

# **Image Encryption Based Triple - DES Cryptosystem**

**- MOHAMED AJUMAL M**

**AALIM MUHAMMMED SALEGH COLLEGE OF ENGINEERING**

## **Abstract**

When encryption grey-scale images are stored using a lossless format (e.g. not JPEG), and a fixed key for a symmetric cryptosystem is used, edges may reveal due areas of the same colour in the image, thus giving some information about the original image. This is particularly problematic when the image contains only two colours, black and white being the worst case scenario. The former problem may be solved using an asymmetric system, but the time employed in its operation is a relevant detrimental factor. However, using the JV Theorem along with TripleDES | with the modification that a variable permutation is used on the input string for the third round of the encryption process first cycle— the output is a greyscale image whose tones present a uniform distribution, according to a chi-square goodness-of-fit test. Also, it is shown that the use of variable permutation strengthens the Triple-DES cryptosystem. On the other hand, since the images are being encrypted using a symmetric cypher algorithm, the processing time is less than that of any known asymmetric algorithm. Furthermore, a criterion for computing the amount of permutations is provided.

**Keywords.** JV Theorem, Chi-square goodness-of-fit test, Triple-DES, Image Encryption, Variable permutations

## 1. Introduction

In the present work, a variation of the standard Triple-DES is used. In the encryption process of this modified Triple-DES, in the and option [1].

The initial permutation —according to the international norm— is not used at the beginning of the cipher process [5]. Instead a variable permutation is applied to the input of the third round of the first encryption cycle and its inverse at the end of the cypher process.

The former is noted as; where is the above mentioned permutation over a 64 bits long string, ; are two 64 bits long keys, and is a 64 bits long plaintext string.

It is also noteworthy that there are images that present areas of the same colour. When such an area is ciphered using fixed keys and permutation, it gives the same grey-scale value for the whole area, which gives rise to distinct edges, thus allowing the emergence in the encrypted image of useful information regarding the original image. One way of avoiding this situation is encrypting the image using a number of different variable permutations in order to obtain a different grey-scale tone.

In order to determine how much the distribution of the grey-scale values on the output image resembles a random distribution, Chi-square ( $\chi^2$ ) goodness-of-fit test is used [15]. This is done in a particular case considering a grey-scale image containing only two colour values (black and white), also proposing a criterion to compute the number of permutation to be used.

The variable permutations used here are obtained using the JV theorem algorithm [14], parting from a number  $1 \leq \leq 64! - 1$ , which is selected randomly. The histogram of the grey tones of an encrypted image is presented, for a particular instance of keys and permutation, in order to illustrate how such distribution approaches a uniform distribution. In this regard, the  $\chi^2$  goodness-of-fit test gives a criterion to decide when an encrypted image should be rejected for not complying with a given degree of randomness. For this a Normal distribution is used to approximate the  $\chi^2$  [15].

Additionally, an algorithm is provided to reduce the time used to calculate the permutations. That is, the permutation associated to with  $1 \leq \leq 64! - 1$  is found, and the other permutations are obtained from this without computing the divisions on the larger factorial (e.g. 63!, 62!, and also so on) [14].

## 2. Modified Triple-DES Goodness-of-Fit Test

Let us begin the current section by answering the question: why should a variable permutation is applied to the input of the third round of the first encryption cycle on the modified Triple-DES algorithm?

It is clear that for a grey-scale image, each pixel is represented by one byte, thus each 64 bits data block contains 8 pixels. Then, a binary image— where each pixel has one of only two values: white and black, represented by 0 and 1— the encryption process may have as input 64 bits long strings such as 000.....0 or 111.....1. In this regard, any initial permutation, whichever it is, would not modify such strings at all. It follows that the first round in which the left and the right blocks are mixed is after the second round: that is, the input to the third round. On the other hand, it is easy to notice that the variable permutation does not need to be applied to the input string to the third round: in fact, it could be applied to the input of the fourth, fifth, or even the 16-th round of the first cycle. Then, why use it specifically on the third round? Simply, because by being the first round with a mix of zeroes and ones, any modification caused by the permutation of this round will have the most rounds after it to mix the information. Recall that the Feistel function [4], along with the keys program and the permutations, are responsible for the task of data mixing in each round.

The kind of grey-scale images most troublesome to encrypt for a symmetric cryptosystem with fixed keys and permutation are precisely images with only two colours (i.e. black and white). This difficulty is due to the emergence of edges and areas with uniform tones in the encrypted image, which gives away useful information. Thus, the current work uses such an image for the experimental tests, where the background is white; there is a black rectangle along with some black characters. The image used is 598 pixels wide and 488 pixels high, as shown in figure 1.

In the rest of the paper, any image is understood to be encrypted if the goodness-of-fit test for the histogram of the potentially encrypted image accepts the null hypothesis; where said null hypothesis states that the frequencies of the different grey tones of the image fit a uniform distribution, against the test hypothesis which states the contrary.

In any problem of hypothesis testing, there are two kinds of error: that which is incurred when the null hypothesis is rejected while being true (Type I) and when the null hypothesis is accepted while being false (Type II) [15].

The error of interest (to be controlled) is the type I error, for which there are some standard values 0.1, 0.05 or 0.01. If the value  $\alpha = 0.1$  is taken for this error, it means that greater evidence is required to accept the null hypothesis, according to the data, than if any of the others values is taken. This last statement is based on the following arguments.

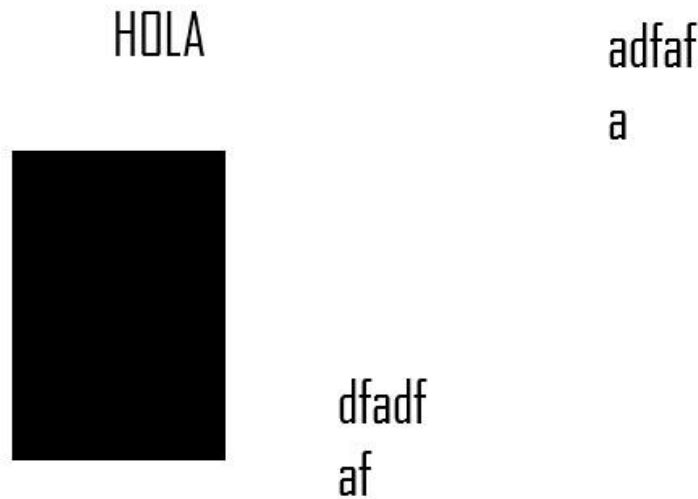


Figure 1: Binary image used in experiments, of dimensions 598x488 pixels

Notice that the quality of the encrypted image is strongly related to how much its histogram fits a horizontal rect line. This can be verified with the  $\chi^2$  goodness-offit test:

$$(2.1) \quad \chi^2 = \sum \frac{(o_i - e_i)^2}{e_i}$$

with  $k - 1$  degrees of freedom, where  $o_i$ ,  $e_i$  are the  $i$ -th observed and expected values, respectively. A small value of  $\chi^2$  indicates a good fit between the observed and the expected data values. For this particular instance, such a small  $\chi^2$  means that the histogram is quite close to having a uniform distribution of values. —In this sense, it is said that the case of  $\alpha = 0.1$  requires more evidence of the histogram being quite similar to a horizontal line than for the cases of  $\alpha = 0.05$  or  $0.01$ , for the null hypothesis to be accepted.

Thus, in order to accept or reject the null hypothesis, the value of the limit number of the right tail of the  $\chi^2$  distribution associated to  $\alpha = 0.1$ ,  $0.05$  or  $0.01$  is taken as a reference. If the computed value for  $\chi^2$  (according to equation 2.1) is greater than this limit value, the null hypothesis is rejected.

Since the grey tones considered in the histogram are 256, it follows that the degree of freedom for the  $\chi^2$  distribution is  $k - 1 = 256 - 1 = 255$ . On the other hand, the Normal distribution —with  $\mu = 255$  and  $\sigma = \sqrt{255}$  — is a good approximation to the  $\chi^2$  distribution with 255 degrees of freedom thanks to the Central Limit Theorem [8]. Thus, the limit values for which any greater value of the computed  $\chi^2$  will make the null hypothesis be rejected are: 284.13, 292.26 and 307.61 for each value of  $\alpha$  equal to 0.1, 0.05 and 0.01, respectively.

In order to analyze the worst case scenarios for this application, let us consider some background concepts first. Thus, according to [14], any integer number  $1 \leq m \leq 64! - 1$  may be written as follows:

$$(2.2) \quad m = C_0 \cdot 0! + C_1 \cdot 1! + \dots + C_{63} \cdot 63!$$

where  $C_i \in \{0, 1\}$  and  $0 \leq C_i \leq 63 - i$  for  $0 \leq i \leq 63$ . These  $C_i$  allow us to find the permutation associated to  $m$ . Assuming that the increments are of the order of  $10^{61}$  (which is close to  $48!$ ), then one permutation and the next will have differences in positions 16 to 63 [14], where the initial position is 0 at the left end.

On the other hand, let's suppose that the values taken by each bit —0 or 1— are taken independently from each other, regardless of their positions, and are equiprobables (i.e. they can be either 0 or 1, both values with probability of  $1/2$ ). Then, the probability of all positions 0 to 63 being equal (i.e. either all these positions are zero or all positions are one) can be computed according to the Bernoulli model [13]:

$$(2.3) \quad 2 \cdot \left( \frac{1}{2} \right)^{64}$$

for  $0 \leq i \leq 15$ .

Notice that positions 0 to 15 may have any value, given that the values of such positions for a given permutation and the following are equal.

Now, since the worst situation happens when the permutations applied to the input string to the third round causes no changes to the string, these kinds of situations are those we want to avoid. Since such lack of changes happens when the input string is made up only by zeroes, or only by ones. Considering the former discussion, the probability computation for 64 bits strings of the former - 0...0; - 1...1 where  $i = 0, 1$  and  $0 \leq i \leq 15$  is as follows:

$$(2.4) \quad \left[ \sum_{i=0}^{15} \left( \frac{1}{2} \right)^{64-i} \right] \cdot \left( \frac{1}{2} \right)^{64-i} = \left( \frac{1}{2} \right)^{64-i}$$

Now, in order to compute the probability of the events  $11...1$  and  $y_0 11...1$  where both strings are 64 bits long and  $y_0 = 0, 1$ ; if both probabilities are added, an amount larger than 1 must be subtracted, since the intersection of both events is not empty. However, the addition of both probabilities becomes an upper bound. In this sense, the probability of such a worst case scenario happening is less than

$$(2.5) \quad \sum_{i=0}^{15} 2 \cdot \left( \frac{1}{2} \right)^{64-i} \approx 1.42 \cdot (10)^{-14}$$

Thus, the worst case scenario for the proposed cryptosystem (regarding the input to the third round of the first cycle) happens when the input string to the third round consists of only zeroes (000...0) or only ones (111...1) from position 16 to position 63, or from position 15 to 63, and so on until from position 0 to 63. Such situations may happen with a probability of less than— $1.42 \times 10^{-14}$ .

### 3. Algorithm for Reducing Permutations Computing Time

Given that the proposed modifications to the Triple-DES needs to calculate a large number of permutations, it is desirable to make the time necessary for such computations as small as possible. Preferably, we would like to have a procedure to reduce the amount of time or bit operations needed, when compared to the original algorithm (as introduced in [14]). This time reduction may be accomplished based on the fact that, as mentioned before, any integer  $1 \leq m \leq 64!$  can be written as shown in equation 2.2, which leads to the permutation associated to  $m$  [14]. Such permutations are obtained from an integer (such that  $1 \leq m \leq 64! - 1$ ) and the increment  $\Delta$ , which has been chosen to be  $10^{61} < 48!$ . Thus, the number  $x + \Delta$  is associated to a permutation which is equal to that of number up to position  $63 - 48 = 15$ , which means that the differences between both permutations appear in position 16 to 63.

The increment value  $\Delta$  is chosen as  $10^{61}$  for a particular reason: a smaller value increases the probability of a worst case scenario happening (see equation 2.5), while a larger value increases the encryption time. Given this trade-off, in the current work the proposed increment is  $10^{61}$ . Furthermore, the increments  $c\Delta$  with  $c = 1; 2, \dots$  must comply with the following condition:

$$(3.1) \quad \%3\Delta < 64! - 1$$

Then, the algorithm to find a permutation given that  $\Delta$  and  $\pi$  are known is the following.

1. - The value of  $x$  is expressed (according to equation 2.2) as:

$$(3.2) \quad x = \pi_1 \cdot 63! + \pi_2 \cdot 62! + \dots + \pi_{63} \cdot 1! + \pi_{64} \cdot 0!$$

where  $\pi_i \in \{0, 1, \dots, 63\}$

2. - The permutations  $\pi_1, \pi_2, \dots, \pi_{64}$  is built according to the values  $\pi_1, \pi_2, \dots, \pi_{64}$  following the algorithm introduced in [14].
3. The value  $6 \in \mathbb{Z}$  is found, such that  $6 > \Delta$  and is the smallest integer which holds to the former condition.
4. - The permutations associated to  $\Delta$  and  $\% \Delta$  have the following array in common:

$$(3.3) \quad \pi_1, \pi_2, \dots, \pi_{64} = 9$$

5. - The difference is computed:

$$(3.4) \quad \Delta = (\%3\Delta) - [!63! \% 62! \% \dots \pi_9]$$

6. - New constants are calculated from  $\Delta$ , such that:

$$(3.5) \quad \Delta = \pi_9 \cdot 6 < 1! \% \pi_9 \cdot 6 < 2! \% \dots \% \pi_1! \% \pi_0!$$

where  $\pi_0 = 0$ .

7. - Taking into account that the permutation is cut at  $\pi_9$  and that in the process to obtain the rest of the permutation values we still have the array 001, 011, ..., 06 < 11. Then, considering that array, the  $0^*1$ , and the constants  $\pi'$  from step 6, the rest of the permutations associated to  $\% \Delta$  is found:  $\pi^{>9} : , \pi^{>9} : , \dots, \pi^{>9} [14]$ .

In order to better illustrate how this algorithm works, let us analyze the following example of permutation. Assuming that we are working with strings of 8 positions— i.e. 0, 1, ..., 7— and that  $\Delta = 20531$ ,  $\Delta = 100$ ; then we follow the algorithm:

1. - The number  $\Delta$  is expressed as:

$$(3.6) \quad \Delta = 47! \% 06! \% 35! \% 04! \% 13! \% 22! \% 1! \% 00!$$

2. - Considering that  $\pi_4 = 0; 3; 0; \pi_1 = 2; \pi_2 = 1; \pi_0 = 0$  the associated permutation is: 4 0 5 1 3 7 6 2.

3. - The smallest integer  $6$  such that  $6! > 100$  is:  $6 = 5$ .

4. - Permutation associated to numbers 20531 and 20531+100 have the following array in common:  $\pi_1 4; \pi_0 y \pi_{B2} 5$ .

5. - The difference  $\Delta$  is computed:

$$(3.7) \quad \Delta = 20631 < 7! \% 6! \% \pi_{B2}$$

6. - Considering that  $\Delta = 111$ , the constants  $\pi_{B2}$  and

$\pi_A$ ,  $\pi_2$ ,  $\pi_1$  are computed; as mentioned before,  $\pi_0 = 0$ . Thus:  
 $\pi_4 = 4; \pi_2 = 2; \pi_1 = 1$

7. - According to [14], the  $0*1$  values are: 0011; 0112; 0213; 0316 and 05<1 417. Considering the constants found in the step 6; the permutation is therefore: 4 0 5 7 3 2 6 1.

Now, let us find how many bit operations are saved by using the proposed algorithm [11]. Given the most operations needed to transform a number into its corresponding permutation are divisions [14], an approximation to the difference on the bit operations saving will be done considering only divisions. Additionally, recall that if the number  $\varepsilon$  Z is represented with C bits, and  $D \varepsilon$  Z with 6 bits, with  $E$  D, then the amount of bit operations needed to compute the quotient and remainder when is divided by is estimated as  $F C6$  [11].

Thus, when a number of  $10^{89} \approx 64! - 1$  —which can be represented as a 296 bit long string— is transformed into a permutation, the amount of bit operations does not exceed  $2962(63) = 5519808$  [13]. The former situation happens because 63 divisions must be done, considering that both numerator and denominator for these divisions are of the order of  $2^{296}$ . Meanwhile, a number of  $10^{61} \approx 48! - 1$  may be represented with a string 201 bits long and needs no more than  $2012(47) = 1898847$  bit operation to be converted into a permutation. Now, by diving these results we have that  $5519808 / 1898847 \approx 2.9$ , which indicates that by following the proposed algorithm, the bit operations incurred by divisions becomes almost one third of the amount of bit operations needed otherwise.

#### 4. Experimental Results

As a preliminary step to the experimental results obtained and their discussion, let us find out how many permutations are needed given the proposed algorithm. First, the authors consider that a different permutation should be applied to each 64 bits block of data in the image to be encrypted, in order to make the tones distribution in the encrypted data blocks as close to random as possible. In this instance, each 64 bits data block includes 8 pixels, thus making the amount of permutations needed very straightforward to calculate, as  $|HI J^* 6K 8/|$ . Therefore, for the case of the image presented in figure 1, the amount of permutations is 36478.

As mentioned before, the proposal of this paper enables us to measure how much the distribution of grey levels of the encrypted image approaches a uniform distribution. This in turn allows us to determine how likely it is an image encrypted following the proposed system is rejected (i.e. the null hypotheses of the goodness-of-fit test is rejected), given the thresholds found for each level of type I error.

For this, 45000 element of the 4-tuple  $, , \Delta$  were chosen pseudorandomly [6], where is a permutation of the order of  $10^{80}$ , , are 64 keys bits each,  $\Delta$  is the increment between permutations of the order  $10^{61}$ . Remember, the amount  $|HI J^* 6K 8/| = 36478$  is fixed. The obtained percentages of rejection are shown in



table 1. Notice that these results are presented in intervals of 15000; this is done in order to illustrate the stability of the reported percentages.

Type I Error Level	Threshold for Rejection	15000 runs	30000 runs	45000 runs
0.1	284.13	9.40%	9.61%	9.64 %
0.05	292.23	5.23%	5.31%	5.26 %
0.01	307.61	1.39%	1.34%	1.37 %

Table 1: Reject rate percentages for intervals of 15000 runs.

Notice however, that the reject percentages bear no relation to the levels of probability of type I error. On the other hand, if the random variable of accepting or rejecting the null hypothesis is considered to be a Bernoulli variable [13] (i.e. 1 for rejection and 0 for acceptance), and its probability distribution is such that  $P(1) = J$  and  $P(0) = 1 - J$ , then the random variable

$$(4.1) \quad \bar{J} = \sum N_i / D$$

has a standard deviation equal to

$$(4.2) \quad \sqrt{J(1-J)/D}$$

Now, if the reject rates shown in table 1 for  $D = 45000$  are considered as estimator of  $J$ , for the different levels of type I error, then the corresponding confidence intervals may be computed [15]. For instance, for a confidence level of 99% (i.e. 0.3 times the standard deviation for each case), the corresponding intervals are shown in table 2.

Type I Error Level	Probability J	Confidence Interval
0.1	0.0964	0.0964 $\pm$ 0.0034
0.05	0.0526	0.0526 $\pm$ 0.001
0.01	0.0137	0.0137 $\pm$ 0.0005

Table 2: Confidence intervals for reject rate percentage with  $n = 45000$

Below is an example of the encryption processed, run on the image shown in figure 1; this example includes the encrypted image and its corresponding histogram for a given 4-tuple.

```
=
89915445645790121264579012126475888888889999956457901218991179
995621458709153705
= 03300000000000003
= 732456189abcdff2
```

$$\Delta = 3745012738810468536049153820452119930465723418450236784901257$$

Notice that given the length of the numeric values assigned to  $P$  and  $4$ , none of these numbers fit into the line. Thus, they appear in two lines to ease their readability, although the two lines correspond to one number, in both cases.

The amount of permutation needed is 36478; and these for values  $\chi^2 = 284.69$ .

The resulting encrypted image is presented in figure 2, while its corresponding histogram (with all grey-level values, from 0 to 255) is shown in figure 3.

At the beginning of this paper, it was mentioned that the inclusion of a variable permutations to the modified Triple-DES system increases its strength against attacks. However, is not the object of the current research to specify the magnitude of such strengthening? Yet, it is safe to presume the following the proposed system (i.e. variable permutation of the order of  $10^{80}$ , an increment between permutations  $\Delta$  of  $10^{61}$ , a pair of keys , ; and the amount of permutations used expressed as  $|HI \quad J*6K8/|$ ).

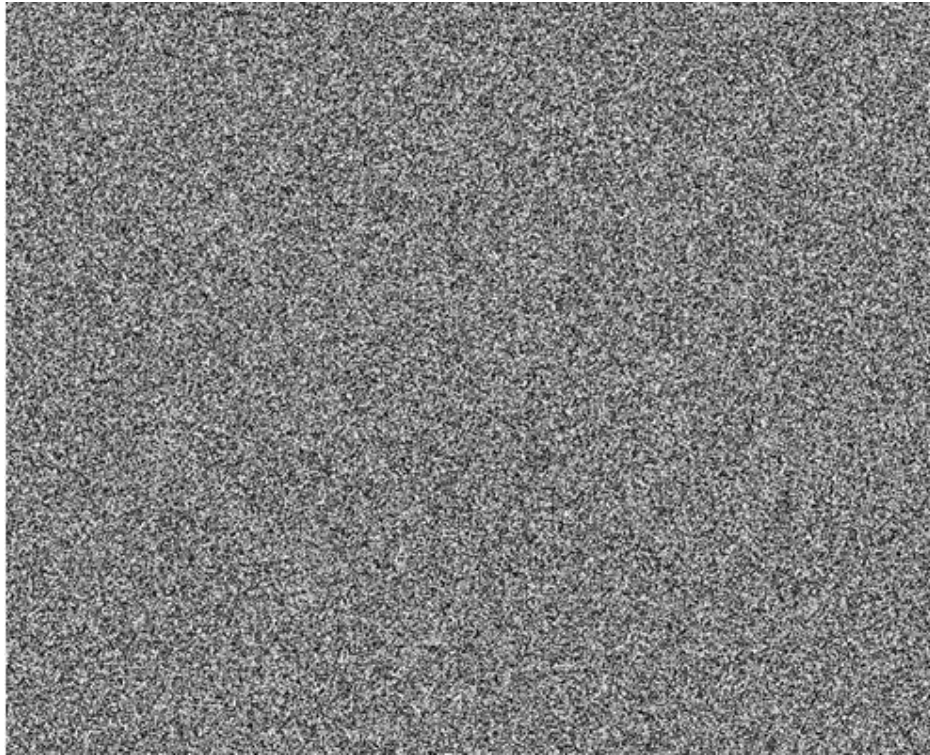


Figure 2: Image of figure 1 after encryption with parameters of the example.

When such an encrypted image is to be decrypted, knowing  $\Delta$ , the pair of keys , ; and the amount of permutations used, if only the previous permutation is changed then the result will not be the original image. This means that knowing the pair of keys , , as well as  $\Delta$  and the amount of permutations is not enough to decrypt the image: more information is needed. In this sense, it is said that Triple-DES is strengthened by including variable permutations.

On the other hand, the time needed to encrypt the image of figure 1 according to the former example is approximately 4 seconds. This is not a particularly onerous time, considering that it was run as ad-hoc program developed in C++, on a single core Pentium IV© system with 2GB of RAM memory.

Regarding what other researchers have reported in contemporary scientific literature, both [16] and [10] the present system focused on encrypting images, but gave no indication of their robustness or the complexity of a brute force attack on such cryptosystem. In the case of the current work, the proposed cryptosystem has a complexity against brute-force attacks of at least  $2^{112}$ . Additionally, since our system is symmetric, it is faster than most known asymmetric cryptosystems, such as RSA, ElGamal or Elliptic Curves [3].

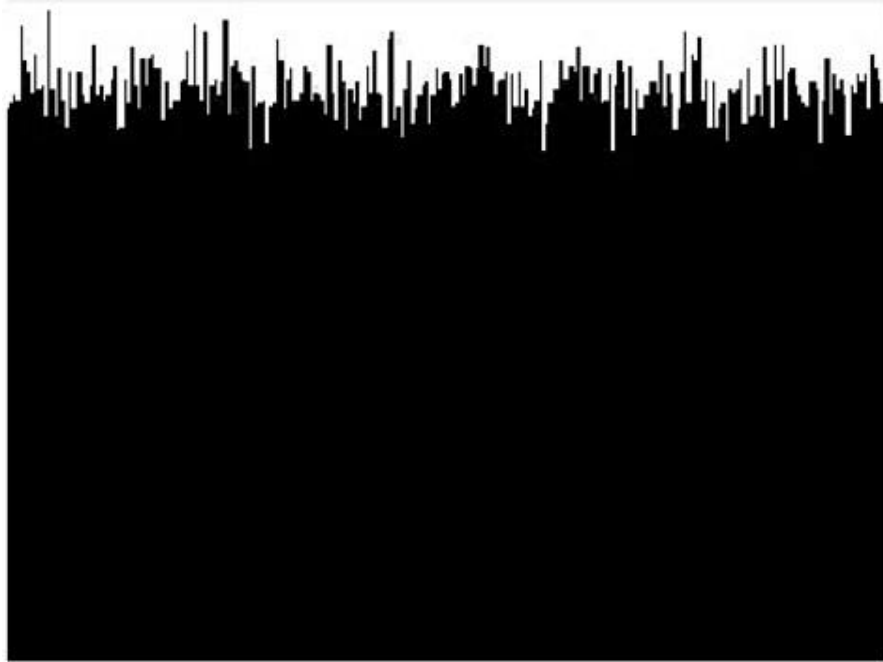


Figure 3: Histogram of the encrypted image in figure 2.

## 5. Conclusions

In the current work, a symmetrical cryptosystem is presented: The modified Triple-DES, which is able to encrypt images in less time than most well-known asymmetric cryptosystems. Additionally, a quality criterion for the encrypted images is given: the  $\chi^2$  goodness-of-fit test is used to determine how evenly spread the histogram of the encrypted image is, along all grey-level values. Then, the reject rates for the null hypothesis are calculated for different levels of type I error. Also, a practical example is given of how the proposed system works.

Regarding the different kinds of attacks the proposed cryptosystem may be subjected to, given that we have an unknown permutation on the input to the third

round and its inverse at the end of the cypher process, the temptation of a linear or differential attack is removed [12,2]. On the other hand, the proposed system has a computational complexity of  $2^{112}$  and there is no computer yet able to try all its keys, such as what happens to DES [7].

Furthermore, it would not be too complicated to extend the proposals of this paper to Advanced-Encryption-Standard [9], since the variable permutation could be interleaved in some of the intermediate rounds.

Finally, even though a considerable number of permutations are used to encrypt an image (i.e. 36478 for the image of the example), the amount of time required for such a process is not prohibitive—it takes approximately 4 seconds for said image—, thanks to the algorithm introduced here to reduce to 1/3 the amount of bit operations used to compute permutations.

**Acknowledgements** The authors would like to thank the Instituto Politécnico Nacional (Secretaría Académica, COFAA, SIP, CIDETEC, CIC, and ESFM), the CONACyT, and SNI for their economic support to develop this work.

## 6. CODE

```
from Crypto.Cipher import DES3

from Crypto.Random import get_random_bytes

from Crypto.Protocol.KDF import PBKDF2

from Crypto.Util.Padding import pad, unpad


def generate_key_from_password(password):

    salt = b'my_salt' # Choose a random salt or derive it based on your use
    case.

    return PBKDF2(password, salt, dkLen=24, count=100000)


def encrypt_image(input_image_path, output_image_path, key):

    try:

        with open(input_image_path, 'rb') as f:

            plaintext = f.read()

            iv = get_random_bytes(DES3.block_size)

            cipher = DES3.new(key, DES3.MODE_CBC, iv)

            ciphertext = iv + cipher.encrypt(pad(plaintext, DES3.block_size))

            with open(output_image_path, 'wb') as f:

                f.write(ciphertext)

            print("Encryption completed. The encrypted image is saved at:",
                output_image_path)

    except Exception as e:
```

```
print("An error occurred during encryption:", str(e))
```

```
def decrypt_image(input_image_path, output_image_path, key):
```

```
    try:
```

```
        with open(input_image_path, 'rb') as f:
```

```
            ciphertext = f.read()
```

```
            iv = ciphertext[:DES3.block_size]
```

```
            cipher = DES3.new(key, DES3.MODE_CBC, iv)
```

```
            decrypted_data = cipher.decrypt(ciphertext[DES3.block_size:])
```

```
            plaintext = unpad(decrypted_data, DES3.block_size)
```

```
            with open(output_image_path, 'wb') as f:
```

```
                f.write(plaintext)
```

```
            print("Decryption completed. The decrypted image is saved at:",
```

```
                  output_image_path)
```

```
        except Exception as e:
```

```
            print("An error occurred during decryption:", str(e))
```

```
def main():
```

```
    choice = input("Enter 'E' for encryption or 'D' for decryption: ").upper()
```

```
    input_image_path = 'path_to_input_image.png'
```

```
    output_image_path = 'path_to_output_image.png'
```

```
    password = input("Enter the password: ").encode() # Get the password from the user
```

```
key = generate_key_from_password(password)

if choice == 'E':

    encrypt_image(input_image_path, output_image_path, key)

elif choice == 'D':

    decrypt_image(output_image_path, input_image_path, key)

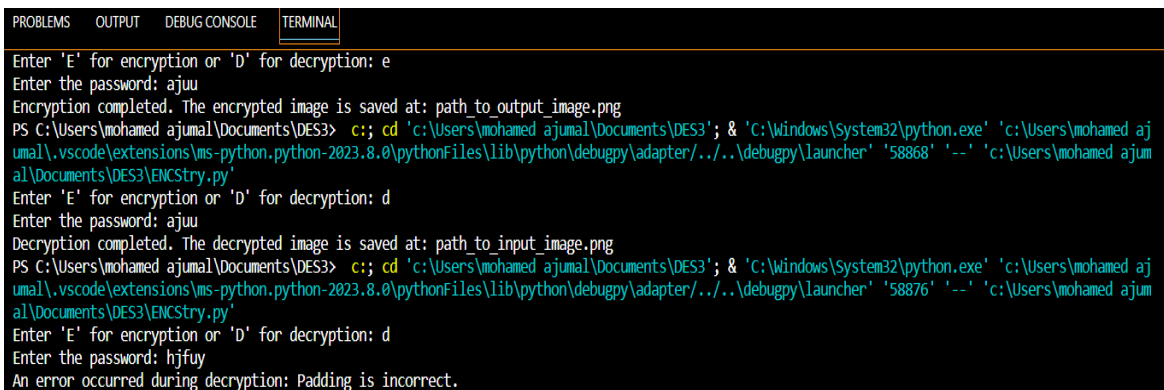
else:

    print("Invalid choice. Please enter 'E' for encryption or 'D' for decryption.")

if __name__ == "__main__":

    main()
```

## 7.OUTPUT



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Enter 'E' for encryption or 'D' for decryption: e
Enter the password: ajuu
Encryption completed. The encrypted image is saved at: path_to_output_image.png
PS C:\Users\mohamed ajumal\Documents\DES3> c:: cd 'c:\Users\mohamed ajumal\Documents\DES3'; & 'C:\Windows\System32\python.exe' 'c:\Users\mohamed ajumal\.vscode\extensions\ms-python.python-2023.8.0\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '58868' '--' 'c:\Users\mohamed ajumal\Documents\DES3\ENCstry.py'
Enter 'E' for encryption or 'D' for decryption: d
Enter the password: ajuu
Decryption completed. The decrypted image is saved at: path_to_input_image.png
PS C:\Users\mohamed ajumal\Documents\DES3> c:: cd 'c:\Users\mohamed ajumal\Documents\DES3'; & 'C:\Windows\System32\python.exe' 'c:\Users\mohamed ajumal\.vscode\extensions\ms-python.python-2023.8.0\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '58876' '--' 'c:\Users\mohamed ajumal\Documents\DES3\ENCstry.py'
Enter 'E' for encryption or 'D' for decryption: d
Enter the password: hjfuy
An error occurred during decryption: Padding is incorrect.
```

## References

- [1] Barker W, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, NIST Special Publication, 2008, 800-67.
- [2] Biham E. and Shamir A., Differential cryptanalysis of the full 16-round DES, Lecturer Notes in computer Science, 1993, 494-502.
- [3] Douglas R. Stinson, CRYPTOGRAPHY: Theory and practice, Chapman & Hall/ CRC Press, 2002, pp. 161-280.
- [4] Feistel H., Cryptography and Computer Privacy, Scientific American, vol. 228, no. 5, 1973.
- [5] FIPS PUB 46-3, Federal Information Processing Standards Publication, 1999.
- [6] Gentle J., Random Number Generation and Monte Carlo Methods, Springer, 2003.
- [7] Grabbe J., Data Encryption Standard: The DES algorithm illustrated, Laissez faire City time, vol. 2, no 28, 2003.
- [8] Grinstead Ch. and Snell L., Introduction to Probability, American Mathematical Society, 1997, pp. 325-360.
- [9] J. Daemen and V. Rijmen, AES Proposal: Rijndael, AES algorithm Submission, FIPS 197, 1999.
- [10] Jian Li and li Gan, Study on Chaotic Cryptosystem for Digital Image Encryption, 2011 Third International Conference Measuring Technology and Mechatronics Automation, IEEE, 2011.
- [11] Koblitz M., A Course in Number Theory and Cryptography, SpringerVerlag, New York Inc., 1987, pp. 10-20.
- [12] Matsui M., Linear Cryptanalysis for DES cipher, Lecture Notes in Computer Science, 1994, 386-397.
- [13] Rosen K., Discrete Mathematics and its Applications, Mc. Graw Hill fifth edition, 2003, pp. 362-370.
- [14] Silva V. et al, Algorithm for Strengthening Some Cryptography Systems, Journal of applied Mathematics and Decision Sciences, Hikari Ltd, 2009, pp. 967-976.
- [15] Wolpe R. and Myers R., Probability and Statistics for Engineers and Scientists, Prentice Hall, 2007.
- [16] Xuemei L. et al, A Novel Scheme Reality Preserving Image Encryption, 2011 Third International Conference Measuring Technology and Mechatronics Automation, IEEE, 2011.