary, the second algorithm is a right-to-left scanning approach. Similar to the previous algorithm, the partial value of $V(d)$ can be obtained as follows:

$$\begin{aligned} Y_i &= \sum_{k=0}^{i} d_k \cdot 2^k = d_i \cdot 2^i + \sum_{k=0}^{i-1} d_k \cdot 2^k \\ &= d_i \cdot 2^i + Y_{i-1}, \end{aligned}$$

and

$$V(Y_i) = \begin{cases} V(Y_{i-1}), & \text{if } d_i = 0, \\ V(2^i)\, V(Y_{i-1}) - V(2^i - Y_{i-1}), & \text{if } d_i = 1. \end{cases}$$

Briefly, the methods scan the binary form bit by bit from left to right or right to left and use the above formulas to derive the partial values. Obviously, it requires two modular multiplications for each bit, no matter what the value of the bit is. In the following, we will develop a method to find the Lucas chains instead of scanning the bits and thus, the computation of modular multiplications can be reduced dramatically.

3. THE PROPOSED ALGORITHMS BASED ON LUCAS CHAINS

In this section, we will develop an algorithm to find a Lucas chain and then use it to evaluate the Lucas sequences.

3.1. Lucas Chains and Lucas Sequences

As we know, the exponentiation problem can be conducted by additions, because of the exponents are additive. Addition chains can theoretically be used to reduce the number of multiplications for the exponentiation [14–18]. Similarly, the computation of a Lucas sequence can also be reduced by a shorter chain, which is called a Lucas chain. In the following, we are devoted to find a shorter Lucas chain and evaluate the Lucas sequence efficiently. By using equation (2), with $P_1 = M$ and $P_2 = 1$, the Lucas system is defined as follows:

$$V(n) = MV(n-1) - V(n-2), \qquad \text{for } n \geq 2, \text{ and } V(0) = 2, V(1) = M, \tag{4}$$

where M is the plaintext. First, the Lucas chain [13] is defined as follows.

DEFINITION 1. *Given an integer n, a Lucas chain for n is a sequence of increasing integers $(a_0, a_1, \dots, a_r)$ such that*

(i) $a_0 = 0$, $a_1 = 1$, *and* $a_r = n$;
(ii) $a_i = a_j + a_k$, *for some* $k \leq j < i$;
(iii) $a_j - a_k \in \{a_0, a_1, \dots, a_r\}$.

From the definition, we can see that a Lucas chain has one more requirement than an addition chain. A sequence satisfies the three conditions will surely be an addition chain. That is, a Lucas chain is definitely an addition chain, whereas the converse may not be true.

DEFINITION 2. *Given a sequence $(a_0, a_1, \dots, a_r)$, the length of the Lucas chain is defined as r.*

EXAMPLE 1. The sequences (0,1,2,3,5,10,15) and (0,1,2,3,4,7,8,15) are both addition chain and Lucas chain for 15. The lengths of the two sequences are 6 and 7, respectively. However, the sequence (0,1,2,3,6,12,15) is an addition chain but not a Lucas chain, since the term $12 + 3 = 15$ but $9 = 12 - 3$ does not belong to the sequence.

Now, we will concentrate on the evaluation of the nth term of a Lucas sequence $V(n)$, which is corresponding to the Lucas chain for n. The construction of a Lucas chain for an integer n can be transformed into a specific way of the computation of $V(n)$. It has been shown that finding a shortest addition chain for a random integer is an NP-complete problem [12]. Intuitively, it is believed that this fact is also true for finding the Lucas chains. Therefore, a heuristic method is used to find a Lucas chain for an integer n. If the shortest Lucas chain for an integer n can be found, the nth term of the Lucas sequence $V(n)$ can also be evaluated with the minimum number of multiplications.

In [4,13], the authors had shown the property of equation (3), which expresses the relationship between terms in a Lucas sequence. The following Lemma is implied straightforwardly from the property.

LEMMA 1. *Let x be an integer,*

(i) $V(2x) = V(x)V(x) - V(0)$;
(ii) $V(3x) = V(2x)V(x) - V(x)$;
(iii) $V(5x) = V(3x)V(2x) - V(x)$.

From (i), *we can find out the relationship between the x^{th} term and the $2x^{th}$ term of a Lucas sequence. In other words, the value of the $2x^{th}$ term can be computed from the x^{th} term. From* (ii), *if an integer y is a multiplier of 3, i.e., $y = 3x$, then the value of the y^{th} term can be obtained from the x^{th} and the $2x^{th}$ terms. Similarly,* (iii) *can also be applied. In Lemma 1, these terms are formed a partial chain, which is called a Lucas subchain. The Lucas subchain for an integer d can be computed by decomposing the value d.*

LEMMA 2. *Suppose an integer d is the d^{th} term of Lucas sequence, if d is decomposed as Lemma 1, then the Lucas subchain for d can be evaluated.*

PROOF.

(i) If d is even, i.e., $d = 2x$, for some $x \in I$, and d is decomposed as (i) in Lemma 1. Let $a_j = x$, $a_i = d = 2a_j$, then (a_j, a_i) is formed a Lucas subchain by Definition 1.
(ii) If d is a multiplier of 3, i.e., $d = 3x$ for some $x \in I$, and d is decomposed as (ii) in Lemma 1. Let $a_k = x$, $a_j = 2x$, and $a_i = d$, then (a_k, a_j, a_i) is also formed a Lucas subchain by Definition 1.
(iii) Similarly, the chain $(x, 2x, 3x, 5x)$ is also formed a Lucas subchain. ∎

For explanation, the decomposition of an integer d can be represented by a tree. The integer d is regarded as a root node, which is then divided into several children according to Lemma 1. The decomposed trees are depicted in Figure 1 for the Cases (i)–(iii), respectively, in Lemma 1. Note that $V(0)$ is a given value and omitted in the tree.

On the other hand, a Lucas chain is also corresponding to a directed graph. As Equation (ii) in Definition 1, a_i, a_j, and a_k denote the nodes. And one arc directing from a_j to a_i and one from a_k to a_i are used to represent the equation $a_i = a_j + a_k$. For example, the Subchains (i) and (iii) of Lemma 1, can be depicted in (a) and (b) of Figure 2, respectively.

However, one problem may be raised. When an integer d is a multiplier of 7, i.e., $d = 7x$, for some x, does the Lucas sequence be $V(7x) = V(4x)V(3x) - V(x)$? That is, can it be decomposed

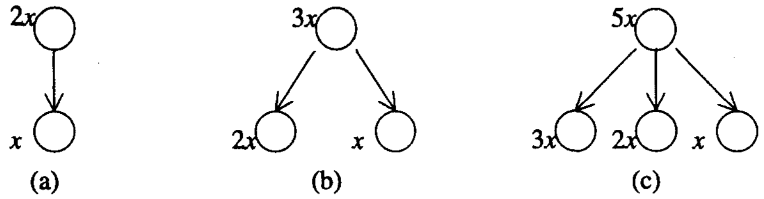

Figure 1. The decomposed trees.

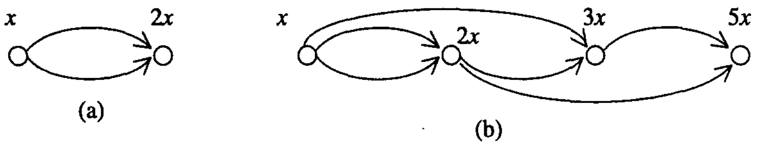

Figure 2. A directed graph representation of Lucas subchains.

into a Lucas subchain for d? The answer is negative. The chain $(x, 3x, 4x, 7x)$ is not a Lucas subchain since $4x = 3x + x$, but $2x = 3x - x$ does not belong to the chain. This violates Condition (iii) in Definition 1. Therefore, we need to consider those numbers, which are not described in Lemma 1.

LEMMA 3. *If x is an odd integer, $V(x) = V(\lfloor x/2 \rfloor + 1)V(\lfloor x/2 \rfloor) - V(1)$.*

Similarly, the above lemma can easily be proved by equation (3). According to the previous division, the decomposed tree can be obtained and indicated in Figure 3. The chain $(\lfloor x/2 \rfloor, \lfloor x/2 \rfloor + 1, x)$ is not a Lucas subchain, since $\lfloor x/2 \rfloor - 1$ does not belong to the chain. Therefore, the terms $\lfloor x/2 \rfloor$ and $\lfloor x/2 \rfloor + 1$ are required to be decomposed further.

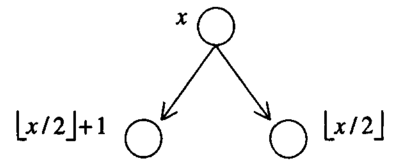

Figure 3. The decomposed tree.

When a decomposed tree is obtained by Lemmas 1 and 3, the nodes can be classified into two types, the dependent nodes and independent nodes.

DEFINITION 3. *Let a decomposed tree be obtained from Lemma 1 or 3. A node is called a dependent node if it can be expressed by previous two terms and that formed a Lucas subchain; otherwise, it is called an independent node.*

For example, the node x in Figure 1 is an independent node, while the nodes $2x$, $3x$, and $5x$ are dependent nodes. And the nodes $\lfloor x/2 \rfloor$ and $\lfloor x/2 \rfloor + 1$ in Figure 3 are independent nodes.

From Lemmas 1 and 3, any number can be decomposed by a corresponding division and indicated as a tree. If its child is a dependent node, its Lucas value of that node can be evaluated by previous nodes. If the child is an independent node, which is required to be decomposed further to a dependent node.

LEMMA 4. *Let y be an odd integer and $x = \lfloor y/2 \rfloor$. The chain $(x, x+1, y)$ is formed a Lucas subchain if two terms $\lfloor x/2 \rfloor$ and $\lfloor x/2 \rfloor + 1$ are included in the chain.*

PROOF. Assume that 0, 1 are two given node values and the two terms $\lfloor x/2 \rfloor$ and $\lfloor x/2 \rfloor + 1$ are included in the chain. Now, we are going to show that $(x, x+1, y)$ is a Lucas subchain. That is, we will prove that x, $x+1$, and y satisfy Definition 1.

(1) Since y is odd, we have $y = \lfloor y/2 \rfloor + (\lfloor y/2 \rfloor + 1) = x + (x+1)$. And $(x+1) - x = 1$, which is the given node value. So, the term y satisfies Definition 1.

(2) Let us consider the terms x and $x+1$. If x is even, then $x = \lfloor x/2 \rfloor + \lfloor x/2 \rfloor, x+1 = \lfloor x/2 \rfloor + (\lfloor x/2 \rfloor + 1)$. So, x and $x+1$ satisfy Definition 1. If x is odd, then $x = \lfloor x/2 \rfloor + (\lfloor x/2 \rfloor + 1)$, $x+1 = (\lfloor x/2 \rfloor + 1) + (\lfloor x/2 \rfloor + 1)$. So, x and $x+1$ also satisfy Definition 1. Therefore, the chain $(x, x+1, y)$ is a Lucas subchain. ∎

THEOREM 5. *Given an integer d, the decomposed tree and Lucas chain for d can be constructed.*

PROOF. From Lemmas 1 and 3, the integer d can be decomposed into a subtree. Its children are composed of dependent nodes and independent nodes. The dependent nodes form a Lucas subchain, which can be proved by Lemmas 2 and 4. Further, the dependent nodes are stored in an array in decreasing manner. The remaining independent nodes are decomposed again by Lemmas 1 and 3. This process will be repeated until the given node $d = 1$ is found. Consequently, the decomposed tree and Lucas chain can be constructed. ∎

THEOREM 6. *Given an integer d and its Lucas chain, the Lucas sequence $V(d)$ can be evaluated.*

PROOF. The Lucas chain for an integer d can be derived by Theorem 5. The chain's node values are stored in an array by their decreasing order. Therefore, we can evaluate the Lucas sequence by the formulas in Lemmas 1 and 3 in a reverse order until the Lucas sequence $V(d)$ of node d is computed.

3.2. Computation of Lucas Chains

Given an integer d, we decompose d by the method of Lemmas 1 and 3. It can be considered as four filters, which inspect the value d and properly generate a Lucas subtree. Each node in the subtree is required a modular multiplication in the computation of a Lucas sequence. Therefore, the main problem is how to construct a Lucas tree such that it has fewer nodes. Moreover, if an integer d can be decomposed simultaneously by the three filters in Lemma 1, how do we select one from them? In general, we would suggest that (i) is the best selection and (iii) is the final consideration. Since the comparison of the number of children in Figure 1 is indicated by the inequality $\log_2 d < 2\log_3 d < 3\log_5 d$. On the other hand, if the integer d is odd, (ii) and (iii) in Lemmas 1 and 3 can be used. The priority of choosing Lemma 1 is higher than Lemma 3, since $2\log_3 d < 3\log_5 d < 2\log_2 d$. The Lucas chain for d can be derived by the following algorithm.

```
ALGORITHM A. [Find a Lucas chain.]
Input: $d$.
Output Lucas chain: $A[I], A[I-1], \dots, A[0]$.
Step 1: [Assign the initial $d$ in the array.]
        $I = 0$, $A[I] = d$.
Step 2: [End of decomposition, output the Lucas chain.]
        IF $d = 1$ THEN output $A[I], A[I-1], \dots, A[0]$; Stop.
Step 3: [Decompose the integer $d$ by filters.]
        IF $d$ is even THEN
            $d = d/2$; $I = I + 1$; $A[I] = d$; GOTO Step 2.
        IF $d$ is divisible by 3 THEN
            $d = d/3$; $I = I + 1$; $A[I] = 2 * d$; $I = I + 1$; $A[I] = d$; GOTO Step2.
        IF $d$ is divisible by 5 THEN
            $d = d/5$; $I = I + 1$; $A[I] = 3 * d$; $I = I + 1$; $A[I] = 2 * d$; $I = I + 1$; $A[I] = d$; GOTO
            Step2.
Step 4: [Other number $d$]
            $dh = \lfloor d/2 \rfloor$; $dl = dh + 1$; $I = I + 1$; $A[I] = dl$; $I = I + 1$; $A[I] = dh$;
        REPEAT
            $dh = \lfloor dh/2 \rfloor$; $dl = dh + 1$; $I = I + 1$; $A[I] = dl$; $I = I + 1$; $A[I] = dh$;
        UNTIL $dh = 2$;
            $d = 1$; $I = I + 1$; $A[I] = d$; GOTO Step 2.
```

In the above algorithm, we use an array to store the chains. First, the number d is stored in $A[0]$. After the independent nodes are decomposed, the dependent nodes are sequentially stored in the array. In Step 3, the filters are used to decompose the independent nodes. However, not all numbers can be filtered in Step 3, the remaining numbers are filtered by Step 4. Now, let us consider an example. If $d = 108$, the decomposed tree can be depicted as in Figure 4.

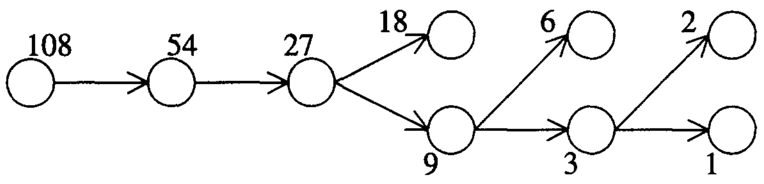

Figure 4. The decomposed tree.

After executing Algorithm A, the Lucas chain can be obtained as in Figure 5.

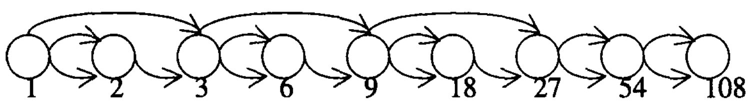

Figure 5. Lucas chain.

3.3. Computation of Lucas Sequences

For simplicity, we assume that the Lucas system is defined as equation (4) in Section 3.1. First, we compute the Lucas chain for the value d (or e) as described in the previous section. To find the Lucas sequence, we can use the given values $d, V(0), V(1)$ and the Lucas chain as the inputs, and repeatedly compute the dependent nodes. The algorithm is described as follows.

```
ALGORITHM B. [Find Lucas sequences]
Input: LUC chain $A[I], A[I-1], \ldots, A[0], d$.
Output: Lucas sequence $V(d)$.
Step 1: [Assign initial values.]
        $V(0) = 2$; $V(1) = M$.
Step 2: [Compute the Lucas sequence.]
        FOR $k = I - 1$ downto 0
        IF $A[k] = 2A[k+1]$ THEN
          $V(A[k]) = V(A[k+1]) * V(A[k+1]) - V(0) \bmod N$;
        ELSE IF $A[k] = A[k+1] + A[k+2]$ THEN
              $V(A[k]) = V(A[k+1]) * V(A[k+2]) - V(A[k+2]) \bmod N$;
            ELSE IF $A[k] = 2A[k+2]$ THEN
                  $V(A[k]) = V(A[k+2]) * V(A[k+2]) - V(0) \bmod N$;
                ELSE $(A[k] = A[k+2] + A[k+3])$
                  $V(A[k]) = V(A[k+2]) * V(A[k+3]) - V(1) \bmod N$;
                ENDIF
            ENDIF
        ENDIF
        NEXT $k$;
        $V(A[0])$ is the output.
```

Note that $V(0) = 2$, $V(1) = M$ are given. In Step 2, the Lucas chains are stored in an array and each Lucas value can be evaluated by the previous dependent nodes.

4. DISCUSSIONS AND APPLICATIONS

4.1. Performance Analyses

In Yen and Laih's methods [13], the integer d has to be represented first in the binary form. Both of their algorithms are required to sequentially scan each bit. Therefore, in order to compute $V(d)$, it requires two modular multiplications for scanning each bit. Totally, $2(1 + \lfloor \log_2 d \rfloor)$ modular multiplications are needed.

In the proposed method, the operations are composed of two parts. In the first part for constructing Lucas chain, it checks if the number d is suitable for filters in Lemma 1 or 3. It requires division operations. In the second part for constructing Lucas sequence, the algorithm evaluates the Lucas value, which is dependent on the previous values. It requires modular multiplications to combine the Lucas values. Obviously, the number of modular multiplications is greater than that of division operations. For the purpose of analyses, we assume that the number d can be expressed as $d = 2^k$ ($d = 3^k$ or $d = 5^k$), and can be applied to Lemma 1. In this case, the required number of modular multiplications in the second part is $\lfloor \log_2 d \rfloor$ ($2\lfloor \log_3 d \rfloor$ or $3\lfloor \log_5 d \rfloor$). The number is corresponding to that of children in the decomposed tree, since each child's node requires one modular multiplication. In [13], their method requires $2(1 + \lfloor \log_2 d \rfloor)$ modular multiplications. By neglecting the constant, the value $2(\lfloor \log_2 d \rfloor)$ will be compared with ours. We can see that the numbers of modular multiplications are approximately reduced to 50%, 63%, and 65%, respectively. Additionally, it requires $\lfloor \log_2 d \rfloor$, $\lfloor \log_3 d \rfloor$, and $\lfloor \log_5 d \rfloor$ division operations, respectively, in the first part of our method. This fact can be regarded as that the half number of the modular multiplications in [13] are transferred to divisions in our method. Thus, the number of modular multiplications is reduced. On the other hand, if the random number d cannot be filtered in Lemma 1, we need to apply Lemma 3. Then, the required number of modular multiplications is $2(\lfloor \log_2 d \rfloor)$, and we can only save two modular multiplications. Now, let us analyze the efficiency in average case. We assume that a decomposed tree with d nodes has levels of $l = \lfloor \log_2 d \rfloor$. If there are $l/2$ levels can be manipulated by Lemma 1 and the others by Lemma 3 on average, the average number of modular multiplications is approximately $(1/2)(2\lfloor \log_3 d \rfloor)$ in Lemma 1 and $(1/2)(2\lfloor \log_2 d \rfloor)$ in Lemma 3. Therefore, the average number of modular multiplications is $\lfloor \log_3 d \rfloor + \lfloor \log_2 d \rfloor$, which is approximately 19% reduced comparing to [13].

4.2. Discussions

(1) From the performance analyses, we can reduce the number of modular multiplications, which can be replaced by the division operations. Therefore, the computation time can be reduced in the proposed method.

(2) There are three filters in Lemma 1. Similarly, there are some other filters can be devised by proper modifications. For example, $V(7x) = V(4x)V(3x) - V(x)$, the chain $(x, 3x, 4x, 7x)$ is not a Lucas chain, but $V(7x) = V(5x)V(2x) - V(3x)$, the chain $(2x, 3x, 5x, 7x)$ is formed a Lucas chain, if x is included to the Lucas chain. Therefore, it is important to construct other filters by using equation (3).

(3) From the definition of Lucas chain, we know that a Lucas chain is definitely an addition chain, while the converse may not be true. Therefore, the created Lucas chain in the proposed method is also an addition chain, it can also be applied to the computation of modular exponentiations. Furthermore, we can modify the decomposed method and obtain fewer nodes, since the addition chain does not satisfy the Condition (iii) as in the definition of Lucas chain.

(4) The relationship between nodes in the decomposed tree and the nodes in the Lucas chain are one-to-one correspondence. The shorter Lucas chain will result in less number of modular multiplications. In our heuristic algorithm, the length of the chain is constructed

by logarithmic operations. However, how can we construct a shorter Lucas chain is an interesting research topic.

4.3. Applications

Most of public key cryptosystems rest on a trapdoor function, which can be employed to encrypt and sign messages. The well-known and widely used schemes, such as RSA [2] and ElGamal's method [5], are based on computing the exponentiation with modulus a large number. Recently, some researchers developed cryptographic scheme based on Lucas functions [6,7]. Assume that the Lucas system is defined as equation (4). Similar to RSA, in the Lucas system, we have to select two large prime numbers p and q, and compute $N = p \times q$. The encryption key e can be randomly selected and the decryption key d is computed such that $d \times e = 1 \text{mod} R(N)$, where $R(N) = lcm((p - (D/p)), (q - (D/q)))$, $D = M^2 - 4$, and $(D/p), (D/q)$ are the Legendre symbols of D corresponding to p and q, respectively. The detailed discussions are given in [3,4]. Afterward, the encryption procedure of the Lucas scheme is given as

$$C = V(e) \text{mod} N,$$

and the signature procedure is

$$S = V(d) \text{mod} N,$$

where C is the ciphertext of the message M, and S is the signature. To speed up the computation of $V(e)\text{mod}N$ (or $V(d)\text{mod}N$), the proposed scheme in Section 3 can be applied for the encryption and signature procedure.

5. CONCLUSIONS

The algorithms for constructing Lucas chains and Lucas sequences have been proposed. Based on the Lucas property, a decomposed tree for an integer d can be constructed, and the corresponding Lucas subchain can also be generated. The independent nodes in the subtree are then repeatedly decomposed and formed a skew tree. Since the decomposed method for d uses the logarithmic operation, the generated Lucas chain will not be long. From the feature of Lucas function, the desired Lucas value is combined and evaluated. From the performance analyses, we can see that the required number of modular multiplications is reduced.

REFERENCES

1. W. Diffie and M. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory* **IT-22** (6), 644–654 (1976).
2. R.L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystem, *Communications of the ACM* **21**, 120–126 (February 1978).
3. P. Smith and M. Lennon, LUC: A new public key system, In *Proceedings of the 9th IFIP Int. Symp. on Computer Security*, pp. 103-107, (1993).
4. P. Smith and C. Skinner, A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms, In *Pre-proceedings Asiacrypt'94*, pp. 298–306.
5. T. ElGamal, A public key cryptosystem and signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory* **IT-31** (4), 469–472 (1985).
6. F.E.A. Lucas, Théorie des fonctions numériques simplement périodiques, *Amer. J. Math.* 1, 184–240 and 289–321 (1878).
7. D.H. Lehmer, An extended theory of Lucas's functions, *Ann. Math.* **31**, 419–448 (1930).
8. D. Bleichenbacher, W. Bosma and A.K. Lenstra, Some remarks on Lucas-based cryptosystems, In *Advances in Cryptology—Proceedings of Eurocrypt'95, Lecture Notes in Computer Science*, pp. 386–396, Springer-Verlag.
9. C.S. Laih, F.K. Tu and W.C. Tai, On the security of Lucas function, *Information Processing Letters* **53**, 243–247 (1995).
10. D.E. Knuth, *The Art of Computer Programming, Volume II: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, (1969).

11. Y. Yacobi, Exponentiating faster with addition chains, In *Advances in Cryptology—Proceedings of Eurocrypt'90, Lecture Notes in Computer Science*, pp. 222–229, Springer-Verlag.
12. P. Downey, B. Leony and R. Sethi, Computing sequences with addition chains, *SIAM Journal on Computing* **10** (3), 638–646 (1981).
13. S.M. Yen, and C.S. Laih, Fast algorithms for LUC digital signature computation, *IEE Proc.—Comput. Tech.* **142** (2), 165–169 (March 1995).
14. J. Bos and M. Coster, Addition chain heuristics, In *Advances in Cryptology—Proceedings of Crypto'89, Lecture Notes in Computer Science*, pp. 400–407, Springer-Verlag.
15. S.M. Yen and C.S. Laih, The fast cascade exponentiation algorithm and its applications on cryptography, In *Advances in Cryptology—Proceedings of Auscrypt'92, Lecture Notes in Computer Science*, pp. 447–456, Springer-Verlag.
16. C.C. Chang, W.J. Horng and D.J. Buehrer, A cascade exponentiation evaluation scheme based on the Lempel-Ziv-Wetch compression algorithm, *Journal of Information Science and Engineering* **11**, 417–431 (1995).
17. D.C. Lou and C.C. Chang, Fast exponentiation method obtained by folding the exponent in half, *Electronics Letters* **32** (11), 984–985 (1996).
18. C.C. Chang and D.C. Lou, Parallel computation of the multi-exponentiation for cryptosystems, *International Journal of Computer Mathematics* **63** (1/2), 9–26 (1997).