Data Encryption Using Image Processing: A Brief Study

Mahmud Rahman Parag¹* and Tania Akter Setu²

¹²Department of Computer Science and Engineering, University of Information Technology & Sciences, Dhaka, Bangladesh.
*Correspondence: 2010352011@uits.edu.bd (Mahmud Rahman Parag, Department of Computer Science and Engineering, UITS, Dhaka, Bangladesh).

ABSTRACT

We can see considerable growth in the field of science and technology over the past few decades. This growth has also brought an enormous threat to the data used by users. Encryption and decryption of data play an important role in safeguarding the data. Many authors have contributed much outstanding research on unauthorized access to user data. In this journal, I have proposed an algorithm for data encryption using images. The main aim is to provide a new and highly customizable approach to protecting data using images and eliminating the use of plaintext. In my research, I will use images to encrypt and decrypt data rather than using plaintext.

Keywords: Cryptography, Data, Encryption, Decryption, Image, Image Pixel, and RGB.

INTRODUCTION:

In this digitalized era, data is everywhere. From daily messaging to storing friends' information on our mobile, from Facebook to criminal information in police office databases, everything is digitalized data. Protecting it from unwanted access is becoming a must to keep our privacy. Cryptography protects this data by changing it into unclear data that can only be accessed via authorized receivers, who then convert the uncertain data into the original textual content (Alemami et al., 2019). By this, we eliminate any kinds of unauthorized access and protect our data.

There are many existing solutions for data encryption. AES and RSA are one of the most used and popular algorithms. AES and RSA heavily relied on plaintext as a key. A long key made encryption much harder to break, but the catch is you have to remember the key. If someone uses different plaintext to encrypt different data then the problem gets much bigger. On top of that, if someone saves their key in a text file then anyone can suspect that text file is a key. On the other hand, AES has a key length limitation. Here, in my thesis, I have shown an alternate approach to eliminate plaintext and use an image as a key for data encryption. This algorithm uses image pixel values as a key and encrypts and decrypts data. On top of that, there is no key length limitation in this algorithm as long as someone has enough disk space and processing power.

Literature Review

Cryptography is a way to conceal confidential information from a third party by implementing keys that are only known by the communicating parties. For example, let us assume that two devices are communicating with themselves. The sender will be A and the receiver will be B. When A will send the message A will the message using a key so that no other than A and B can see the message and when B will receive the
message, B will use the same key as A to decrypt the message to get the original data back. While C who is an eavesdropper won’t be able to get the original message as it is encrypted which is shown in Fig. 1.

**Fig. 1: Secure Data Transfer Between A and B.**

**Classification**
Cryptography is classified into two categories: Symmetric and Asymmetric Key Cryptography. Sometimes we can find a combination of symmetric and asymmetric key cryptography in some applications called hybrid schemes. It is because both types of algorithms have their strength and weakness. When they are combined, they protect each other weaknesses.

**Symmetric Key Cryptography**
Symmetric Key Cryptography is a type of cryptography where a single secret key is used for both encryption and decryption. When parties send a message to another party, they use a single key to encrypt and decrypt their data.

![Symmetric Key Cryptography](image)

**Fig. 2: Symmetric Key Cryptography.**

This Symmetric Key Cryptography has two types of structure.

**Block cipher**
In these types of cipher, data is divided into blocks and then encrypted at the same time. There are plenty of examples of this type of structure. One of the most popular algorithms which are used worldwide is the AES algorithm.

**Asymmetric Key Cryptography**
Asymmetric Cryptography is also called public-key cryptography. In asymmetric public-key cryptography, there are two keys: a private key and a public key are used (Jirwan et al., 2013). When someone wants to encrypt their data, they generate these two keys and use their private key to encrypt their data and use their public key to decrypt. The RSA algorithm is the best example of this kind of cryptography.

![Asymmetric Key Cryptography](image)

**Fig. 3: Asymmetric Key Cryptography.**

**Encryption and Decryption of Data Using Image Algorithm**
Encryption and Decryption of data using images need several steps which are enlisted below:

**Encryption Process:**
1. Plaintext Processing
2. Key Generation
3. Encryption

**Decryption Process:**
1. Key Generation
2. Decryption

All designs of these processes will be explained in detail below. Before entering encryption and decryption 3 key variables should be initialized.

1) NUMBER_OF_COLUMN_TO_MAP
2) NUMBER_OF_ROW_TO_MAP
3) RANDOMIZE_BOX

These three variables should be implemented in the algorithm for encryption and decryption and must be the same for both. If all values of these three are chan-
ged in the algorithm, then all values of these three variables have to be changed and have to be the same in the decryption algorithm also for a given connection. For instance, if a and b want to transfer data to each other then a will encrypt his data and b will decrypts data using the same value of these three variables.

Encryption Process

Plaintext Processing

It is a mapping-like algorithm. It maps several characters to several boxes. We use two important parameters:

1) Number of characters to map per row (character = byte)

2) The number of rows and columns need to map into

Number of Characters to map per row have to be smaller than or equal to the number of rows needed to map. These two parameters are hardcoded in the algorithm. Now, it takes the plain text and split the text depending on those two parameters if there are any remaining characters it will take those characters after doing the encryption of previous characters. For example, let us assume the plain text is “00112233445566778899AABBCCDDEEFF”, the Number of characters to map per row is 2, the Number of rows needs to map into is 16, and the number of columns needs to map is 10. So, the actions are:

1) Convert every character to their ascii value. So, the ascii hex value of plaintext will be 48 48 49 49 50 50 51 51 52 52 53 53 54 54 55 55 56 56 57 57 65 65 66 66 67 67 68 68 69 69 70 70

2) Then, from the parameter, every two ASCII values will be mapped into every 10 rows. It will be like this Fig. 4

Key Generation

In this part, we take an image as a key. Let us assume, we take a 20 X 20 size image which is 20 * 20 = 400 * 8 = 3200 bit key. We take any channel from R, G, or B and use it as a key.

```
Simple MATLAB Code
```

Fig. 5: Simple MATLAB Code for plaintext processing.

Fig. 6: A 20 X 20 Size Image as a key.

Fig. 7: Image Red Channel.

Fig. 8: Image Green Channel.
Let’s take the red channel for demonstrating the example. After taking a channel, split it according to the value of the Number column to map. So, we split it on every 10 values until we reached the end of the red channel. If the last split misses some value, we will fill up those values using the beginning value of the red channel. Fig. 9 will depict the process.

**Fig. 9: Image Blue Channel.**

**Fig. 10:** Final Processed Key.

**Simple MATLAB Code**

```matlab
% Define the key rows
key = [151 150 149 148 147 146 145 144 143 142];

% Perform XOR operation on the mapped plaintext by the first key row
xor_result = xor(mapped_plaintext, key(1));

% Perform circular right shift on the XOR result
shift_amount = length(key);
shifted_result = circshift(xor_result, shift_amount, 1);

% Generate a fixed random matrix/array
random_matrix = randi([0, 255], [1, length(key)]);
random_matrix = unique(random_matrix);

% Randomize the right-shifted row cells according to the random number
randomized_row = randperm(length(key));

% Continue the Circular Right Shift and Randomize the Row until all the rows of the result of XOR are done.
```

**Fig. 11:** Simple MATLAB Code for Key processing.

**Fig. 12:** First Row of the key.

**Encryption**

We begin the encryption by XORing all mapped plaintext by the first key row.

**Fig. 13:** XOR of the first row and all rows of mapped plaintext.

After that, we do a circular right shift to the first row of the result of XOR where the amount is the number of rows needed to map subtract to the current row number.

**Fig. 14:** Circular right shift of the first row.

After that, we generate a fixed random matrix/array where the length of it is equal to the number of the column. These random numbers are unique and should not be greater than the number of columns.

**Fig. 15:** A Fixed Random Matrix/array.

Then, we randomize the right-shifted row cells according to the random number.

**Fig. 16:** Randomized Cells of the First Row.

And we continue to do the Circular Right Shift and Randomize the Row until all the rows of the result of XOR are done.

**Fig. 17:** Full result of Circular Right Shift, and Randomize of every Row.
Then we again do a circular right shift to the entire matrix/array. Where the amount of right shift is the number of keys subtracted from the row number of the current key.

Fig. 18: The full matrix of the result of the circular right shift.

Then we repeat the full process until all of the keys are used. In other words, all of the rows of the key matrix are used. And the final matrix will be the chipper text.

Fig. 19: The Chipper Matrix.

Flow Diagram

![Flow Diagram of Encryption Process.](image)

Fig. 20: Flow Diagram of Encryption Process.

**Simple MATLAB Code**

```matlab
SimpleKey = AES_OF_KEY(key);

for i = 1:length(SimpleKey)
    BlockText = text2num(SimpleKey(i));
    tempBlockText = circshift(BlockText, -num_shift);
    tempBlockText = num2text(tempBlockText);
    BlockText = tempBlockText;
end

PlainText = text2num(BlockText);
```

**Decryption Process**

**Key Generation**

Generating a key in the Decryption process is the same as the key generation processes of Encryption.

**Decryption Process**

The decryption algorithm is the exact opposite of the encryption algorithm. First, we convert the chipper text to the number of columns needed to map by the Number of rows needed to map the matrix then we do a full circular left shift and then we reconstruct the row from the randomized row and do a circular left shift until all row is done and then XOR it from the last row number or the last key until the first. And the result will be plain text.

Fig. 22: The Plain text Matrix.

![The Plain text Matrix.](image)
**Flow diagram**

![Flow Diagram of Decryption Process]

**Fig. 23:** Flow Diagram of Decryption Process.

**Simple MATLAB Code**

```matlab
% key = [456; 789; 123; 456];
% plaintext = [0:9; 0:9; 0:9; 0:9];
% ciphertext = encrypt(plaintext, key);
% decrypted = decrypt(ciphertext, key);
% assert(decrypted == plaintext);
```

**Fig. 24:** MATLAB Code for Decryption Process.

**Evaluation**

**Introduction**

To implement it in a system, it has to pass some speed tests. To evaluate it properly, we are going to do a speed test for different keys so that we can conclude which key is most appropriate for general use in terms of length.

**Speed Measurement**

For the speed test, we will use 256-bit text and 3200-bit, 28800-bit, and 20280000-bit images. We are going to five times for every key and going to take their average.

3200-bit KEY:

**Encryption**

**Table 1:** Encryption Time Measurement for 3200-bit KEY.

<table>
<thead>
<tr>
<th>Number of Run</th>
<th>Time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.051 s</td>
</tr>
<tr>
<td>2</td>
<td>0.048 s</td>
</tr>
<tr>
<td>3</td>
<td>0.055 s</td>
</tr>
<tr>
<td>4</td>
<td>0.045 s</td>
</tr>
<tr>
<td>5</td>
<td>0.055 s</td>
</tr>
</tbody>
</table>

Average time: 0.051 s

**Decryption**

**Table 2:** Decryption Time Measurement for 3200-bit KEY.

<table>
<thead>
<tr>
<th>Number of Run</th>
<th>Time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.036 s</td>
</tr>
<tr>
<td>2</td>
<td>0.050 s</td>
</tr>
<tr>
<td>3</td>
<td>0.038 s</td>
</tr>
<tr>
<td>4</td>
<td>0.034 s</td>
</tr>
<tr>
<td>5</td>
<td>0.039 s</td>
</tr>
</tbody>
</table>

Average time: 0.039 s

28800-bit KEY:

**Encryption**

**Table 3:** Encryption Time Measurement for 28800-bit KEY.

<table>
<thead>
<tr>
<th>Number of Run</th>
<th>Time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.073 s</td>
</tr>
<tr>
<td>2</td>
<td>0.112 s</td>
</tr>
<tr>
<td>3</td>
<td>0.072 s</td>
</tr>
<tr>
<td>4</td>
<td>0.089 s</td>
</tr>
<tr>
<td>5</td>
<td>0.103 s</td>
</tr>
</tbody>
</table>

Average time: 0.090 s

**Decryption**

**Table 4:** Decryption Time Measurement for 28800-bit KEY.

<table>
<thead>
<tr>
<th>Number of Run</th>
<th>Time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.051 s</td>
</tr>
<tr>
<td>2</td>
<td>0.048 s</td>
</tr>
<tr>
<td>3</td>
<td>0.055 s</td>
</tr>
<tr>
<td>4</td>
<td>0.045 s</td>
</tr>
<tr>
<td>5</td>
<td>0.055 s</td>
</tr>
</tbody>
</table>

Average time: 0.051 s

Plaintext: 00112233445566778899AABBCCDDEEFF

Plaintext: 00112233445566778899AABBCCDDEEFF
20280000-bit KEY:

Plaintext: 00112233445566778899AABBCCDDEEFF.

Encryption

Table 5: Encryption Time Measurement for 20280-000-bit KEY.

<table>
<thead>
<tr>
<th>No.</th>
<th>Time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17.250 s</td>
</tr>
<tr>
<td>2</td>
<td>25.497 s</td>
</tr>
<tr>
<td>3</td>
<td>25.487 s</td>
</tr>
<tr>
<td>4</td>
<td>25.456 s</td>
</tr>
<tr>
<td>5</td>
<td>25.551 s</td>
</tr>
<tr>
<td>Average time</td>
<td>21.849 s</td>
</tr>
</tbody>
</table>

Decryption

Table 6: Decryption Time Measurement for 2028-0000-bit KEY.

<table>
<thead>
<tr>
<th>No.</th>
<th>Time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25.456 s</td>
</tr>
<tr>
<td>2</td>
<td>25.246 s</td>
</tr>
<tr>
<td>3</td>
<td>25.228 s</td>
</tr>
<tr>
<td>4</td>
<td>25.524 s</td>
</tr>
<tr>
<td>5</td>
<td>25.064 s</td>
</tr>
<tr>
<td>Average time</td>
<td>25.304 s</td>
</tr>
</tbody>
</table>

CONCLUSION AND RECOMMENDATIONS:
People use cryptography to safeguard their personal and commercial data. There are many algorithms available for that like AES, and RSA but a problem with those is a limited length of key and every complex algorithm to implement. On top of that, remember plain text key is a tough job. In this research, I have used an image as a key, which opens up a door to remove the plain text key and introduce a limitless bit of key which doesn't require memorization of the key. It also provides a highly customizable algorithm where we can also use any kind of matrix-based mathematical function which produce an 8-bit int result and has an inverse function. This algorithm has a huge amount of chance for customization. Combining it with AES may create more security than it is now. Also, through some research, there are some possibilities of optimizing this algorithm for the longer keys to reduce time complexity.

ACKNOWLEDGEMENT:
First of all, I would like to express my profound gratitude to my honorable supervisor, Ms. Tania Akter Setu, for her constant and meticulous supervision, valuable suggestions, patience, and encouragement to complete the project work. I would also like to thank the CSE department of the UITS for providing us with the opportunity to have an industrial-level design experience as part of our curriculum for the master’s program. Finally, I would like to thank our families and everybody who supported us and provided us with guidance for the completion of this project.

CONFLICTS OF INTEREST:
The authors state that there is no potential conflict of interest in publishing this research article.

REFERENCES: