Enhancing security in cryptographic algorithm based on LECCRS

K. Mani

Department of Computer Science,
Nehru Memorial College,
Puthanampatti, Trichy, 621007, India
Email: nitishmanik@gmail.com

A. Devi*

Department of Computer Science,
Lowry Memorial College,
Bangalore, 560016, India
Email: devianbu75@gmail.com
*Corresponding author

Abstract: The efficiency of any cryptographic algorithms depends on both security and operational speed. When a message is encrypted using any cryptosystems it produces only one level of security because it converts the plaintext into ciphertext where the ciphertext is not easily tractable. To increase the security in any cryptographic algorithm, normally the size of the plaintext and key should be taken very largely. Further, additional levels of security have been introduced in this paper to enhance the security. For that, the message to be encrypted is encoded first so that original form of the message is altered called the first level of security. The encoded message is sometimes too large so that it is compressed which results in changing the encoded form of message called second level security. Finally, the compressed message is encrypted using any one of the public-key cryptosystems by determining the block size dynamically which produces the third level of security. Thus, an integrated approach is proposed called encoded compressed cryptosystem in this paper.

Keywords: Lucas encoding; Fibonacci encoding; Huffman compression; RSA encryption.


Biographical notes: K. Mani received his MCA and MTech from the Bharathidasan University, Trichy, India in Computer Applications and Advanced Information Technology, respectively. Since 1989, he has been in the Department of Computer Science at the Nehru Memorial College, affiliated to Bharathidasan University where he is currently working as an Associate Professor. He completed his PhD in Cryptography with primary emphasis on the evolution of framework for enhancing the security and optimising the run-time in cryptographic algorithms. He published and presented around 15 research papers at international journals and conferences.
A. Devi received her MCA and M.Phil from Bharathidasan University, Trichy, India in Computer Science Applications. During 2004–2016 (April), she had been with the Department of Computer Science at the Lowry Memorial College, affiliated to Bangalore University, Karnataka, India where she was working as an Associate Professor. During 1998–2001, she was working as a programer in different software companies. She is currently working as a Professor in Cavalier Animation and Media Science, affiliated to Mysore University, Karnataka, India. She is pursuing her PhD in Compressed Cryptosystem, Bharathidasan University, Trichy, India.

1 Introduction

Data security is provided over the internet by a variety of methods but straightforward security method is only to keep sensitive information on removable storage media like portable flash memory drives or external hard drives. The most popular form of security rely upon is using encryption which is the process of altering the information in such a way that only the person with the key can correctly decrypt it. It is the transformation of data into a form and its purpose is to ensure privacy by keeping information hidden from anyone who has access to the encrypted data. Decryption is the reverse of encryption, i.e., the transformation of encrypted data back into an intelligible form. Digital signature and encryption must be applied to the ciphertext when it is created to avoid tampering otherwise any node between the sender and the encryption agent could potentially tamper with it.

Cryptography includes both encryption and decryption. It is the study of methods of sending messages in disguised form so that only the intended recipients can remove the disguise and read the message. It involves the study of mathematical techniques that allow the practitioner to provide the security, authentication, and non-repudiation. A cryptographic system (Stinson, 2002) consists of a plaintext message space $M$, a ciphertext message space $C$, an encryption key space $K$, and the decryption key space $K'$, an efficient key generation algorithm $G: N \times K \times K'$, an efficient encryption algorithm $E: M \times K \times C$ and an efficient decryption algorithm $D: C \times K^{-1} \times M$. For $k \in K$, and $m \in M$, we denote by $C = E_k(m)$ and $m = D_k(C)$.

There are two major classes of algorithms in cryptography: private-key and public-key algorithms. In a private-key cryptosystem, encryption and decryption use the same key, i.e., $k_d = k_e$. In a public-key cryptosystem, encryption and decryption key use different keys: for every $k \in K'$, and $k_d \neq k_e$. The RSA public-key cryptosystem is the most popular form of public-key cryptography. It can be used not only for encryption but also for authentication and other various techniques (https://en.wikipedia.org/wiki/RSA).

Before the encryption is performed using any cryptographic algorithms, some coding scheme may be used to alter the plaintext. The conventional encoding scheme is ASCII in which each character is coded as byte which is a fixed length code. An alternate to ASCII coding is a Fibonacci coding, a variable length code which is based on Fibonacci numbers. It is a popular coding which may completely change the form of a character in a plaintext and may produce longer length code for some integer. An alternative to Fibonacci coding, Lucas coding is proposed in this paper which produces shorter coding
than the Fibonacci coding for any integer because the series starts with 2 and 1. As the length of the both Fibonacci and Lucas coding for the same integer is somewhat larger than the length of conventional binary form of ASCII value a character which presents in the plaintext and also to reduce the size of plaintext, we consider compression which eventually results in the shorter code for the encoded plaintext. As the number of bits required to represent plaintext is reduced drastically the time taken to transmit the data over a network will also be decreased or increased the available disk space.

Compression algorithms (Delfs and Knebl, 2001) exploit statistical redundancies in the data which should not be eliminated when the message is encrypted, therefore an encrypted message should not be able to compress that well. Compression after encoding of plaintext can help against some attacks. Various codes have been proposed in the literature for data compression and they are categorised as fixed length and variable length codes. Variable length codes use some statistical method in contrast with fixed-length codes. In variable length code, shorter codes are assigned to symbols or groups of symbols that have a higher probability of occurrence, and longer codes are assigned to the symbols or group of symbols that have a lower probability.

People who design and implement variable-length codes have to deal with these two problems namely, assigning codes that can be decoded unambiguously and assigning codes with the minimum average size. Huffman code (http://en.wikipedia.org/wiki/Data_compression) is one of the variable length code. It is a lossless data compression for encoding a character into some other form. The compressed code will be sent for encryption using known RSA public-key algorithm. In Mani and Devi (2015), we proposed a scheme called Integrated Encrypted Compressed CryptoSystem (IENCCRS) in increasing the speed for transmitting the ciphertext, the strength of cryptosystem and provides the double security of encryption through Fibonacci encoding scheme. It shows that the security level and the compression ratio is less in IENCCRS scheme. As the length of the coding the plaintext is increasing, to reduce the length an alternate coding scheme called Lucas Encoding Scheme is proposed for encoding the plaintext in this paper. It is proved that the security level before and after compression, the compression ratio is higher than Fibonacci coding. The number of bits is reduced after compression and the time taken by data transmission is also less.

The rest of the paper is organised as follows. Related work is presented in Section 2. Section 3 presents the mathematical background necessary for Lucas series and Fibonacci series, the procedure for encoding of an integer using Lucas coding with an example. The proposed methodology is discussed for Fibonacci and Lucas series with proper algorithm and example in Section 4. The compression of Lucas encoding using Huffman algorithm is discussed in Section 5. Modified RSA encryption with dynamic block size is discussed in Section 6. The Experimental result for integrated encryption of compressed encoding scheme of text is presented in Section 7. Finally, Section 8 ends with a conclusion.

2 Related work

Mahmoud et al. (2010) proposed an efficient technique of hybrid compression with encryption for securing of SMS in Symbian operating system. This technique uses the compression and encryption technique so that SMS sent through mobiles can be made secure. Here, RSA-based encryption technique is used to reduce the eavesdropping but encryption increases the size of the text message, hence bandwidth is not utilised.
Chu et al. (2011) proposed technique regarding secure subscription of mobile in sensor-encrypted data.

In SMS-SED, a node or a mobile device stores a secret key of size independent of the total number of sensor nodes and time periods. Khan et al. (2012) implemented a new technique of transmission of compressed audio through SMS. SMS cannot be used for transfer of audio data since SMS may contain less bandwidth. Hence the idea is to compress the audio signal data into text and compresses so that it will not take much of the bandwidth and data is transferred to the receiver through SMS. Singhrova and Prakash (2008) provides a new technique in the field of mobile communication. In this technique, analysis of various security protocols in mobiles has been proposed.

The various encryption and authentication technique needed during the transmission of data from mobiles has been implemented. Rein et al. (2006) implemented compression of text data in mobiles. It provides a low complexity based arithmetic coding compression of the text transferred through Mobiles. Arif et al. (2009) implemented an enhanced static data compression on short messages. It explains a new technique of data compression on Bengali short messages for the small devices using the concept of masking and dictionary. The technique implemented here is an efficient technique, especially for the short Bengali messages. Putro et al. (2007) proposed a new technique of data compression which is based on arithmetic encoding. It explains the limitations of the message length are presented and a solution of how we can send more than 160 characters in the message.

From the existing literature, it is found that no authors have proposed an integrated approach for encoded compressed cryptosystem, thus an integrated approach termed as Lucas Encoding based Compressed CRyptoSystem (LECCRS) is proposed in this paper wherein for encoding a new Lucas coding is proposed, for compression the lossless variable-length Huffman coding is considered and in performing encryption, RSA encryption with varying block size is proposed in this paper.

3 Mathematical background

This section presents an overview of Lucas series and Lucas encoding which are useful in our work.

3.1 Lucas and Fibonacci sequence

The Lucas numbers are similar to those of Fibonacci numbers and the Lucas numbers often occur in various formulae for the Fibonacci Numbers. With Fibonacci, the series starts with 0 and 1 whereas with Lucas the series starts with 2 and 1 The Lucas sequence (https://en.wikipedia.org/wiki/LucasSequence) is a set of number that starts with 2, followed by 1, and each number is equal to the sum of the preceding two numbers. In general, Lucas sequence is generated using

\[ L_n = L_{n-1} + L_{n-2} \quad \text{if} \quad n \geq 1 \quad \text{with} \quad L_0 = 2 \quad \text{and} \quad L_1 = 1 \]  \hspace{1cm} (1)

and the Fibonacci sequence is generated using

\[ F_n = F_{n-1} + F_{n-2} \quad \text{if} \quad n \geq 1 \quad \text{with} \quad F_0 = 0 \quad \text{and} \quad F_1 = 1 \]  \hspace{1cm} (2)
A part of Lucas number series is
\[ L = 2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, \ldots \]
and a part of Fibonacci number series is
\[ F = 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots \]
There are a number of relationships between the two series. The two important relations are
\[ L(n) = F(n - 1) + F(n + 1) \tag{3} \]
where \( L(n) \) is the \( n \)th Lucas number and \( F(n - 1) \) and \( F(n + 1) \) are the \((n - 1)\)th and \((n + 1)\)th Fibonacci numbers, respectively. Adding alternative Lucas numbers produces multiples of the Fibonacci series, where the multiple is 5.
\[ 5F(n) = L(n - 1) + L(n + 1) \tag{4} \]

**Definition 1** (Lucas binary encoding and computation of its value): Let \( L(n) = a_0a_1a_2 \ldots a_p \) be the Lucas binary encoding of a positive integer \( n \). The value of the Lucas binary encoding, denoted \( V(L(n)) \), is defined as follows:
\[ V(L(n)) = \sum_{i=0}^{p} a_i L_i (ai \in \{0,1\}, 0 \leq i \leq p) \tag{5} \]
\[ F(n) = \frac{\Phi^n - (-\varphi)^n}{\Phi - (-\varphi)} = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n \sqrt{5}} \text{ where } n \geq 1 \tag{6} \]
where \( \varphi \) is the well-known golden ratio, \( \varphi \approx 1.6180 \).

The relation in equation (6) is called Binet’s formula. The Lucas numbers can be written in the general form as:
\[ L(n) = \frac{\Phi^n + (-\varphi)^n}{\Phi + (-\varphi)} = \frac{(1+\sqrt{5})^n + (1-\sqrt{5})^n}{2^n \sqrt{5}} \text{ where } n \geq 1 \tag{7} \]
The Lucas and Fibonacci sequence is calculated using equations (6) and (7) and they are listed in Table 1. Since \( \Phi - \varphi = 1 \) and \( \Phi + \varphi = \sqrt{5} \), which shows there is a relationship between Lucas numbers and Fibonacci numbers formula.

**4 Proposed methodology**

To enhance the security, the original message which is to be encrypted is not directly taken in the proposed methodology instead it is encoded with Lucas coding. The Lucas form of the coded message is called as intermediate message (IM1). As the original form of the message is initially altered the eavesdropper may not know the original
message which is termed as the first level of security. The encoded message is then compressed with Huffman algorithm which also alters the encoded message called the second level of security. As the compression is used, it reduces the size of the encoded message which results in increasing the speed during transmission as well as encryption and decryption processes. It is noted that the compressed message is not directly used for encryption instead, the intermediate message2(IM2) for each block is computed from the compressed coding by generating the block size dynamically based on the two primes \( p \) and \( q \) which play a vital role in determining the key as well as block size in any public-key cryptosystems. Once IM2 is computed, it is then used for encryption using any one of the public key algorithms like RSA, ElGamal etc., and hence it is termed as the third level of security as opposed to conventional RSA wherein conventional RSA the block size is either 2048 or 4096 bits and the message to be encrypted is directly taken as plaintext for encryption. The proposed methodology is shown in Figure 1.

The details of each step in proposed methodology are explained in the next subsections.

### Table 1 Lucas and Fibonacci series

<table>
<thead>
<tr>
<th>( N )</th>
<th>( \Phi^n )</th>
<th>( (-\phi)^n )</th>
<th>( Ln = \Phi^n + (-\phi)^n )</th>
<th>( Fn = \Phi^n - (-\phi)^n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0000</td>
<td>1.0000</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1.6180</td>
<td>-0.6180</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2.6180</td>
<td>0.31819</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4.2360</td>
<td>-0.2360</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6.854</td>
<td>0.1458</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>11.091</td>
<td>-0.0901</td>
<td>11</td>
<td>5</td>
</tr>
</tbody>
</table>

**Figure 1** Proposed LECCRS scheme: (a) encryption process and (b) decryption process

#### 4.1 Lucas coding

To encode any integer using an acceptable sequence, the integer \( n \) is represented as a sum of elements from a particular sequence. For that, select the number from the sequence which is immediately less than the number and subtract from the number for which encoding is performed which leaves smaller value. The process is repeated until the difference is 0. For Lucas encoding, choosing the largest Lucas number equal to or less than \( n \) (where \( n \) is an integer for which Lucas coding is to be generated) in which the
number is subtracted from $n$ and keeping track of the remainder. If the number subtracted is the $i$th Lucas number $L(i)$, put a 1 in place $i - 2$ in the code word. Repeat the previous steps by substituting the remainder for $n$, until the remainder is 0. An additional one must be added after the rightmost digit in the code word. The steps involved in Lucas Encoding are shown in Algorithm 1.

**Algorithm 1: Lucas Coding**

BLN Lucas_Coding(N)
// N is an integer for which Lucas coding to be found
// Ln is Lucas number, n=0,1,2,...
// BLN is Binary Lucas coding for N ; LBLN is length of BLN
// RCW is rightmost code word; || is concatenation Input N

Output BLN

1. read N
2. BLN ← ϕ
3. find $L(i) ← L_n$ where $i$ is the position of nearest value of $n$ and $L_n$ is i, i-1, i-2...
4. while (N=0) do begin
   If $L(i) ≤ n$ then $L(i) ← 1$ else $L(i) ← 0$
   end if
   BLN ← BLN || $L(i)$
   Find the remainder $N ← N - L_n$ i ← i - 1
   end while N
5. while (i ≠ 0) do begin
   $L(i) ← 0$
   BLN ← BLN || $L(i)$ end while
6. RCW ← 1
7. BLN ← BLN || RCW

return BL_N

**4.1.1 Lucas coding and Fibonacci coding – example**

Table 2 shows the encoded form of the integer 65, 90 using Lucas coding as $65 = 18 + 47$ and $90 = 3 + 11 + 76$. The additional 1 is always appended. Similarly, encoded form of the integer using Fibonacci coding as $65 = 2 + 8 + 55$ and $90 = 1 + 89$.

**5 Compression with Huffman coding**

Once the encoding is performed for an integer based on Lucas or Fibonacci coding, to reduce the number of bits in Lucas coding, compression is performed. For compression, Huffman algorithm is used in this paper which takes the decimal value of Lucas binary coding of each character as input. Let DBLmi represents decimal value of Lucas binary coding for the $i$th character $mi$ in a message $M$ (where $mi \in M$) to be encrypted called IM1. The Huffman algorithm (Huffman, 1952) is a greedy approach in which the
algorithm chooses the best available option at each step. This is sufficient for finding the best encoding. The basic idea behind the algorithm is to build the tree bottom-up. A leaf node for each symbol should be created and add it to the priority queue. Two nodes of highest priority from the queue must be removed and creating a new internal node with these two nodes as children. It is shown in Algorithm 2. A new node to be added to the queue and the remaining node is the root node, then the tree is complete. The compression ratio and space saving is calculated as

\[
\text{Compression ratio} = \frac{\text{Uncompressed file size}}{\text{Compressed file size}} \quad (8)
\]

\[
\text{Space saving} = 1 - \frac{\text{Compressed file size}}{\text{Uncompressed file size}} \quad (9)
\]

**Algorithm 2: Huffman’s Compression**

HBL.m Huff_Code (DBL.m )

// HBL.m is Compressed Huffman code for BL.m where m =

mi, i=1,2, ..., k(m)

Input BL.N(m)

Output HBL.m

1. HBL.m ← ∅
2. for each DBL.mi, mi∈m, i=1,2,..,n, with probability p(DBL.mi) and let S=the set of terminal nodes,
   a. create a terminal node
3. select nodes DBL.mi and DBL.mj with i≠j in S with the two smallest probabilities.
4. replace DBL.mi and DBL.mj in S by a node with probability p(LUCmi) + p(DBL.mj).
   a. Also, create a node in the tree which is the parent of DBL.mi and DBL.mj
5. Assign 0 to left child of parent and 1 to the right child
6. Repeat steps (2)-(3) until |S|=1 or root node
7. for each terminal nodes form the prefix_code by
   concatenating the 0 or 1 starting from root HBL.m ← HBL.mi + prefix_code
8. return HBL.m

6 Modified RSA encryption and decryption

After obtaining Huffman code for the message, it is encrypted using RSA. The RSA cryptosystem is a public-key cryptosystem which includes both encryption and digital signatures. Ronald Rivest, Adi Shamir, and Leonard Adleman developed the RSA system in 1977 (Stallings, 2005). In conventional RSA encryption, the block size is normally considered as 1024, 2048 etc. Some modifications are performed in the conventional RSA in this paper called modified RSA. In the modified RSA, the block size (b) is determined dynamically on the basis of n where \( n = p \times q \) which plays a vital role in determining the modulus and key value. Further, the numeral for the plaintext characters is taken from either compressed Lucas or Fibonacci coding instead of ASCII value. The proposed dynamically generated block size is shown in Algorithm 3. After determining the block size (b), it is used for encryption and decryption in RSA. The modified RSA is shown in Algorithm 3.
Table 2  Lucas and Fibonacci coding for the integers 65 and 90

<table>
<thead>
<tr>
<th>n</th>
<th>Lucas coding</th>
<th>Fibonacci coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>90</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
</tbody>
</table>
Algorithm 3: Modified RSA Encryption and Decryption
In this algorithm Alice is the receiver and Bob is the sender

//Key generation and determine the block size b
1. generate two large distinct primes p, and q, most probably both are of same size
2. compute $n = pq$, $\phi(n) = (p-1)(q-1)$
3. convert n into binary $nb \leftarrow (n)2$
4. find $b \leftarrow \text{len}(nb)$
5. select a random integer $c$, $1 < c < \phi(n)$ such that $\gcd(c, \phi(n)) = 1$.
6. use the extended Euclidean Algorithm to Sho compute the unique Integer $d \leq c < \phi(n)$
   such that $ed \equiv 1 \pmod{\phi(n)}$
7. A’s public-key is $(n, e)$; A’s private-key is $(n, d)$.

//RSA Encryption and Decryption Based on Compressed Lucas coding
SUMMARY B encrypts a message IM2 for A, in which A decrypts.
1. Encryption. B should do the following:
   a. obtain A’s authenticate public-key $(n, e)$.
   b. $c \leftarrow c$
   c. repeat
      i. Read the first $b$ bits from HBLm
      ii. Convert the bits into binary. Let it be IM2 Compute IC \leftarrow IM2 mod n.
      iii. $c \leftarrow c \| IC$
      iv. read the next $b$ bits from HBLm until all bits are read in HBLm
   d. send the ciphertext $c$ to A.
2. Decryption. To recover the text IM2 from $c$, A do the following:
   Use the private-key $d$ to recover $IM2 = c^d \mod n$. Once the IM2 is obtained, the whole
   process is reversed, so that the receiver can get the original plaintext $m$. Similar methodology can
   also be performed for encrypting the message with Fibonacci coding.

6.1 Proposed methodology – example

In the proposed methodology, the message $M$ is taken as ‘kannanbaba’. The corresponding Lucas coding after encoding of message ‘Kannanbaba’ is 1000000101
10001001 101000101 101000101 10001001 101000101 1001001 10001001 1001001
10001001. The Intermediate Message1 (IM1) for Lucas coding of each character is
computed which is then used for further processing. It is shown in Table 3.

The IM1 is further compressed after obtaining Lucas coding which is shown in
Figure 2. From Figure 2, the compressed code of ‘kannanbaba’ is 111 0 10 10 0 10 110 0
110 0. Its decimal equivalent is 479948 which is called as intermediate plaintext
message2 (IM2). It is then encrypted using RSA. To encrypt IM2, let $p = 50053$, $q = 50069$. Then, $n = 50053(50069) = 2506103657$, $\phi(n) = 50052(50068) = 2506603536$. Let $e = 56989$. Using Extended Euclidean algorithm $d$ is computed as $d = 2472671653$. Since $25061036572$ is 1001 0101 0110 0000 0001 1011 0110 1001, the number of bits
in $25061036572$ is 32. Thus, the key size is 32 bits and block size is taken in such
a way that it must be less than 32. Here, the block size is 19, because, after
applying Huffman coding the code is reduced to 19 bits. To encrypt, \( C = IM2^e \mod n = 479948 \mod 831878136 \). To decrypt, \( IM2 = C^d \mod n = 831878136 \mod 479948 \). The entire process is reversed, so that original plaintext message M is obtained. It is shown in Table 4. A similar methodology is adopted for Fibonacci coding, and encryption and decryption are performed before and after compression too.

### Table 3 Encoding of plaintext characters using Lucas sequence

<table>
<thead>
<tr>
<th>Lucas series</th>
<th>Lucas coding</th>
<th>IM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>k</td>
<td>107</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>97</td>
<td>0</td>
</tr>
<tr>
<td>n</td>
<td>110</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>97</td>
<td>0</td>
</tr>
<tr>
<td>n</td>
<td>110</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>97</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>98</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>97</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>98</td>
<td>0</td>
</tr>
</tbody>
</table>


### Table 4 Decoding of ciphertext using Lucas coding

<table>
<thead>
<tr>
<th>HDC</th>
<th>NHDC</th>
<th>2</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>7</th>
<th>11</th>
<th>18</th>
<th>29</th>
<th>47</th>
<th>76</th>
<th>Lucas decoding</th>
<th>ASCII</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>517</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1000000101</td>
<td>( 2 + 29 + 76 = 107 )</td>
<td>k</td>
</tr>
<tr>
<td>0</td>
<td>1337</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1000000101</td>
<td>( 3 + 18 + 76 = 97 )</td>
<td>a</td>
</tr>
<tr>
<td>10</td>
<td>3225</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>101000101</td>
<td>( 1 + 4 + 29 + 76 = 110 )</td>
<td>n</td>
</tr>
<tr>
<td>0</td>
<td>1337</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>100000101</td>
<td>( 3 + 18 + 76 = 97 )</td>
<td>a</td>
</tr>
<tr>
<td>10</td>
<td>3225</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>101000101</td>
<td>( 1 + 4 + 29 + 76 = 110 )</td>
<td>n</td>
</tr>
<tr>
<td>111</td>
<td>773</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1001001</td>
<td>( 4 + 18 + 76 = 98 )</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>1337</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>100000101</td>
<td>( 3 + 18 + 76 = 97 )</td>
<td>a</td>
</tr>
<tr>
<td>111</td>
<td>773</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1001001</td>
<td>( 4 + 18 + 76 = 98 )</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>1337</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>100000101</td>
<td>( 3 + 18 + 76 = 97 )</td>
<td>a</td>
</tr>
</tbody>
</table>

HDC – Huffman Decoding; NHDC – Numeral for HDC.

### 7 Experimental result

The proposed methodology is implemented in VC++ using Pentium processor with various file sizes. The security level is measured using All Block Cipher (ABC) Universal Hackman tool which uses dictionary attacks. The time taken for encryption
and decryption for the three methods namely, RSA, RSA-FIB, RSA-LUC with varying file sizes namely 1MB, 2MB, 4MB, 8MB and 16MB is tabulated in Table 5 and its corresponding graphs are shown in Figure 3.

From Table 5, it is observed that the time taken for encryption and decryption process for conventional RSA is less than that of RSA with Fibonacci (RSA-FIB) and RSA with Lucas (RSA-LUC). This is because in RSA-LUC and RSA-FIB, several processes are involved before encryption namely, forming IM1 using coding, the IM1 is further compressed and the compressed IM1 is then converted into decimal etc. Since each process requires a considerable amount of time, as a result it takes more time than the conventional RSA wherein conventional RSA, just the ASCII value of each character alone is considered, which is then used for encryption. The decryption time for RSA-LUC and RSA-FIB is also increasing due to the reverse process of said reasons. The encryption and decryption time ratio for consecutive file sizes are shown in Table 6.

![Figure 2](image-url)  
**Figure 2** Compression of ‘kannanbaba’ using Huffman algorithm

![Table 5](table-url)  
**Table 5** Encryption and decryption time before compression

Using equations (8) and (9), the compression ratio and space storage for 1MB file size are calculated as compression ratio for RSA-FIB(1MB) $= \frac{1024}{758} = 1.3514$, space storage for RSA-FIB(1MB) $= 1 - \frac{758}{1024} = 0.260$, compression ratio for RSA-LUC(1MB) $= \frac{1024}{717} = 1.4286$, space storage for RSA-LUC(1MB) =
1 – (717/1024) = 0.30. The compression ratio for other file sizes with the proposed methods RSA-FIB and RSA-LUC are calculated in this manner and they are shown in Table 7 and its graphical representation is shown in Figure 4.

Figure 3 (a) Encryption and (b) decryption time (ms) before compression (see online version for colours)
Table 6  Ratio for encryption and decryption time

<table>
<thead>
<tr>
<th>Method</th>
<th>Encryption time ratio</th>
<th>Decryption time ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>File size</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1MB and 2MB</td>
<td>2MB and 4MB</td>
</tr>
<tr>
<td>RSA</td>
<td>1.95</td>
<td>2.00</td>
</tr>
<tr>
<td>RSAFIB</td>
<td>1.98</td>
<td>1.99</td>
</tr>
<tr>
<td>RSALUC</td>
<td>1.99</td>
<td>1.97</td>
</tr>
<tr>
<td></td>
<td>4MB and 8MB</td>
<td>16MB</td>
</tr>
<tr>
<td>RSA</td>
<td>1.98</td>
<td>2.00</td>
</tr>
<tr>
<td>RSAFIB</td>
<td>1.97</td>
<td>1.99</td>
</tr>
<tr>
<td>RSALUC</td>
<td>1.99</td>
<td>1.99</td>
</tr>
<tr>
<td></td>
<td>8MB and 16MB</td>
<td></td>
</tr>
<tr>
<td>RSA</td>
<td>1.99</td>
<td>2.00</td>
</tr>
<tr>
<td>RSAFIB</td>
<td>1.99</td>
<td>1.99</td>
</tr>
<tr>
<td>RSALUC</td>
<td>1.99</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Table 7  Compression ratio after Lucas encoding

<table>
<thead>
<tr>
<th>Method</th>
<th>Compression ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>File size (MB)</td>
</tr>
<tr>
<td>RSA</td>
<td>1.000</td>
</tr>
<tr>
<td>RSAFIB</td>
<td>1.3514</td>
</tr>
<tr>
<td>RSALUC</td>
<td>1.4286</td>
</tr>
</tbody>
</table>

It is evident from Table 7 that in the conventional RSA, whatever be the file size the compression ratio is always 1 which indicates that the size of the plaintext and the ciphertext remains same. But in the case of proposed RSA-FIB and proposed RSA-LUC the compression ratio is increasing when the file size is also increasing which means that for larger the size of file, the size of the coding is also increasing but the increasing ratio is not that much of increasing the file size.

After compressing the IM1 using Huffman Algorithm, the encoded message will be further reduced into a smaller size so that a smaller intermediate plaintext is created. This compressed intermediate plaintext is taken into RSA algorithm and encryption and decryption time is also recorded in Table 8. Figure 5 shows its corresponding graphical representation. It is observed from Table 8 that the encryption and decryption time taken by RSA-LUC is somewhat less when it is compared with RSA-FIB. This is because the number of bits required in coding for any integer is more using Fibonacci series than the Lucas series as the Fibonacci series starts with 0 and 1 but the Lucas series starts with 2 and 1.

Table 8  Encryption and decryption time after Huffman compression

<table>
<thead>
<tr>
<th>Method</th>
<th>Encryption time after compression (ms)</th>
<th>Decryption time after compression (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>File size</td>
<td></td>
</tr>
<tr>
<td>RSA</td>
<td>1MB 2MB 4MB 8MB 16MB</td>
<td>1MB 2MB 4MB 8MB 16MB</td>
</tr>
<tr>
<td>RSAFIB</td>
<td>3158 6148 12,313 24,467 48,939</td>
<td>3071 6151 12,235 24,505 48,923</td>
</tr>
<tr>
<td>RSALUC</td>
<td>2779 5490 10,696 20,618 41,993</td>
<td>2773 5483 10,680 20,537 41,986</td>
</tr>
</tbody>
</table>

It is evident from Table 7 that in the conventional RSA, whatever be the file size the compression ratio is always 1 which indicates that the size of the plaintext and the ciphertext remains same. But in the case of proposed RSA-FIB and proposed RSA-LUC the compression ratio is increasing when the file size is also increasing which means that for larger the size of file, the size of the coding is also increasing but the increasing ratio is not that much of increasing the file size.

After compressing the IM1 using Huffman Algorithm, the encoded message will be further reduced into a smaller size so that a smaller intermediate plaintext is created. This compressed intermediate plaintext is taken into RSA algorithm and encryption and decryption time is also recorded in Table 8. Figure 5 shows its corresponding graphical representation. It is observed from Table 8 that the encryption and decryption time taken by RSA-LUC is somewhat less when it is compared with RSA-FIB. This is because the number of bits required in coding for any integer is more using Fibonacci series than the Lucas series as the Fibonacci series starts with 0 and 1 but the Lucas series starts with 2 and 1.

Table 8  Encryption and decryption time after Huffman compression

<table>
<thead>
<tr>
<th>Method</th>
<th>Encryption time after compression (ms)</th>
<th>Decryption time after compression (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>File size</td>
<td></td>
</tr>
<tr>
<td>RSA</td>
<td>1MB 2MB 4MB 8MB 16MB</td>
<td>1MB 2MB 4MB 8MB 16MB</td>
</tr>
<tr>
<td>RSAFIB</td>
<td>3158 6148 12,313 24,467 48,939</td>
<td>3071 6151 12,235 24,505 48,923</td>
</tr>
<tr>
<td>RSALUC</td>
<td>2779 5490 10,696 20,618 41,993</td>
<td>2773 5483 10,680 20,537 41,986</td>
</tr>
</tbody>
</table>
Figure 4  Compression ratio for consecutive file size (see online version for colours)

Figure 5  (a) Encryption and (b) decryption time after compression (see online version for colours)
If the successive numbers generated by both series are considered, the magnitude of the number is increasing when the sequence is generated by Lucas series rather than Fibonacci series. For example, when \( n = 5 \), the Lucas series number is 11 whereas it is 5 if Fibonacci series is used. Table 2 clearly shows that for the integer 65, the number of bits required for Lucas coding is only 10 bits whereas Fibonacci coding requires 12 bits. Thus, for large integer, the number of bits required in coding using Lucas series is always less than that of Fibonacci series. Table 9 shows the ratio between two consecutive file sizes for both encryption and decryption process of various methods after compression. It is evident from Table 9 that the ratio is almost two for all the methods as in the case of before compression. The reason for the time taken for both encryption and decryption with RSA-FIB and RSA-LUC is higher than RSA, though the same operations are repeated for all file sizes and hence the ratio remains same.

Table 9  Ratio of encryption and decryption time after Huffman compression

<table>
<thead>
<tr>
<th>Method</th>
<th>1MB and 2MB</th>
<th>2MB and 4MB</th>
<th>4MB and 8MB</th>
<th>8MB and 16MB</th>
<th>1MB and 2MB</th>
<th>2MB and 4MB</th>
<th>4MB and 8MB</th>
<th>8MB and 16MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>1.95</td>
<td>2.00</td>
<td>1.99</td>
<td>2.00</td>
<td>2.00</td>
<td>1.99</td>
<td>2.00</td>
<td>2.00</td>
</tr>
<tr>
<td>RSAFIB</td>
<td>1.98</td>
<td>1.95</td>
<td>1.93</td>
<td>2.04</td>
<td>1.98</td>
<td>1.95</td>
<td>1.92</td>
<td>2.04</td>
</tr>
<tr>
<td>RSALUC</td>
<td>1.95</td>
<td>1.95</td>
<td>2.01</td>
<td>1.98</td>
<td>1.93</td>
<td>1.97</td>
<td>2.01</td>
<td>1.98</td>
</tr>
</tbody>
</table>

To determine the efficiency of any cryptographic algorithm, security plays a vital role. To measure the security level, ABC Hackman tool is used in this paper and the security level is analysed individually for all the three methods with various plaintext size of 1MB, 2MB, 4MB, 8MB and 16MB. It is shown in Table 10 and its graphical representation is shown in Figure 6. The average security level is calculated for conventional RSA, proposed RSA-FIB and proposed RSA-LUC as 82%, 93.2% and 94.4%, respectively. It is evident from Table 10 that the proposed RSA-FIB and RSA-LUC produced more security than the existing RSA. Among the proposed methods, RSA-LUC is outperforming than RSA-FIB. This is because for encrypting the plaintext character, Fibonacci and Lucas series are used rather than ASCII value. Moreover, the generation of Fibonacci series starts with 0 and 1, but the Lucas series starts with 2 and 1. When encoding the plaintext, the original plaintext will be changed into some other coded form. This leads to the first level of security produced by RSA-FIB and RSA-LUC. After encoding, the binary coded form of Fibonacci and Lucas series will be converted into numerals. Only those numerals will be considered for compression using Huffman algorithm to reduce the size of plaintext further which leads to the second level of security.

In conventional RSA, though the prime number is taken very large but the block size and key size are always taken as fixed (static), i.e., 1024 or 2048 or 4098 bits. As they are fixed, the eavesdropper may sometimes hack the plaintext. In the proposed RSA-FIB and RSA-LUC, the block size is dynamically determined on the basis of length of the binary value \( n \), where \( n = p \times q \), \( p \) and \( q \) are primes which are taken as input. For performing encryption using RSA-FIB and RSA-LUC, the numeral of the plaintext is calculated
as the decimal value of \( l \) bits from the compressed code which leads to the third level of security. Thus the security level for RSA-FIB and RSA-LUC is higher than conventional RSA.

### Table 10 Security level

<table>
<thead>
<tr>
<th>Method</th>
<th>Security level (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1MB</td>
</tr>
<tr>
<td>RSA</td>
<td>88</td>
</tr>
<tr>
<td>RSA-FIB</td>
<td>96</td>
</tr>
<tr>
<td>RSA-LUC</td>
<td>97</td>
</tr>
</tbody>
</table>

### Figure 6 Security level produced by RSA, proposed RSA-FIB and proposed RSA-LUC (see online version for colours)

### 8 Conclusion

An alternate encoding scheme Lucas encoding is thought of similar to Fibonacci coding and it is encapsulated with plaintext. The Lucas coding acts as a wrapper so that the original form of plaintext is altered which may produce the first level of security. The coded plaintext is further compressed so that the size of the coded plaintext is reduced which leads to the second level of security. It also increases the speed and occupies a low bandwidth of the channel during transmission time when it is compared with uncompressed coded. As the block size is determined dynamically, the eavesdropper may not identify the correct block size which results in another level of security. Further, as the plaintext is not directly encrypted using any cryptographic algorithms (RSA is taken for test case), but the plaintext is altered in several forms due to several levels involved in the proposed methodology. When the message is encrypted using RSA with the proposed
methodology, the security level is increasing. The experimental result is also further strengthening the increase of security level using the proposed method.

References


Websites

Huffman Coding, Available at http://en.wikipedia.org/wiki/Data_compression
Lucas Sequence, Available at https://en.wikipedia.org/wiki/LucasSequence
RSA, Available at https://en.wikipedia.org/wiki/RSA