Two Fast Computation Algorithms for LUC Cryptosystems

Z.M. Ali\textsuperscript{1}, M. Othman\textsuperscript{2}, M.R.M Said\textsuperscript{3} and M.N. Sulaiman\textsuperscript{2}

\textsuperscript{1}Computer Science Department, Faculty of Technology and Information Science, Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia.
\textsuperscript{2}Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia.
\textsuperscript{3}Department of Mathematics, Faculty of Science, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia.

Abstract
Most of public-key cryptosystems are based on one-way functions, which can be used to encrypt and sign messages. Among them, probably the most promising and widely used is the encryption and signature operations are based on the computation of exponentiation (as in the RSA). Another public-key cryptosystems are proposed and based on Lucas Functions which was known as LUC cryptosystems. The Lucas Functions is a special form of second-order linear recurrence relation using a large public integer as the modulus. The performances of its computations are influence by computation of $V_e$ and $V_d$, the public and private keys process, respectively. In this paper, we are presenting two fast computation algorithms for LUC cryptosystems. We are manipulated some properties of Lucas Functions relationships by introducing two different sequences. These sequences will provide an efficient technique to compute LUC cryptosystems. In order to compare the efficiency of each algorithm, we show computation time for each algorithm.

1. Introduction.
The best known and most widely used trapdoor function(2) for public-key cryptosystems is the exponentiation-based(7). Smith and Lennon(8) introduced a public key based on Lucas function instead of exponentiation, which offer a good alternative.

The reason of using Lucas functions instead of exponentiations is it cryptographically strength. It is much stronger than or at least strong as the exponentiation-based systems.

Performance has always been the most critical issues of a cryptographic function, which determines its effectiveness. Some previous attempts on speeding up the LUC cryptosystems computations can be found in(1)(9)(11). While in(4), the authors proposed an algorithm to compute full Lucas Sequences. So how to efficiently compute $V_e$ (mod N), is a big challenge, for $e$ (public-key), $p$ and $q$ to be a random big integers. N is also big where $N=p\times q$. A $d$ (private-key) can be calculated as suggested in (8).

Our algorithms are differ from all previous attempts, where we introduce two special sequences; and proposed two fast computation algorithms, Algorithm A and Algorithm B.

2. LUC Cryptosystems.
We only included minimal amount of information needed to understand this paper, details can be found in(7).

2.1 Lucas Functions.
Assume that $P_1$, $P_2$, $P_3$... $P_n$ are integers, a Lucas Function is a sequence of integers, $T_n$ defined as follows:

$T_n = P_1T_{n-1} + P_2T_{n-2} + \ldots + P_nT_{n-m}$

The computations of $V_n$ need enormous computations (3)(6) because of their nature of recurrence relations. Two functions $U_n$ and $V_n$ in Lucas sequences are defined as follows ($P$ and $Q$ are two relatively prime numbers):

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

2.2 Encryption and Decryption for LUC cryptosystems.
A ciphertext, $C$ is obtained by encrypting the message, $M$ by $Enc(M) = V_e(M,1) \mod N = C \mod N$, $V_e$ is a Lucas Function.

While, the decryption function is applied to $C$ by $Dec(C) = V_d(C,1) = V_d(V_e(M,1),1) = V_{ed}(M,1) = M \mod N$.

2.3 Lucas Functions Relationships Properties.
We are manipulating some important properties that can be found in (3)(8)(10).

Among them are:

$V_n = MV_{n-1} - V_{n-2}$ \quad (1)
$V_{2n} = (V_n)^2 - 2Q^n$ \quad (2)
$V_{2n-1} = V_0V_{n-1} - MQ^{n-1}$ \quad (3)
$V_{2p+1} = M(V_0)^2 - QV_nV_{n-1} - MQ^n$ \quad (4)
$V_{2} = DU_2 + 4Q^0$ \quad (5)

With initial value $V_0=2$ and $V_1=P$. While, $D=P^2 - 4Q$ is a discriminant. We only concentrate on $V_n$ with $Q=1$.
3. A Propose Algorithms.

The computations of LUC cryptosystems with inefficient algorithm required huge time and space. Our main concern here is how to speed up the computation of $V_e$ and $V_d$. We need to evaluate $V_n \pmod{N}$ in a recursive manner, where $N=p\times q$. To better understand of a propose algorithms, we use C-like notation to show our algorithms. A denotes Algorithm A and B for Algorithm B.

3.1 Algorithm A.

A special sequence is needed in order to organize the computations by choosing the index value of sequence. We use:

a) 2: terminate.
b) 1: compute $V_n=V_{2n+1}$ using Eq.(1) and $V_j=V_{2n}$ using Eq.(2).
c) 0: compute $V_n=V_{2n}$ using Eq.(2) and $V_j=V_{2n-1}$ using Eq.(3).

Input: $n$, $x=0$
Output: Array $k$

\[
\begin{align*}
& k[0] = 2; \\
& \text{While (} n!\neq 1) \\
& \quad x++; \\
& \quad \text{If} (n \mod 2) == 1 \\
& \quad \quad n=n-1; \\
& \quad \quad k[x] = 1; \\
& \quad \text{Else} \\
& \quad \quad n=n/2; \\
& \quad \quad k[x] = 0; \\
& \quad \text{End If} \\
& \text{End While}
\end{align*}
\]

Fig. 1. Algorithm A (Part 1)

Input: Array $k$, $N$, $x$, $M$
Output: $V_n$

\[
\begin{align*}
& \text{Compute } V_2, V_3, V_4 \\
& \text{If } k[x] = 1 \text{ Then} \\
& \quad V_{2n} = V_3; \\
& \quad V_{2n+1} = V_4; \\
& \text{Else} \\
& \quad V_{2n} = V_2; \\
& \quad V_{2n+1} = V_3; \\
& \text{End If} \\
& \text{While (} x!=1) \\
& \quad \text{If } k[j] = 1 \text{ Then} \\
& \quad \quad V_i = V_{2n+1} \star M - V_{2n} \pmod{N}; \\
& \quad \quad V_{2n+1} = V_{2n+1} \star V_{2n+1} - V_{2n} \pmod{N}; \\
& \quad \quad V_{2n+1} = V_{2n+1} \star V_i - M \pmod{N}; \\
& \quad \text{Else} \\
& \quad \quad V_{2n+1} = V_{2n} \star V_{2n+1} - M \pmod{N}; \\
& \quad \quad V_{2n+1} = V_{2n} \star V_{2n} - 2 \pmod{N}; \\
& \quad \text{End If} \\
& \quad x--; \\
& \text{End While}
\end{align*}
\]

Fig. 2. Algorithm A (Part 2)

3.2 Algorithm B.

In this algorithm, the sequence is similar as we convert some value in decimal numbers into binary numbers. We use:

a) 1: compute $V_i=V_{2n+1} \star P - V_{2n}$ using Eq.(1), $V_{2n}=V_{2n+1} \star V_{2n+1} - 2$ using Eq.(2) and $V_{2n+1} = V_{2n+1} \star V_i - P$ using Eq.(1);
b) 0: compute $V_{2n+1} = V_{2n} \star V_{2n+1} - P$ using Eq.(1) and $V_{2n}=V_{2n} \star V_{2n+1} - 2$ using Eq.(2).

Input: $n$, $x=0$
Output: Array $k$

\[
\begin{align*}
& \text{temp1} = n/2; \\
& \text{temp2} = n \mod 2 \\
& \text{If (} \text{temp2} == 1) \\
& \quad k[x] = 1; \\
& \quad n = \text{temp1}; \\
& \text{Else} \\
& \quad k[x] = 0; \\
& \quad n = \text{temp1}; \\
& \text{End if} \\
& x++; \\
& \text{End While}
\end{align*}
\]

Fig. 3. Algorithm B (Part 1)

Input: Array $k$, $N$, $x$, $M$
Output: $V_n$

Compute $V_2$, $V_3$, $V_4$

\[
\begin{align*}
& \text{If } k[x] = 1 \text{ Then} \\
& \quad V_{2n} = V_3; \\
& \quad V_{2n+1} = V_4; \\
& \text{Else} \\
& \quad V_{2n} = V_2; \\
& \quad V_{2n+1} = V_3; \\
& \text{End If} \\
& \text{While (} x!=1) \\
& \quad \text{If } k[j] = 1 \text{ Then} \\
& \quad \quad V_i = V_{2n+1} \star M - V_{2n} \pmod{N}; \\
& \quad \quad V_{2n} = V_{2n+1} \star V_{2n+1} - 2 \pmod{N}; \\
& \quad \quad V_{2n+1} = V_{2n+1} \star V_i - M \pmod{N}; \\
& \quad \text{Else} \\
& \quad \quad V_{2n+1} = V_{2n} \star V_{2n+1} - M \pmod{N}; \\
& \quad \quad V_{2n} = V_{2n} \star V_{2n} - 2 \pmod{N}; \\
& \quad \text{End If} \\
& \quad x--; \\
& \text{End While}
\end{align*}
\]

Fig. 4. Algorithm B (Part 2)
4. Implementations and Results

We show some examples and results on both algorithms.

4.1 Implementations.

In A, if we need to calculate $V_{1103}$, then $n=1103$, our sequence would produce an array $k$ as \{2,1,0,1,0,0,0,1,0,0,0\}. So that, $m=15$. We have to use this sequence in backward, then $k[m]=\{0,0,0,1,0,1,0,1,0,1,0,1,0,1,1,1\}$. While, in B, our sequence is \{1,1,1,1,0,1,1,0,0,0\}. Then $j=10$. We have to use this sequence in backward, then $k[x]=\{0,0,0,1,1,0,1,1,1,1\}$. Examples of required computations for $V_{1103}$ have been shown in Fig. 5 and 6.

<table>
<thead>
<tr>
<th>$k[m]$</th>
<th>$V_n$</th>
<th>$V_j$</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$V_2$</td>
<td>$V_1$</td>
<td>$m=15$; Calculate $V_n$, $V_j$</td>
</tr>
<tr>
<td>0</td>
<td>$V_4$</td>
<td>$V_3$</td>
<td>$m=14$; Calculate $V_n$, $V_j$</td>
</tr>
<tr>
<td>0</td>
<td>$V_8$</td>
<td>$V_7$</td>
<td>$m=13$; Calculate $V_n$, $V_j$</td>
</tr>
<tr>
<td>0</td>
<td>$V_{16}$</td>
<td>$V_{15}$</td>
<td>$m=12$; Calculate $V_n$, $V_j$</td>
</tr>
<tr>
<td>1</td>
<td>$V_{17}$</td>
<td>$V_{16}$</td>
<td>$m=11$; Calculate $V_{n+1}$</td>
</tr>
<tr>
<td>0</td>
<td>$V_{34}$</td>
<td>$V_{33}$</td>
<td>$m=10$; Calculate $V_n$, $V_j$</td>
</tr>
<tr>
<td>0</td>
<td>$V_{48}$</td>
<td>$V_{47}$</td>
<td>$m=9$; Calculate $V_n$, $V_j$</td>
</tr>
<tr>
<td>0</td>
<td>$V_{156}$</td>
<td>$V_{155}$</td>
<td>$m=8$; Calculate $V_n$, $V_j$</td>
</tr>
<tr>
<td>1</td>
<td>$V_{137}$</td>
<td>$V_{136}$</td>
<td>$m=7$; Calculate $V_{n+1}$</td>
</tr>
<tr>
<td>0</td>
<td>$V_{274}$</td>
<td>$V_{273}$</td>
<td>$m=6$; Calculate $V_{n+1}$</td>
</tr>
<tr>
<td>1</td>
<td>$V_{275}$</td>
<td>$V_{274}$</td>
<td>$m=5$; Calculate $V_{n+1}$</td>
</tr>
<tr>
<td>0</td>
<td>$V_{550}$</td>
<td>$V_{549}$</td>
<td>$m=4$; Calculate $V_n$, $V_j$</td>
</tr>
<tr>
<td>1</td>
<td>$V_{551}$</td>
<td>$V_{550}$</td>
<td>$m=3$; Calculate $V_{n+1}$</td>
</tr>
<tr>
<td>0</td>
<td>$V_{1102}$</td>
<td>$V_{1101}$</td>
<td>$m=2$; Calculate $V_n$, $V_j$</td>
</tr>
<tr>
<td>1</td>
<td>$V_{1103}$</td>
<td>$V_{1102}$</td>
<td>$m=1$; Calculate $V_{n+1}$</td>
</tr>
<tr>
<td>2</td>
<td>$V_{1103}$</td>
<td>$V_{1102}$</td>
<td>$m=0$; Output: $V_{1103}$</td>
</tr>
</tbody>
</table>

Figure 5: Illustration of Algorithm A (Part 2).

Table 1. Time required for Different key sizes.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.541</td>
<td>0.741</td>
<td>133.182</td>
<td>141.132</td>
</tr>
<tr>
<td>19</td>
<td>5.288</td>
<td>9.384</td>
<td>143.967</td>
<td>160.852</td>
</tr>
<tr>
<td>79</td>
<td>26.878</td>
<td>37.624</td>
<td>163.585</td>
<td>174.401</td>
</tr>
<tr>
<td>159</td>
<td>56.927</td>
<td>65.854</td>
<td>192.256</td>
<td>197.173</td>
</tr>
<tr>
<td>339</td>
<td>285.730</td>
<td>269.758</td>
<td>322.783</td>
<td>326.670</td>
</tr>
<tr>
<td>499</td>
<td>497.805</td>
<td>446.432</td>
<td>325.488</td>
<td>328.252</td>
</tr>
<tr>
<td>579</td>
<td>608.955</td>
<td>545.706</td>
<td>329.684</td>
<td>368.482</td>
</tr>
<tr>
<td>619</td>
<td>674.670</td>
<td>624.658</td>
<td>284.660</td>
<td>314.432</td>
</tr>
</tbody>
</table>

Table 2. Time required for Different messages sizes.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>19.648</td>
<td>25.206</td>
<td>1050.700</td>
<td>1103.453</td>
</tr>
<tr>
<td>80</td>
<td>21.271</td>
<td>26.889</td>
<td>1034.177</td>
<td>1047.866</td>
</tr>
<tr>
<td>160</td>
<td>22.623</td>
<td>28.421</td>
<td>1047.466</td>
<td>1068.436</td>
</tr>
<tr>
<td>190</td>
<td>23.334</td>
<td>28.001</td>
<td>1031.023</td>
<td>1058.582</td>
</tr>
<tr>
<td>250</td>
<td>24.105</td>
<td>37.203</td>
<td>1030.301</td>
<td>1080.304</td>
</tr>
<tr>
<td>330</td>
<td>26.187</td>
<td>32.807</td>
<td>1050.811</td>
<td>1061.777</td>
</tr>
<tr>
<td>390</td>
<td>28.531</td>
<td>32.757</td>
<td>1033.266</td>
<td>1035.999</td>
</tr>
</tbody>
</table>

Table 3. Number of Iterations (Different Key Size).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
<td>199</td>
<td>985</td>
</tr>
<tr>
<td>19</td>
<td>95</td>
<td>57</td>
<td>199</td>
<td>984</td>
</tr>
<tr>
<td>79</td>
<td>402</td>
<td>257</td>
<td>199</td>
<td>985</td>
</tr>
<tr>
<td>159</td>
<td>817</td>
<td>523</td>
<td>199</td>
<td>1004</td>
</tr>
<tr>
<td>339</td>
<td>1683</td>
<td>1120</td>
<td>199</td>
<td>977</td>
</tr>
<tr>
<td>499</td>
<td>2467</td>
<td>1652</td>
<td>199</td>
<td>977</td>
</tr>
<tr>
<td>579</td>
<td>2868</td>
<td>1918</td>
<td>199</td>
<td>992</td>
</tr>
<tr>
<td>619</td>
<td>3084</td>
<td>2051</td>
<td>199</td>
<td>988</td>
</tr>
</tbody>
</table>

Table 1, 2 and 3 shows overall time comparison for Algorithm A and B for both encryption and decryption for different key sizes, messages size and prime p and q sizes. All results are based on running time for each algorithm in C language with Windows XP environment, Crusoe Processor TM5800.

4.2 Results.

We only record the computations time. Table 1, 2 and 3 show all computations time for A and B. While, Table 4 shows a number of iterations required for different key size only.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
<td>199</td>
<td>985</td>
<td>658</td>
</tr>
<tr>
<td>19</td>
<td>95</td>
<td>57</td>
<td>199</td>
<td>984</td>
<td>658</td>
</tr>
<tr>
<td>79</td>
<td>402</td>
<td>257</td>
<td>199</td>
<td>985</td>
<td>658</td>
</tr>
<tr>
<td>159</td>
<td>817</td>
<td>523</td>
<td>199</td>
<td>1004</td>
<td>657</td>
</tr>
<tr>
<td>339</td>
<td>1683</td>
<td>1120</td>
<td>199</td>
<td>977</td>
<td>658</td>
</tr>
<tr>
<td>499</td>
<td>2467</td>
<td>1652</td>
<td>199</td>
<td>977</td>
<td>658</td>
</tr>
<tr>
<td>579</td>
<td>2868</td>
<td>1918</td>
<td>199</td>
<td>992</td>
<td>658</td>
</tr>
<tr>
<td>619</td>
<td>3084</td>
<td>2051</td>
<td>199</td>
<td>988</td>
<td>658</td>
</tr>
</tbody>
</table>
In Algorithm B from the total computations are 4r + 2s. Where r is a total of k[m-i]=0, u is a total of k[m-i]=1 and 0 \leq i \leq m. Then, algorithm A required (r*2+s) multiplications and (r*2+s) subtractions, where the total computations are 6t + 4u.

Let m be a size of array k for A, t is a total of k[m-i]=0, u is a total of k[m-i]=1 and 0 \leq i \leq m. Then, algorithm A required (r*2+s) multiplications and (r*2+s) subtractions, were the total computations are 6t + 4u.

Let j be a size of array k for B, t is a total of k[j-i]=0, u is a total of k[j-i]=1 and 0 \leq i \leq j. Then, algorithm B required (t*3+2*u) multiplications and (t*3+2*u) subtractions, where the total computations are 6t + 4u.

We already know that m > j, surely the size of array k in A is bigger than B. But the efficiency of computations in A is better than B. In the situation of key size <= 300 digits, we found that A can perform better than B. Otherwise, if key size > 300 digits, B can perform better than A.

We also show the reductions of number of multiplications and subtractions. Therefore, the computation time can be reduced in the proposed method. In each algorithm the time need to calculate that particular sequences are approximately 10% in Algorithm A and 8% in Algorithm B from the total of computation time required to compute LUC cryptosystems.

6. Conclusions and further research.
We found that, for private-key process, different outcomes were found where changing the key sizes the private-key has same length in digits for all data. In other situation, changing the messages size, the private-key has almost same length in digits for all data.

Changing the prime size, the private-key has different length in digits. So that, the private-key process for changing primes size required more computation time for different sets of data. Both algorithms can speed up the computations of LUC cryptosystems. We can also improve it by combining the efficiency of computations in A (part 2), and use the sequence in B (part 1).

As shown in Table 1, 2 and 3, this algorithm proved that the speed can be increased by reducing a number of steps of multiplication and a number of iterations. It makes the LUC cryptosystem computations more efficient for security implementation. It is also leads to a high reduction in the multiplications required for both the encryption and decryption operations without sacrificing the key size of LUC cryptosystem security.

The shorter sequences will result the less number of modular multiplications. An interesting research topic falls into how efficiently generate a shorter sequence. Another interesting research topic is also on reducing some modular multiplications in Lucas functions itself.

References


