A New Encryption Algorithm Based on Fibonacci Polynomials and Matrices

Orhan Dişkaya¹, Erdinç Avaroğlu², Hamza Menken¹, Ahmet Emsal²

¹ Department of Mathematics, Faculty of Science, Mersin University, Mersin 33110, Turkey
² Computer Engineering Department, Faculty of Engineering, Mersin University, Mersin 33110, Turkey

Corresponding Author Email: eavaroglu@mersin.edu.tr

https://doi.org/10.18280/ts.390501

Received: 29 June 2022
Accepted: 2 October 2022

Abstract
Confusion and diffusion features are two fundamental needs of encoded text or images. These features have been used in various encryption algorithms such as Advanced Encryption Standard (AES) and Data Encryption Standard (DES). The AES adopts the S-box table formed with irreducible polynomials, while the DES employs the Feistel and S-box structures. This study proposes a new encryption algorithm based on Fibonacci polynomials and matrices, which meets the fundamental needs of image encryption and provides an alternative to other encryption algorithms. The success of the proposed method was tested on three different images, as evidenced by the histogram analysis results of the sample images, together with the number of changing pixel rate (NPCR) and the unified averaged changed intensity (UACI). In addition, the root mean squared error (RMSE) suggests that the decoded images are consistent with the original images. It can therefore be summarized that the proposed encryption algorithm is suitable for image encryption.

1. INTRODUCTION

Since ancient times, all of humanity has shared the commitment to protecting their right to privacy. For governments, organizations, and people in general, privacy protection is crucial. For all these parties, the common issue is to protect their privacy, and keep their conversations and data confidential, so that no attacker can obtain or read this information. Cryptography was used to encrypt the sent messages with an algorithm using a key that was only known to the sender and receiver and to guarantee their confidentiality.

Many encryption methods have been put forth through the years to safeguard privacy. The classical encryption techniques include Caesar, Hill, Vigenere, etc. But these techniques cannot meet the security needs in the fast-evolving modern society. This gives birth to modern encryption algorithms, such as Data Encryption Standard (DES), Advanced Encryption Standard (AES), Rivest–Shamir–Adleman (RSA) algorithm, and digital signature algorithm (DSA). Capable of applying to various areas and meeting security needs, modern encryption algorithms can be divided into symmetric encryption and asymmetric encryption.

DES, a secret-key symmetric (single-key) encryption algorithm, relies on block encryption to encrypt huge amounts of data. Despite being the most widely used encryption algorithm ever, it has been defeated by contemporary computers. The DES approach is intricate and incorporates both confusion and diffusion processes. It utilizes S-box and Feistel architecture.

Another approach for data encryption is AES, which was developed after DES was cracked. The keys used for both encryption and decryption in this symmetric-key encryption process are connected to one another. Four steps make up AES, including substituting bits, shifting rows, mixing columns, and adding keys. AES often employs the same confusion and diffusion processes as DES. The difference is that AES operates on irreducible polynomials. Polynomials play a significant role in mathematics and are widely employed, particularly in the field of cryptology. One of these is the Fibonacci polynomial, which has been employed in a number of research that have been published in the field of cryptology [1-8]. In this paper, the literature review summarizes these investigations.

Fibonacci polynomials and matrices are used in several of the above studies. However, in the encryption phase, con and diff operations are not applied. This paper develops a new encryption technique, providing an alternative to the encryption algorithms that offer con and diff operations in the literature. Fibonacci polynomials and matrices are adopted in the suggested strategy. The next term in a series of numbers known as the Fibonacci numbers is determined by the sum of the two initial conditions. Three separate images were encrypted and decrypted using the suggested technique. The success of the proposed technique was demonstrated by the histogram analysis of the operations conducted. The value obtained from the mean squared error (MSE) was zero, indicating the consistency between the original and decoded images. The number of pixel change rate (NPCR) and the unified averaged changed intensity (UACI) were also provided to demonstrate the effectiveness of our approach.

This study makes the following contributions:
- Uniform distribution is achieved through encryption.
- Confusion and diffusion processes are provided by Fibonacci polynomial matrices.
- The Fibonacci polynomial matrices in confusion and diffusion are calculated differently in each cycle, while the matrix of the column shuffling step in the AES encryption algorithm is the same in every cycle.
- The proposed approach can be applied easily to both images and texts.
- The statistical analysis proves the good performance of our approach.

The remaining parts are organized as follows: Section 2 reviews the literature on Fibonacci polynomials and matrices; Section 3 explains the proposed method; Section 4 applies our approach on 3 different images, and presents the results; Section 4 discusses the results, and summarizes the strengths and defects of our approach.

2. LITERATURE REVIEW

According to a novel method put forth by Mukherjee and Samanta [1], a message can be both encoded and hidden in an image by using the Fibonacci series to encrypt it.

In their study, Uçar et al. [2] developed two new encoding/decoding methods that make use of Fibonacci Q-matrices and R-matrices. The models rely on blocked message matrices and key-based encryption for each message matrices. These new algorithms have great accuracy capabilities in addition to increasing information security.

Zou et al. [3] introduced a brand-new technique for digital image mixing based on the Fibonacci numbers. There have been concerns raised about the mixing transform's homogeneity and periodicity. The benefits of mixing transform are as follows: Encoding and decoding are both very easy to use in real-time scenarios. The mixing effect is excellent, as the image data is randomly redistributed across the image. The technique is resistant to common image attacks like noise, compression, and data packet loss.

Khadija et al. [4] dealt with text encoding in such a way that the receiver can only find out the original message without any data loss or shift or data leakage. Some possible approaches were presented to solve their identified problem, using Fibonacci series. The data was encrypted by combining the original data with Fibonacci numbers to get a cipher text, which is incomprehensible to any intruder. Only the receiver knows the logic of the way doing this.


3. MATERIALS AND METHODS

In mathematics, Fibonacci polynomials are considered a generalization of Fibonacci numbers. In 1883, Fibonacci polynomials \( f_n(x) \) were examined, for the first time, by Belgian Mathematician Eugene C. Catalan. This work was followed by German mathematician Ernst Jacobsthal (1882-1965). In 1973, Hoggatt and Bicknell [9] defined Fibonacci polynomials and matrices.

The Fibonacci polynomials sequence can be defined by the recurrence relationship:

\[
 f_{n+2}(x) = xf_{n+1}(x) + f_n(x), \quad n \geq 0
\]

with the initial conditions being \( f_0(x)=0 \) and \( f_1(x)=1 \).

The matrix of the Fibonacci polynomials \( Q(x) \) can be defined as:

\[
 Q(x) = \begin{pmatrix} x & 1 \\ 1 & 0 \end{pmatrix}
\]

and the general Fibonacci polynomials matrix \( Q'(x) \) can be expressed as:

\[
 Q'(x) = \begin{pmatrix} f_{n+1}(x) & f_n(x) \\ f_n(x) & f_{n-1}(x) \end{pmatrix}
\]

Much information about Fibonacci numbers can be found in the book Fibonacci and Lucas Numbers with Applications authored by Koshy [10]. According to this book, the Cassini's identity, an important identity of the Fibonacci numbers, can be explained as follows:

\[
 f_{n+1}(x) f_{n-1}(x) - f_n^2(x) = (-1)^n
\]

Since the determinant of the Fibonacci polynomial matrix equals the Cassini’s identity (2) and is non-zero, the Fibonacci polynomial matrix \( Q'(x) \) is invertible for all \( n \) values. This is an important feature for cryptology. The inverse of the Fibonacci polynomial matrix \( Q'(x) \) can be given as:

\[
 Q'^{-1}(x) = \begin{pmatrix} f_{n+1}(x) & f_n(x) \\ f_n(x) & f_{n-1}(x) \end{pmatrix}
\]

4. RESULT AND DISCUSSION

4.1 Our approach

The encryption algorithm involves three rounds, the initial round, the main round, and the final round. In the initial round, XOR operation is performed between the state matrix (clear text) and the key matrix (private key).

In the main round, the mixing process is applied in general. The confusion and diffusion in this round are created in the following steps:

- Traverse conversion
- Fibonacci s-box conversion
- Change row and column matrix
- Multiply matrix (multiplication by Fibonacci polynomial matrix)

The results obtained through con and diff operations and the key generated from the round key are subjected to XOR operation. The above operations in the main round are repeated twelve times.

In the final round, the matrix from the main part is once again processed through Fibonacci s-box conversion and the last key, before terminating the encryption process.

The process of our approach is detailed as follows:

Firstly, cross transform is applied on the add matrix, and the matrix is mixed using Fibonacci AES s-box [11].

Step 1 (Initial round): The 128-bit image (state matrix) and the polynomials in each cell of the key matrices are added as follows:

Suppose \( X_{ij} \), \( Y_{ij} \), and \( Z_{ij} \) are order 7 polynomials, where \( 1 \leq i \), \( j \leq 4 \),
\[
Z_y = X_y + Y_y = \left(a^{(0)}_x + b^{(0)}_y \right)x^7 + L + a^{(0)}_x + b^{(0)}_y,
\]
\[
a^{(0)}_x, b^{(0)}_y \in \{0, 1\}, 0 \leq k \leq 7.
\]

is obtained.

**Step 2 (Main Round):** According to mod 2, the order 7 polynomials are divided into two order 3 blocks, forming the 4th degree A and B polynomial elements. That is:

\[
Z_0 = c_x^{(1)}x^3 + c_x^{(2)}x^2 + c_x^{(0)}x + L + c_y^{(0)} , c_x^{(0)} \in \{0, 1\}, 0 \leq k \leq 7.
\]

In the case that \(A_{ij}, B_{ij}\) are order 3 polynomials;

\[
A_y = c_1^{(0)}x^3 + c_1^{(2)}x^2 + c_1^{(1)}x + c_1^{(0)}, c_1^{(0)} \in \{0, 1\}, 4 \leq k \leq 7,
\]
\[
B_y = c_1^{(0)}x^3 + c_1^{(2)}x^2 + c_1^{(1)}x + c_1^{(0)}, c_1^{(0)} \in \{0, 1\}, 0 \leq k \leq 3.
\]

is obtained.

Then, the values marked in the transpose conversion matrix are multiplied by the first eight Fibonacci elements obtained using mod 2 and the irreducible polynomial \(x^4+x+1\).

The first eight Fibonacci elements are:

\[
f_1(x) = 1, f_2(x) = x, f_3(x) = x^2 + 1, f_4(x) = x^2 = x, f_5(x) = x^2, f_6(x) = x^2 + x, f_7(x) = x^2.
\]

The multiplication goes as follows:

\[
C_y = A_y \times f_{i,j} \pmod{2} \quad \text{or} \quad C_y = B_y \times f_{i,j} \pmod{2}
\]

If polynomials \(A_{ij}, B_{ij}\) and \(C_{ij}\) of the obtained matrix are converted using the hexadecimal base around the Fibonacci s-box table, it is possible to obtain the Fibonacci s-box conversion matrix. The rows of this matrix are shifted by the first four terms of the Fibonacci numbers (i.e., 1, 1, 2 and 3), respectively. Then similar operations are applied to the columns, respectively:

- The first row of the matrix is shifted 1 bit left based on the value of the first term of the Fibonacci s-box table.
- The second row of the matrix is shifted 1 bit to the left based on the value of the second term of the Fibonacci s-box table.
- The third row of the matrix is shifted 2 bits to the left based on the value of the third term of the Fibonacci s-box table.
- The fourth row of the matrix is shifted 3 bits to the left based on the value of the fourth term of the Fibonacci s-box table.

The row changes result in a change row matrix. Then, the column change is applied to the change row matrix:

- The first column of the matrix is shifted up by 1 bit based on the value of the first term of the Fibonacci s-box table.
- The second column of the matrix is shifted up by 1 bit based on the value of the second term of the Fibonacci s-box table.
- The third column of the matrix is shifted up by 2 bits based on the value of the third term of the Fibonacci s-box table.
- The fourth column of the matrix is shifted up by 3 bits based on the value of the fourth term of the Fibonacci s-box table.

Thereby, a change column matrix is obtained, and then divided into blocks to be multiplied by the 2x2 Fibonacci polynomial matrix (1). The generated new 2x2 type matrices adopt mod 2 and irreducible polynomials \(x^4+x+1\), respectively. On this basis, the multiply matrix can be obtained by:

\[
\begin{bmatrix}
K_{22} & L_{22} & f_{i,j} & f_{i,j} \\
K_{55} & L_{55} & f_{i,j} & f_{i,j} \\
K_{25} & L_{25} & f_{i,j} & f_{i,j} \\
K_{21} & L_{21} & f_{i,j} & f_{i,j} \\
K_{22} & L_{22} & f_{i,j} & f_{i,j} \\
K_{55} & L_{55} & f_{i,j} & f_{i,j} \\
K_{25} & L_{25} & f_{i,j} & f_{i,j} \\
K_{21} & L_{21} & f_{i,j} & f_{i,j}
\end{bmatrix}
= \begin{bmatrix}
D_{11} E_{11} \\
D_{21} E_{21} \\
D_{31} E_{31} \\
D_{41} E_{41}
\end{bmatrix}
\]

and \(\{n=2^i+1, i=1 \} \) where for 1\(\leq i \leq 4\), \(K_{ij}, L_{ij}, D_{ij}\), and \(E_{ij}\) are order 4 polynomials.

The \(n\) value is updated for these operations each round. The add matrix is created in the final phase of the main round by adding the multiply matrix and round key 0. The round key is added to the key created using the prior key in each loop.

**Step 3 (Final Round):** The final round takes place after the image has been encrypted in the main round. The image that needs to be encrypted will undergo multiply matrix operations in the final round, be added with round key 13, putting an end to the encryption process.

Figure 1 depicts the flow of the encryption process. Figure 2 presents the flow of the encryption algorithm.

---

**Figure 1. The encryption process**
Tables 1-5 display the nominal codes of cross transformation, s-box function, mix column and row, product of matrix and sum of matrix stages in the flowchart, respectively.

4.2 Practical applications of the proposed method and analysis results

This section discusses the practical application and test results of the proposed encryption algorithm. Specifically, our approach was applied to Lena, baboon and baby images, which
are widely used in the literature. All tests were carried out on a computer with Intel Core i5 (7300HQ) quad-core CPU, 8 GB DDR4 RAM, 1 TB HDD and Windows 10. All images were processed by MATLAB R2019b. The application effect was measured by histogram analysis, root mean square error (RMSE) and differential attack analysis.

Table 1. The cross transformation

```csharp
Function CrosTransformation (Inputs: blockMtx, Fibonacci polynomials)
{
    byte dec, a, b, c;
    Polynomial poly, polyF, polyRes;
    bool direction;
    for (byte i = 0, j; i < Faes.rowSize; i++)
    {
        for (j = 0; j < Faes.columnSize; j++)
        {
            dec = blockMtx[i, j];
            a = (byte)(dec >> 4);
            b = (byte)(dec & 15);
            direction = ((i + j) & 1) == 0;
            if (direction) //Left
            {
                poly = Polynomial.WithDec(a);
                polyF = fibonacciPolynomials[Convert.ToInt32((i + 1) * (j + 1)) & 15]);
            }
            else
            {
                poly = Polynomial.WithDec(b);
                polyF = fibonacciPolynomials[Convert.ToInt32((i + 1) * (j + 1)) & 15])
            }
            polyRes = (poly * polyF).reduction().mod2();
            c = Convert.ToByte(polyRes.evaluate());
            blockMtx[i, j] = direction
            Convert.ToByte((c << 4) + b); Convert.ToByte((a << 4) + c);
        }
    }
}
```

Table 2. SBOX function

```csharp
Function sbox (Inputs: blockMtx, sBoxTables)
{
    for (byte i = 0, j, dec; i < Faes.rowSize; i++)
    {
        for (j = 0; j < Faes.columnSize; j++)
        {
            dec = blockMtx[i, j];
            blockMtx[i, j] = sBoxTables[dec >> 4, dec & 15];
        }
    }
}
```

Table 3. Mix column and row

```csharp
Function mixColumnsAndRows (Inputs: blockMtx, method)
{
    List<byte> row = new List<byte>(new byte[Faes.columnSize]);
    List<byte> column = new List<byte>(new byte[Faes.rowSize]);
    byte r, i, b, l;
    if (method == Methoddecrypt)
    {
        for (r = 0, l = Convert.ToInt32(Faes.horizontalShift.Length); r < l; r++)
        {
            for (i = 0; i < Faes.columnSize; i++) row[i] =
            blockMtx[r, i];
            for (i = 0; i < Faes.horizontalShift[r]; i++)
            {
                b = row[0];
                row.RemoveAt(0);
                row.Add(b);
            }
            for (i = 0; i < Faes.columnSize; i++) blockMtx[r, i] = row[i];
            for (r = 0, l = Convert.ToInt32(Faes.verticalShift.Length); r < l; r++)
            {
                for (i = 0; i < Faes.rowSize; i++) column[i] =
            blockMtx[i, r];
                for (i = 0; i < Faes.verticalShift[r]; i++)
                {
                    b = column[0];
                    column.RemoveAt(0);
                    column.Add(b);
                }
                for (i = 0; i < Faes.rowSize; i++) blockMtx[i, r] = column[i];
            }
    }
    else
    {
        for (r = 0, l = Convert.ToInt32(Faes.verticalShift.Length); r < l; r++)
        {
            for (i = 0; i < Faes.rowSize; i++) column[i] =
            blockMtx[i, r];
                for (i = 0; i < Faes.verticalShift[r]; i++)
                {
                    b = column[Faes.rowSize - 1];
                    column.RemoveAt(Faes.rowSize - 1);
                    column.Insert(0, b);
                }
                for (i = 0; i < Faes.rowSize; i++) blockMtx[i, r] = column[i];
        }
    }
```
Table 4. Product of matrix

<table>
<thead>
<tr>
<th>Function productOfMatrices (Inputs: blockMtx, roundIdx, method)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>Polynomial[,] group = new Polynomial[2, 2];</td>
</tr>
<tr>
<td>Polynomial[,] fibonacciMatrix;</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>int round = roundIndex + 1,</td>
</tr>
<tr>
<td>multiplier = method == Method.decrypt ? 1 : 1;</td>
</tr>
<tr>
<td>fibonacciMatrix = new Polynomial[2, 2] {</td>
</tr>
<tr>
<td>[this.fibonacciPolynomials[round] + 1 *</td>
</tr>
<tr>
<td>multiplier], this.fibonacciPolynomials[round] },</td>
</tr>
<tr>
<td>this.fibonacciPolynomials[round - 1 * multiplier] },</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>Polynomial res;</td>
</tr>
<tr>
<td>uint d;</td>
</tr>
<tr>
<td>for ( byte k = 0, l, i, j, m, dec;</td>
</tr>
<tr>
<td>k &lt; 2;</td>
</tr>
<tr>
<td>k++</td>
</tr>
<tr>
<td>)</td>
</tr>
<tr>
<td>for (l = 0; l &lt; 4; l++)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>for (i = 0; i &lt; 2; i++)</td>
</tr>
<tr>
<td>group[i, 0] = Polynomial.WithDec(</td>
</tr>
<tr>
<td>Convert.ToByte(blockMtx[(k &lt;&lt; 1) + i, l] &gt;&gt; 4));</td>
</tr>
<tr>
<td>group[i, 1] = Polynomial.WithDec(</td>
</tr>
<tr>
<td>Convert.ToByte(blockMtx[(k &lt;&lt; 1) + i, l] &amp; 15));</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>for (i = 0; i &lt; 2; i++)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>dec = 0;</td>
</tr>
<tr>
<td>for (j = 0; j &lt; 2; j++)</td>
</tr>
<tr>
<td>res = new Polynomial();</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>for (m = 0; m &lt; 2; m++)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>fibonacciMatrix[m, j];</td>
</tr>
<tr>
<td>res = PolynomialMatrix.Sumation(group[i, m] *</td>
</tr>
<tr>
<td>fibonacciMatrix[m, j]);</td>
</tr>
<tr>
<td>d = res.reduction()</td>
</tr>
<tr>
<td>.mod2();</td>
</tr>
<tr>
<td>.evaluate();</td>
</tr>
<tr>
<td>dec += (byte)(j == 0 ? d &lt;&lt; 4 : d);</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>blockMtx[(k &lt;&lt; 1) + i, l] = dec;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Table 5. Sum of matrix

<table>
<thead>
<tr>
<th>Function sumOfMatrices (Inputs: blockMtx, key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>for (byte i = 0, j; i &lt; Faes.rowSize; i++)</td>
</tr>
<tr>
<td>for (j = 0; j &lt; Faes.columnSize; j++)</td>
</tr>
<tr>
<td>blockMtx[i, j] ^= key[i * Faes.columnSize + j];</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

private void sumOfMatrices(ref byte[,] blockMtx, ref byte[,] roundKeys, byte roundIndex) |
| {                                             |
|     for (byte i = 0, j; i < Faes.rowSize; i++) |
|     for (j = 0; j < Faes.columnSize; j++)      |
|         blockMtx[i, j] ^= roundKeys[roundIndex, i, j]; |
| }                                             |

4.2.1 Histogram analysis

For an image with a brightness of [0, L - 1], the histogram can be described by a discrete function:

\[ H(rk) = nk, \]

where, \( rk \) is the brightness intensity; \( nk \) is the number of image pixels with brightness intensity of \( rk \) \([12]\). For the target image of histogram analysis, the number of pixels with brightness of 0, 1, 2 \ldots \ L-1 is calculated. The results are placed on the vertical axis. The histogram provides useful statistics about the image, which facilitates image compression, and segmentation. The histogram of encrypted images must be uniform \([13, 14]\). Figures 3-5 present the original, encrypted and decrypted states of the sample images, respectively. The R, G, B values are given on the right of these states. A uniform distribution has been obtained for all values, as shown by the coded histograms in the images. The histogram of the original images and the encrypted images are very different from one another. A uniform distribution makes it challenging to draw statistical inferences and makes statistical attacks on the suggested encryption technique more difficult.

4.2.2 Root Mean Square Error (RMSE)

This section investigates whether there is a difference between the original image and the decoded image, using the RMSE defined in formula (4). This metric measures the magnitude of error in quadric terms. As the standard deviation of the estimation errors, it is often used to find the distance between the predicted and true values. The value range of RMSE is from 0 to \( \infty \). The smaller the RMSE, the better the prediction. If RMSE equals zero, the model has made no error \([15]\):

\[ RMSE = \sqrt{\frac{\sum_{i=0}^{n} e_i^2}{n}} \]  

(4)

where, \( A_i \) and \( P_i \) are the actual and predicted values, respectively; \( e_i = A_i - P_i \) is the error; \( n \) is the size of the dataset.

The magnitude of the error, absolute error, and squared error can be respectively expressed as \( D1 = A_i - P_i \) is the error; \( n \) is the size of the dataset.

The results in Table 6 demonstrate that no loss arises from encryption in the original images, for the decoded image that was acquired from the encrypted image has no difference. The histogram density graphs of the encrypted and decoded images are identical, as can be seen in Figures 3, 4, and 5.
Figure 3. Histogram of the Lena image

Figure 4. Histogram of the baby image

Figure 5. Histogram of the baboon image
This metric is a yardstick of the indicating that the approach proposed approach, useful improve original analyzing differential

\[ U \]

ACL is another metric of image encryption efficiency. According to the results in Table 1 about the

\[ 100\% \]

\[ \sum_{i=1}^{M} \sum_{j=1}^{N} D(i, j) \times \% \]

\[ NPCR = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} D(i, j) \times \% 100 \]  

\[ NPCR = 0.9960 \]  

\[ UACI \]

\[ U = \left( \sum_{i=1}^{M} \sum_{j=1}^{N} \left| C(i, j) - C(i, j) \right| \right) \times \% 100 \]  

\[ UACI = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} \left| C(i, j) - C(i, j) \right| \times \% 100 \]  

\[ \text{Proposed method} \]

\[ \text{Fibonacci polynomial matrix} \]

Table 6. RMSE values of images

<table>
<thead>
<tr>
<th>Sample pictures</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lena</td>
<td>0</td>
</tr>
<tr>
<td>Baboon</td>
<td>0</td>
</tr>
<tr>
<td>Baby</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7. UACI and NPCR test results

<table>
<thead>
<tr>
<th>Sample pictures</th>
<th>NPCR</th>
<th>(UACI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lena</td>
<td>0.9961</td>
<td>0.335</td>
</tr>
<tr>
<td>Baboon</td>
<td>0.9960</td>
<td>0.334</td>
</tr>
<tr>
<td>Baby</td>
<td>0.9960</td>
<td>0.333</td>
</tr>
</tbody>
</table>

Table 8. Comparison of the proposed method with the studies in the literature

<table>
<thead>
<tr>
<th>Reference</th>
<th>Method</th>
<th>Image</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artuğer and Orzkaynak [21]</td>
<td>Random (chaotic) Sbox</td>
<td>Baboon</td>
<td>NPCR= 0.9960 UACI=0.3343</td>
</tr>
<tr>
<td>Zhu et al. [22]</td>
<td>1D chaotic map</td>
<td>Lena</td>
<td>NPCR= 0.9963 UACI=0.3347</td>
</tr>
<tr>
<td>Benlashram et al. [23]</td>
<td>3D chaotic map and Pixel shuffling</td>
<td>Lena</td>
<td>NPCR= 0.9965 UACI=0.3360</td>
</tr>
<tr>
<td>Shariatzadeh et al. [24]</td>
<td>Dynamic AES</td>
<td>Lena</td>
<td>NPCR= 0.9999 UACI=0.3364</td>
</tr>
<tr>
<td>El-Latif et al. [25]</td>
<td>Arnold cat map</td>
<td>Lena</td>
<td>NPCR= 0.9921 UACI=0.3343</td>
</tr>
<tr>
<td>Norouzi et al. [26]</td>
<td>hyper-chaotic system</td>
<td>Lena</td>
<td>NPCR= 0.9915 UACI=0.3338</td>
</tr>
<tr>
<td>Ge and Ye [27]</td>
<td>3D cat map</td>
<td>Lena</td>
<td>NPCR= 0.9961 UACI=0.3349</td>
</tr>
<tr>
<td>Sayed et al. [28]</td>
<td>2D Affine Transformation</td>
<td>Lena</td>
<td>NPCR= 0.9960 UACI=0.3348</td>
</tr>
<tr>
<td>Zhang et al. [29]</td>
<td>3D Chaotic map and DNA coding</td>
<td>Lena</td>
<td>NPCR= 0.9960 UACI=0.3346</td>
</tr>
<tr>
<td>Nematzadeh et al. [30]</td>
<td>Deoxyribonucleic Acid (DNA) sequence and Binary Search Tree (BST)</td>
<td>Lena</td>
<td>NPCR= 0.9962 UACI=0.3354</td>
</tr>
<tr>
<td>Zhang and Wang [31]</td>
<td>Deoxyribonucleic acid (DNA) encoding and chaotic system</td>
<td>Lena</td>
<td>NPCR= 0.9961 UACI=0.3345</td>
</tr>
<tr>
<td>Suseela et al. [32]</td>
<td>Torus Automorphism and Rubik’s cube</td>
<td>Lena</td>
<td>NPCR= 0.9984 UACI=0.3687</td>
</tr>
<tr>
<td></td>
<td>Proposed method</td>
<td>Lena</td>
<td>NPCR= 0.9961 UACI=0.335</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Baboon</td>
<td>NPCR= 0.99602 UACI=0.334</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Baby</td>
<td>NPCR= 0.99603 UACI=0.333</td>
</tr>
</tbody>
</table>

As shown in formula (5), the ideal value for the NPCR value is 100%. According to the results in Table 1 about the proposed approach, the NPCR value is very close to 100%, indicating that the approach is extremely precise to the smallest changes in the input. Hence, changing a pixel value in the original image causes all pixels in the encrypted image to change.

UACL is another metric of image encryption efficiency. This metric is a yardstick of the avalanche effect:

\[ UACL = \left( \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{\left| C(i, j) - C(i, j) \right|}{255 \times M \times N} \right) \times \% 100 \]

NPCR is one of the criteria for the effect of differential attacks on image encryption. NPCR evaluates an algorithm's sensitivity to minute alterations in the original image. First, the original image is subjected to the C1 cryptographic technique in order to determine the NPCR. Then, a randomly chosen pixel from the original image is altered. After that, the encryption algorithm is applied to the image of which a pixel has been modified again, yielding a second encrypted image, denoted C2. The following formulae are used to calculate the NPCR value [19, 20].

\[ NPCR = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} D(i, j) \times \% 100 \]

5. CONCLUSION

Confusion and diffusion features are two basic requirements for encrypted text or images, and have been studied extensively in history. The most well-known solution is AES, which adopts irreducible polynomials. In the AES encryption
algorithm, the same matrix is used for column shuffling in every cycle. By contrast, the proposed approach solves the Fibonacci polynomial matrices used in confusion and diffusion differently in each cycle. This helps to deal with more complex scenarios.

This study mainly presents a new encryption algorithm based on Fibonacci polynomials and matrices, providing an alternative to other encryption algorithms designed to meet the needs of confusion and diffusion features. The success of our algorithm was evaluated through encryption and decryption of 3 different images. The results of histogram analysis show that the encryption results in a uniform distribution. Besides, the RMSE (which is 0 for the 3 images) suggests no difference between the original and decoded images. Further, NPCR (approximate 100%) and UACI (approximate 33%) test results prove that our algorithm works successfully.

The proposed system was demonstrated to be practicable. By making the system more complex, the system security is greatly increased. The future research will propose novel encryption techniques using polynomials and matrices of number series like Lucas, Pell, Jacobsthal, Padovan, and Perrin. New S-box tables can also be created as an alternative to the s-box tables of AES.

REFERENCES


[15] https://veribilimcisi.com/2017/07/04/mse-rmse-mace-


