Journal of Computer Science 6 (10): 1203-1207, 2010
ISSN 1549-3636

Reduce Computation Steps Can Increase the Efficiency of Computation Algorithm

Zulkarnain Md. Ali
Center of Computer Science, Faculty of Information Science and Technology,
University Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia

Abstract: Problem statement: Lucas function is a special form of second-order linear recurrence relation. It is used in LUC cryptosystems. The performance of LUC cryptosystem depends on the size of public key, messages and two relatively primes. The increasing of size of these parameters will increase the computation time need to perform the LUC Cryptosystem computation. The efficiency means the quality to avoid wasted time. **Approach:** Therefore, the main theme of this study was to design and implement an improve version of computation algorithm. The efficiency of computation can skip some computations time for a computerized calculation. Smaller computation time means the algorithm is better and more efficient than the other algorithm. In this study, the technique on reducing redundant number of computations steps in LUC Cryptosystem was investigated. The use of two variables w and t were proposed in order to reduce some computations steps in LUC Cryptosystem computation. **Results:** The new technique showed a better computation time compared to the existing algorithm. It also reduced some redundant multiplications without sacrificed the security of LUC Cryptosystem. At the same time, it increases the efficiency of computation algorithm. **Conclusion:** The proposed algorithm showed better speed and efficiency by reducing some redundant computation steps. It can reduce up to 20% of computation efforts compare to the existing one.

Key words: Efficiency of computation algorithm, computation time, LUC cryptosystem

INTRODUCTION

The analysis of algorithms is the area of computer science that provides tools for contrasting the efficiency of different methods of solution. Notice the use of the term methods of solution rather than programs; it is important to emphasize that the analysis concerns itself primarily with significant differences in efficiency where these differences that can be usually obtain only through superior methods of solution and rarely through clever tricks in coding.

Although the efficient use of both time and space is important, inexpensive memory has reduced the significance of space efficiency. Thus, in this study the focus is primarily on time efficiency. Based on this focus, the possible approach is to implement the existing and proposed algorithms in C and run the programs.

In general, the total amount of computation for LUC Cryptosystems is approximately equal to the amount needed for the RSA cryptosystems (Rivest *et al*., 1978). Horster *et al*. (1996), the author discussed another approaches of using Lucas functions. Full Lucas Functions computation that involved both sequences U_n and V_n are shown in (Horster *et al*., 1996). The cubic analogue of Lucas Functions is discussed in (Castagnos, 2007).

Meanwhile, the work in (Chiou and Laih, 2000) shows different technique compared to the work in (Ali *et al*., 2010). Smith and Lennon (1993) also provided two factors that give an impact to the performance and behavior of calculation of LUC cryptosystems, there are:

(a) Computations of V_e and V_d are huge and require long computation for large values of e and d
(b) The private-key d has to be recomputed for each block of messages

The related techniques to overcome these factors are provided. For factor (a) above, the computation of Lucas Function, V_e is almost the same as computation of powers. Therefore, the fast technique for the computation of powers can also be implemented on the computation of Lucas Functions.

Meanwhile, factor (b) shows that more computation works are required in the calculation of private key d than in the calculation of the private key for RSA systems. One of the techniques to compute private key d is shown in (Ali *et al*., 2010). However, this technique is only work when the value of two primes p and q are known.

In this study, the new computation technique that required smaller numbers of computation compare to the existing computation algorithm will be discussed. Briefly, the proposed algorithm is based on Addition Chain.

Two variables such w and t are introduced in order to design an efficient computational algorithm. These values could be used in order to skip some redundant computation of LUC Cryptosystem.

As a result the number of computation is reduced. At the same time, the computations time were also reduced. The efficiency of computation algorithm was increased.

MATERIALS AND METHODS

LUC cryptosystem: Lucas Functions are the second order linear recurrence relation using large public integer as modulo. These functions used in LUC Cryptosystem.

These functions are used in encryption and decryption processes. It is clear that LUC Cryptosystem is a recurrence relation based cryptosystem compared to the RSA that was based on exponentiation (Ribenboim, 1988).

Some important Lucas Functions properties are as follows (Horster *et al*., 1996; Ribenboim, 1988; Smith and Lennon, 1993 Williams, 1982):

$$V_n = PV_{n-1}-V_{n-2} \quad (1)$$

$$V_{2n} = V_n^2 - 2Q^n \quad (2)$$

$$V_{2n-1} = V_nV_{n-1} - PQ^{n-1} \quad (3)$$

$$V_{2n+1} = PV_n^2 - QV_nV_{n-1} - PQ^n \quad (4)$$

$$V_n^2 = DU_n^2 + 4Q^n \quad (5)$$

Initial values are $V_0 = 2$ and $V_1 = P$. While, $D = P^2-4Q$ is discriminate. Two functions U_n and V_n in Lucas sequences are defined as follows:

$U_0 = 0, U_1 = 1; U_n= PU_{n-1}–QU_{n-2}$ for $n\geq2$
$V_0 = 2, V_1 = P; V_1 = PV_{n-1}-QV_{n-2}$ for $n\geq2$

Only concentrates on function V_n with $Q = 1$. The computations of V_n need enormous computations (Horster *et al*., 1996; Joye and Quisquater, 1996) since the nature of Lucas Functions are based on recurrence relation.

In Lucas Functions computation, the computation of V_n needs two values of previous computation. The initial values should be V_0 and V_1.

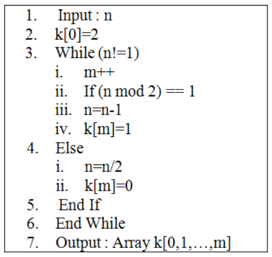
A ciphertext, C is obtained by encrypting the message, P by E (P) = $V_e(P,1)$(mod N) = C (mod N). Where, V_e is a Lucas Function and e is a public key. While, the decryption function is applied to C by D(C) = $V_d(C,1) = V_d(V_e(P,1),1) = V_{ed}(P,1)$ = P(mod N). Where, V_d is a Lucas Function and d is a private key.

An existing algorithm: The existing algorithm can be found in (Ali *et al*., 2010). In this algorithm, the generated Addition Chain will be used. The following ideas should be considered:

(a) An Addition Chain provides an idea of skipping the iterations steps needed to compute Lucas Functions.
(b) The computation of V_{2n}, V_{2n-1} and V_{2n+1} is done for every k[m] = 0
(c) No computation need for every k[m] = 1 and the only thing to do here is to initiate the next value to be used for next computation

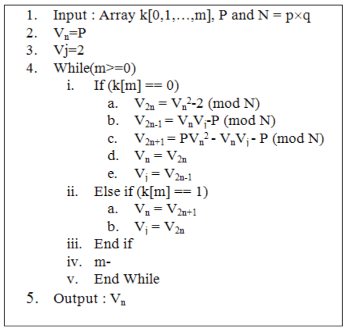
Detail explanation on this algorithm can be found in (Ali *et al*., 2010). The algorithm to generate Addition Chain is shown in Fig. 1, while the algorithm on how to use it is shown in Fig. 2.

The generated array k [0,1,…m] should be used in backward. It means that the computation start with k [m], k [m-1] until k[0] and k[0] used to stop the iterations. In Fig. 2, V_n^2 is computed twice, first in line 4.i.a and secondly in line 4.i.c.



```
1. Input : n
2. k[0]=2
3. While (n!=1)
   i.   m++
   ii.  If (n mod 2) == 1
   iii. n=n-1
   iv.  k[m]=1
4. Else
   i.   n=n/2
   ii.  k[m]=0
5. End If
6. End While
7. Output : Array k[0,1,…,m]
```

Fig. 1: An algorithm to generate array k [0,1,…,m]



```
1. Input : Array k[0,1,…,m], P and N = p×q
2. $V_n$=P
3. Vj=2
4. While(m>=0)
   i.   If (k[m] == 0)
        a. $V_{2n} = V_n^2$-2 (mod N)
        b. $V_{2n-1} = V_nV_j$-P (mod N)
        c. $V_{2n+1} = PV_n^2 - V_nV_j$ - P (mod N)
        d. $V_n = V_{2n}$
        e. $V_j = V_{2n-1}$
   ii.  Else if (k[m] == 1)
        a. $V_n = V_{2n+1}$
        b. $V_j = V_{2n}$
   iii. End if
   iv.  m-
   v.   End While
5. Output : $V_n$
```

Fig. 2: Existing algorithm for Luc cryptosystem based on addition chain

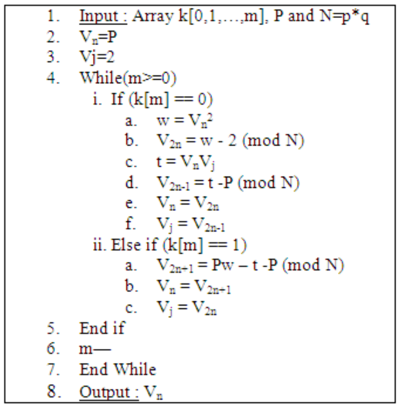
It is also clear that V_nV_j is also compute twice as shown in line 4.i.b and 4.i.c. Meanwhile, V_nV_j is also computed twice as can found it in line 4.i.b and line 4.i.c. These redundant computations can increase the computation time and surely decrease the efficiency of computation algorithm.

On the other hand, in line 4.i.c, the computation of (4) can be skipped. This is unnecessary computation step found in this algorithm.

Therefore, the computation of this function for each k[m] = 1. Table 1 shows the illustration of LUC Cryptosystem computation for V_{1103} for the existing algorithm.

Remember that, array k [0,1,…,m] is generated by Fig. 1. The generating of array k did not show in this study, because the detail of generating array k can be found in (Ali *et al*., 2010).

Noted that, 'Yes' indicates the required computation, while 'No' indicates no computation is required.



```
1.  Input : Array k[0,1,…,m], P and N=p*q
2.  $V_n$=P
3.  Vj=2
4.  While(m>=0)
      i. If (k[m] == 0)
           a.  $w = V_n^2$
           b.  $V_{2n} = w - 2 \pmod N$
           c.  $t = V_nV_j$
           d.  $V_{2n-1} = t - P \pmod N$
           e.  $V_n = V_{2n}$
           f.  $V_j = V_{2n-1}$
      ii. Else if (k[m] == 1)
           a.  $V_{2n+1} = Pw - t - P \pmod N$
           b.  $V_n = V_{2n+1}$
           c.  $V_j = V_{2n}$
5.  End if
6.  m—
7.  End While
8.  Output : $V_n$
```

Fig. 3: Algorithm on reducing the number of computation

Proposed algorithm: This algorithm shows another computation technique for LUC Cryptosystems. This technique is definitely better than the sum redundancy computation in the existing algorithm. The using of two variables w and t can reduce redundancy in computation of LUC Cryptosystem. The main purpose of this algorithm is to reduce the computation of Lucas Functions. The improvements from the existing algorithm are shown in the following facts:

(a) The values of w and t are computed in every k[m] = 1, therefore this technique can reduce some computations
(b) The value w is calculated from V_n^2 and t is calculated from V_nV_j
(c) Clearly, both values of w and t are small parts of Eq. 2 and 3
(d) In detail, the computation of V_{2n} and V_{2n-1} is done for every k [m] = 0
(e) The computation of V_{2n+1} is done for every k [m] = 1 using w and t that are calculated previously
(f) The next values to be used such as V_n and V_j are also initiated in each k[m] = 0 and k [m] = 1

Figure 3 shows the proposed algorithm which is concentrated on reducing some redundant computations.

Two variables w and t are introduced in such away to reduce some computation steps. It is clearly shown in lines 4.i.a and 4.i.t.

These values are then used in line 4.ii.a. Table 2 shows the illustration of LUC Cryptosystem computation for V_{1103} for the proposed algorithm. Once again, noted that the array k [0, 1,..., m] is generated by Fig. 1.

Table 1: Existing algorithm for computation of V_{1103}

M	k[m]	V_n	V_j	V_{2n}	V_{2n+1}	V_{2n-1}
15	0	V_2	V_1	Yes	Yes	Yes
14	0	V_4	V_3	Yes	Yes	Yes
13	0	V_8	V_7	Yes	Yes	Yes
12	0	V_{16}	V_{15}	Yes	Yes	Yes
11	1	V_{17}	V_{16}	No	No	No
10	0	V_{34}	V_{33}	Yes	Yes	Yes
9	0	V_{68}	V_{67}	Yes	Yes	Yes
8	0	V_{136}	V_{135}	Yes	Yes	Yes
7	1	V_{137}	V_{136}	No	No	No
6	0	V_{274}	V_{273}	Yes	Yes	Yes
5	1	V_{275}	V_{274}	No	No	No
4	0	V_{550}	V_{549}	Yes	Yes	Yes
3	1	V_{551}	V_{550}	No	No	No
2	0	V_{1102}	V_{1101}	Yes	Yes	Yes
1	1	V_{1103}	V_{1102}	No	No	No
0	0	V_{1103}	V_{1102}	No	No	No

Table 2: Proposed algorithm for V_{1103}

M	k[m]	V_n	V_j	V_{2n}	V_{2n+1}	V_{2n-1}	w	t
15	0	V_2	V_1	Yes	No	Yes	Yes	Yes
14	0	V_4	V_3	Yes	No	Yes	Yes	Yes
13	0	V_8	V_7	Yes	No	Yes	Yes	Yes
12	0	V_{16}	V_{15}	Yes	No	Yes	Yes	Yes
11	1	V_{17}	V_{16}	No	Yes	No	No	No
10	0	V_{34}	V_{33}	Yes	No	Yes	Yes	Yes
9	0	V_{68}	V_{67}	Yes	No	Yes	Yes	Yes
8	0	V_{136}	V_{135}	Yes	No	Yes	Yes	Yes
7	1	V_{137}	V_{136}	No	Yes	No	No	No
6	0	V_{274}	V_{273}	Yes	No	Yes	Yes	Yes
5	1	V_{275}	V_{274}	No	Yes	No	No	No
4	0	V_{550}	V_{549}	Yes	No	Yes	Yes	Yes
3	1	V_{551}	V_{550}	No	Yes	No	No	No
2	0	V_{1102}	V_{1101}	Yes	No	Yes	Yes	Yes
1	1	V_{1103}	V_{1102}	No	Yes	No	No	No
0	0	V_{1103}	V_{1102}	No	No	No	No	No

RESULTS

In this study, the efficiency of those algorithms will be tested with very large size of keys, messages and primes. These three different situations may produce different computation time. When the size of keys, messages and primes are increased; the computation time is increased.

The public key e would also be relatively large integer, because e is chosen so that e is relatively primes to p and q. The primes p and q should also be in large size. Therefore, the value of N is also large, because N = p×q. It is a property of the encryption technique that modest increase in computational cost can produce vast increases in security.

In many applications, the cryptography itself accounts for only a small fraction of the computing cost, compared to such processes of video, voice or image compressions required to prepare the material of encryption. The size of encryption keys could strengthen the security of the systems and the difficulty of all possible keys grows exponentially with the number of bits used. The bigger the primes size, the more work is needed to compute modulo N.

DISCUSSION

The discussion on these algorithms will look into all aspect that could give an impact onto their performance. The following symbols are useful in the discussion:

(a) M is a total number of iterations for Addition Chain
(b) R is a total number of k [m] = 0 for Addition Chain
(c) s is total number of k [m] = 1 for Addition Chain

Based on the notation of symbols above, the simplified number of computation for the existing algorithm as 9r and the number of computation for proposed algorithm as 4r+3s.

Table 3: Computation time on different size of public key

Public Key e (digits)	Encryption (sec) Existing	Proposed	Decryption (sec) Existing	Proposed
19	7.78	6.79	190.56	155.79
159	88.68	68.73	255.10	211.56
339	380.26	301.34	414.01	393.31
579	775.88	687.99	417.05	389.51

Table 4: Computation time on different size of messages

Messages P (digits)	Encryption (sec) Existing	Proposed	Decryption (sec) Existing	Proposed
20	31.98	28.33	1497.65	1301.43
160	36.09	31.54	1441.57	1276.54
250	39.69	36.44	1440.48	1345.32
390	46.98	42.44	1442.57	1365.35

Table 6-8 show the combination of r and s for Table 3-5. The total of 1 and 0 must equal to the total number of iterations for Addition Chain.

Consider the situation of Addition Chain (Table 6) with the size of public key is e = 579 with 2868 iterations, s = 948 and r = 1920. The total number of computation for existing algorithm is 17280 (9×1920).

On the other hand, the total number of computation for proposed algorithm is only 10524 (4×1920+3×948). The efficiency of those algorithms also gives an impact to the decryption process.

Therefore, decryption process required 5940 (9×660) computations for existing algorithm. Meanwhile, the proposed algorithm only required 3636 (4×660+3×332). Total number of computations for numerical experiments is shown in Table 9-11.

The proposed algorithm has the smallest total number of computations compared to the existing one. As the size of primes increase, the size of private keys also increases. The size of private key d will also increase in almost double of the prime sizes.

Table 5: Computation time for existing and proposed algorithms on different size of primes

Primes p and q (digits)	Encryption (sec) Existing	Proposed	Decryption (sec) Existing	Proposed
100	88.75	74.12	209.58	175.54
160	246.60	205.78	972.86	834.12
220	344.45	295.99	1830.33	1683.44
280	456.61	400.12	3072.36	2734.34

Table 6: Combination of s and r, for different size of public key

Public key e	Encryption s	r	Decryption s	r
19	36	59	324	660
159	292	525	345	659
339	561	1122	317	660
579	948	1920	332	660

Table 7: Combination of s and r, for different size of messages

Messages P	Encryption s	r	Decryption s	r
20	36	59	699	1324
160	36	59	699	1324
250	36	59	656	1321
390	36	59	656	1321

Table 8: Combination of s and r, for different size of primes

Primes p and q	Encryption s	r	Decryption s	r
100	292	525	331	656
160	292	525	537	1059
220	292	525	715	1449
280	292	525	874	1854

Table 9: Number of computations for different size of public key

	Encryption		Decryption	
E	Existing	Proposed	Existing	Proposed
19	531	334	5940	3612
159	4725	2976	5931	3671
339	10098	6171	5940	3591
579	17280	10524	5940	3636

Table 10: Number of computations for different size of messages

	Encryption		Decryption	
P	Existing	Proposed	Existing	Proposed
20	531	334	11916	7393
160	531	334	11916	7393
250	531	334	11889	7252
390	531	334	11889	7252

Table 11: Number of computations for different size of primes

	Encryption		Decryption	
p and q	Existing	Proposed	Existing	Proposed
100	4725	2976	5904	3617
160	4725	2976	9531	5847
220	4725	2976	13041	7941
280	4725	2976	16686	10038

Overall, the popular Moore's Law could be useful to estimate the minimum strength required as a function of time. Taking into account both the life span of cryptographic equipment and the secrets it protects.

CONCLUSION

The proposed algorithm shows better computation time and smaller total number of computations. The algorithms are proposed to manipulate the Addition Chain. All algorithms have been implemented to increase the speed of LUC Cryptosystem computation. The proposed algorithm surely shows better speed and efficiency by reducing some redundant computation steps.

The longer the Addition Chain, the longer computation time is needed. Therefore, the size of array k[m] generated by Addition Chain is always depends on the size of public key or private keys. Thus, increasing the key size to gain greater security is feasible by the computation algorithm. Surely, the size of primes plays important roles in LUC Cryptosystems computations.

The two primes, p and q built up modulo N used in LUC computations. It is also found that, the size of private key is almost double when the size of primes is increased. The proposed algorithm proposed can speed up the computations of LUC cryptosystems compared to the existing algorithm (Ali *et al*., 2010).

The proposed techniques discussed here leads to high reduction in the multiplications required for both encryption and decryption operations without sacrificed the key size of LUC Cryptosystem security. It makes the LUC cryptosystem computations more efficient for security implementation.

The improvement of the proposed algorithms in this study can be focused on: (a) Generating shorter sequence compare to Addition Chain and (b) Finding new property of Lucas Functions.

ACKNOWLEDGEMENT

The researchers like to thank the Govement of Malaysia for Study Leave and their Sponsorship (SLAB).

REFERENCES

Ali, Z.M., M. Othman, M.R.M. Said and M.N. Sulaiman, 2010. Computation of cryptosystem based on Lucas functions using addition chain. Proceedings of the International Symposium on Information Technology (IT'10), Kuala Lumpur, pp: 1082-1086.

Chiou, S. and C.S Laih, 2000. An efficient algorithm for computing the LUC chain. IEEE Proc. Comput. Digit. Tech., 147: 263-265.

Castagnos, G., 2007. An Efficient probabilistic public-key cryptosystem over quadratic fields' quotients. Finite Field Appli., 13: 563-576.

Horster, P., M. Michels and H. Petersen, 1996. Digital Signature schemes based on Lucas functions, Proceedings of the 1st International Conference on Communications and Multimedia Security, (CMS'96), Chapman and Hall, Graz, pp: 178-190.

Joye, M. and J.J. Quisquater, 1996. Efficient computation of full Lucas sequences. IEEE Elect. Lett., 32: 537-538.

Ribenboim, P., 1988. The Book of Prime Number Records. 1st Edn., Springer-Verlag, New York, Berlin, Heidelberg, pp: 777.

Rivest, R.L., A. Shamir and L.M. Adleman, 1978. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21: 120-126.

Smith, P.J. and M.J.J. Lennon, 1993. LUC: A New Public Key System, Proceedings of the 9th IFIP Symposium on Computer Security, May 12-14, North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, pp: 103-117.

Williams, H.C., 1982. A p+1 method of factoring. Math. Comput., 39: 225-234.