Optimizations of a Cryptographic Method Using Gray Scale Digital Images

M. Y. R. Gadelha, C. F. F. Costa Filho, M. G. F. Costa Technology and Information Center
Federal University of Amazonas, Manaus, Brazil

Abstract— **In a world connected through a myriad of communication media, such as mobile phones and the Internet, it is extremely important that confidential information traveling across public networks reaches its destination with privacy preserved. This paper proposes an optimization of a method for data encryption using images that explore the random spatial distribution of pixel gray levels of an image. For the optimized method test, text messages were created in Portuguese. Although the encryption time of the proposed algorithm is still higher than AES and RSA, the encryption time improved as much as 39% in the tests.**

Keywords – text encryption; image; pixel gray level.

I. INTRODUCTION

Nowadays it is critical that information flows securely across telecommunication networks. Every second a myriad of safe operations must occur and every day new techniques are developed to listen to and modify this information [1,2].

In this context cryptography has been developed to prevent this confidential information from being intercepted by intruders. Information cryptography includes not only file cryptography, but also communication channel cryptography [1].

In this paper, an optimization of the algorithm of cryptography using gray scale digital images is presented [3]. The method describes how to use an image as a cryptography key. The consequences and findings of applying this optimization to the algorithm are also analyzed. The main idea behind this technique is to use the random spatial distribution of pixel gray levels of an image to encrypt one text message. The technique is a kind of symmetric cryptography key: the same image is used for both message encryption and decryption. The optimization proposed consists of creating lookup tables for faster access to the image coordinates, necessary for the encryption. The results of the proposed optimization must be evaluated and compared with the most commonly used techniques of data encryption, both with a symmetric key technique, AES [5], as well as with an asymmetric key technique, RSA [6].

The AES encryption technique is the standard cryptographic pattern adopted nowadays in the United States and was proposed as a replacement for the DES cryptographic technique [7]. In order to achieve high encryption speeds, both with software and hardware implementation, the employed encryption algorithm, the Rijndael, uses permutation and replacement operations [5].

The RSA encryption technique was proposed in 1978 and uses two keys: one for encryption and the other for decryption. The algorithm is based on a technique of factoring large numbers [6].

This paper is organized according to the following sequence: section 2 presents the proposed optimization for the cryptography method; section 3 describes the results of applying this optimized algorithm with data encryption and section 4 provides a summary of this paper, presenting a conclusion and describes future research on the same theme.

II. CRYPTOGRAPHY METHOD

The following assumptions underlie the proposed cryptography method:

1. The intensity of a pixel in a gray scale digital image can usually assume 256 possible values ranging from 0 (low intensity pixel) to 255 (high intensity pixel).
2. The ASCII table used to represent all characters of a text can assume values ranging from 0 to 255 [4].

Based on this similarity of range values that exists between the pixel intensities of a gray scale digital image and the character representation using the ASCII table values, the algorithm encrypts from one text character with a random (x,y) coordinate pair of an image. In this coordinate pair, there is a pixel with intensity value equal to the character ASCII value. The optimization consists of creating lookup tables for faster retrieval of the coordinates to improve the encryption speed for some special cases.

A. Encryption Process

The basic encryption algorithm is described in Fig. 1. This algorithm requires an image that is used as a cryptography key. Hereafter, this image is called a key-image. At the end of the encryption process, the output encrypted message, msgCrypt, is a set of coordinate pairs. As an example of an application of this algorithm, suppose we need to encrypt an original message composed of the characters “ab”. The key-image used in this example is shown in Fig. 2(a).

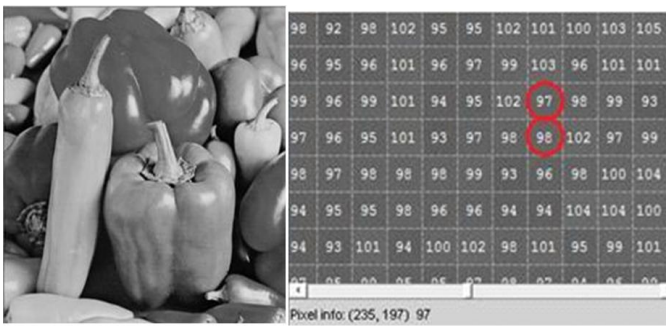
During the encryption process, the algorithm verifies each character value. The values that represent the characters “a” and “b” in the ASCII table are “97” and “98”, respectively.

For the "97" value the encryption algorithm then searches in the image of (Fig. 2a) pixels for equal intensity values. According to the encryption algorithm, a *list* with coordinate pairs of pixels is provided with intensity value equal to "97". A random position *k* is then chosen from this list and the coordinate pair of this position is used to represent the value "97" in the message *msgCrypt*. A similar procedure is used to represent value "98". Suppose that the coordinate pairs used to represent values "97" and "98" are those highlighted by circles in Fig. 2b. The ordered set of values in the encrypted message *msgCrypt* is shown in the following vector:

*msgCrypt*_vector: [235,197, 235, 198]

1. Read the original message msg from a file
2. Read key-image *img*
3. Initiate encrypted message *msgCrypt* as null
4. **For** each character *c* in *msg* **do**
   a. Return a *list* of coordinate pairs *(x,y)* where *img(x,y) = c*
   b. Randomly choose a position *k* in *list*
   c. Add at the end of *msgCrypt* the coordinate pair of the position *list(k)*
5. **End for**
6. Save *msgCrypt* to a file

Figure 1. Encryption algorithm 1



(a) (b)

Figure 2. (a) Image used as the cryptography key; (b) Intensity values ramdomly chosen to encrypt the message "ab" (the coordinate pair of value 97 is shown in the lower left side.)

When key-images are used in the cryptography process, two consequences must be noted: the first is the increase in the randomness of the encrypted file, because an intensity value could be found in any coordinate within an image; the second one is that the size of the encrypted file increases compared with the original message file. The encrypted file is normally eight times higher than the original message file. The reason for this is that in the encryption process, a coordinate pair is generated in the encrypted file for each character (one byte of information). If each coordinate is represented by an integer number of 4 bytes, the coordinate pair occupies 8 bytes (4 bytes for *x* coordinate and 4 bytes for the *y* coordinate).

To minimize this problem, a lower number of bytes could be used to represent each coordinate. Table 1 shows some representations for integer type - *int*. As shown, the integer type could represent extremely high numbers that probably are not used to represent an image coordinate. Otherwise, some integer types could assume negative values that must never be used to represent an image coordinate. So, the original encryption algorithm can be modified in order to use the lowest number of bits to represent a coordinate, depending on the size of the image used as the cryptography key. For example, if the key-image is 511x511 pixels, the encryption algorithm must choose an unsigned integer representation with 9 bits, as shown in Table 2.

TABLE I. DIFFERENT INTEGER TYPES

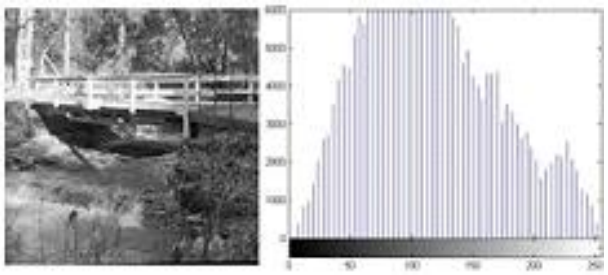
Integer Type	bits (bytes)	Value intervals	
		Low value	High value
int	32 (4)	-2147483648	2147483647
unsigned int	32 (4)	0	4294967295
short int	16 (2)	-32768	32767
unsigned short int	16 (2)	0	65535
n-bit int	n (n/8)	-2^{n-1}	$2^{n-1}-1$
unsigned n-bit int	n (n/8)	0	$2^{n}-1$

TABLE II. UNSIGNED INTEGER BETWEEN 9 AND 15 BITS

Integer type	Value intervals	
	Low value	High value
unsigned 09-bits int	0	511
unsigned 10-bits int	0	1023
unsigned 11-bits int	0	2047
unsigned 12-bits int	0	4095
unsigned 13-bits int	0	8191
unsigned 14-bits int	0	16383
unsigned 15-bits int	0	32767

Two advantages of this intelligent choice of the integer type used in the encryption algorithm can be cited: the first one is that the size of the encrypted file will be reduced, because an 8-bit character is encrypted using fewer bits (considering a dimension of 511 pixels, 9 bits for each coordinate); the second one is that it would be more difficult for an intruder to decrypt the encrypted message with no information of dimensions of the image used as the cryptography key.

A problem with the encryption algorithm will occur if an ASCII value of a character read from the original message does not correspond to one-pixel intensity in the key-image. An example illustrates this situation. Suppose that an encryption process uses the key-image shown in Fig. 3a. The corresponding key-image histogram is shown in Fig. 3b. Investigating this histogram, it is noted that at some intensities of the horizontal axis (range 0-255) there are no vertical bars, suggesting that no pixel with these intensities can be found in the image of Fig. 3a. If the ASCII value of a character in the original message corresponds to one of these intensities, this paper suggests that this ASCII value be represented by the nearest pixel intensity found in the key-image and, to signal the insertion of these approximated value, *flags* must be used. A complete description of how the encryption algorithm carries out this procedure is provided below.



(a) (b)

Figure 3. (a) Example of a Gray scale image used as a cryptography key; (b) Corresponding histogram of image of (a).

At the beginning, the algorithm verifies whether there is at least one pixel in the key-image whose intensity corresponds to the ASCII value of the first character in the original message. If there is, the algorithm inserts a *flag* 0 in the encrypted file. If not, the algorithm inserts a *flag 1* in the same file.

If the first scenario occurs, after the insertion of the *flag 0*, the algorithm inserts the coordinate pair of the pixel. In sequence the next character of the original message is verified. Again, if there is a pixel whose intensity value corresponds to the character ASCII value, the coordinate pair of this pixel is inserted in the encrypted file. This procedure is repeated until a character whose ASCII value does not correspond to any pixel intensity is found. If this occurs, a *stop flag* is inserted in the encrypted message. The stop flag value is defined as the sum of the larger image dimension plus one. After the insertion of the *stop flag*, *flag 1* is inserted to signal that an approximate value will be inserted in the sequence. After finding the nearest pixel intensity to represent the character, a token is inserted to signal if the pixel intensity is less or higher than the character ASCII value. In the sequence, a number is inserted corresponding to how many units the pixel intensity is less or higher the ASCII value. Finally these elements are inserted: another *stop flag*, *flag 0* and the coordinate pair of the pixel whose value is the nearest one to the character ASCII value.

To illustrate this procedure suppose that we need to encrypt an original message described by the vector [a d h]. This vector corresponds to the ASCII vector [97 100 104]. It will be assumed also that there are no pixels in the key-image key with intensities ranging from 103 to 107. Supposing that random coordinate pairs were found according to the procedure just described, the encrypted message is represented by the following vector (flags are represented in boldface):

*msgCrypt*_vector: [**0** 243 412 283 398 **513 1** 45 2 **513 0** 274 117]

As noted, the encryption algorithm first inserted a *flag 0* and filled the encrypted message with two coordinate pairs. After, a *stop flag* was inserted. The key-image dimension is supposed be 512x512 pixels, so the *stop flag* value is 513. In sequence a *flag 1* was inserted to signal that next encrypted coordinate pair represents an approximate value. In this example, token 45(43) was used to signal that the nearest pixel intensity found is less (higher) than the character ASCII value. These values were chosen because, in the ASCII table, they represent the arithmetic signals "-" and "+". So, value 45 is inserted to signal that the pixel intensity whose coordinates will be inserted is less than the ASCII value of the third original message character. Next, value 2 is inserted, because the nearest intensity pixel found was 102, two units less than the ASCII value 104, which represents the message's last character. Finally, *flag 0* and the coordinate pair of the nearest pixel value are inserted.

If the second option occurs, after the insertion of *flag 1*, the following insertions are made at the beginning of the encrypted message: a token to signal that the pixel intensity value is less or higher than the character ASCII value, a number corresponding to how many units the pixel intensity is less or higher the ASCII value, *stop flag* and *flag 0*. To illustrate this situation, suppose that the message just encrypted be changed to [h a d]. Supposing the same previous hypothesis, the encrypted message is represented by the following vector:

*msgCrypt*_vector: [**1** 45 **2 513 0** 274 117 243 412 283 398]

The new encryption algorithm, encryption algorithm 2, taking into account the procedure just described, is shown in Fig. 4.

B. Decryption Process

In the decryption process, the same key-image of the encryption process is used. Taking as the starting point the encrypted message, the decryption algorithm decrypts each coordinate pair, to generate each one of the characters of the original message. At the end of the process, the decrypted message is equal to the original one. The decryption algorithm is shown in Fig. 5.

III. OPTIMIZATIONS ON THE ALGORITHM

The proposed optimization of the cryptographic algorithm has the objective of decreasing the encryption time of a message when there are missing ASCII values for the image. The proposed optimization changes the algorithm so it does not represent the value using the nearest value, but uses any other value presented for the image instead.

To implement this optimization, the following changes were made to the algorithm:

- In the beginning of the algorithm, another list is created, along with the list of coordinates: it will contain all the valid ASCII values for the image.
- The list is accessed when the algorithm finds a missing ASCII value in the message. When this happens, a random value is chosen from the list of valid ASCII values and this value is used instead of the nearest value.

This approach brings two main advantages over the original algorithm, which always searched for the nearest valid pixel. The first is speed, the algorithm is faster because it does not have to search for the nearest valid ASCII value. With the proposed optimization, the algorithm only has to randomly choose one entry in a list, which will return a valid ASCII

value and the algorithm can carry on calculating distance and choosing coordinates.

Secondly, there is an increase of randomness in the encrypted message, because if the algorithm always chooses the nearest ASCII value and a missing value occurs more often in the message, the same nearest ASCII value will be chosen. For example, let us suppose that an image only has values between 0 and 200. All the values over 200 will be replaced with 200, the nearest valid ASCII value, and the coordinates with value 200 will be repeated many times in the encrypted message.

The new encryption algorithm is shown in Fig. 6. The decryption algorithm remains unchanged.

1. Read original message file *msg*
2. Read key image *img*
3. Calculate the maximum number of bits *n* needed to represent *img* coordinates
4. Calculate stop flag value as the maximum dimension of *img* plus one.
5. Initiate encrypted message *msgCrypt as a null unsigned n integer file*.
6. **For** each character *c* in *msg* **do**
    a. Return a *list* of coordinate pairs *(x,y)* where *img(x,y) = c*
    b. **If** *list* is different from *null*
        i. Choose a random position *k* in *list*
        ii. *encryptedChar = list(k)*
    c. **If not**
        i. *temp* = [1 (43/45) *dist stop flag* 0]
        ii. Return a *list* of coordinate pairs *(x,y)* where *img(x,y) = c (+/-) dist*
        iii. Choose a random position *k* in *list*
        iv. *encryptedChar = temp* concatenated with *list(k)*
    d. End if
    e. **If** size of *encryptedChar* == 2
        i. *encryptedChar = flag 0* concatenated with *encryptedChar*.
    f. **Else if** size of *encryptedChar* > 2 e *msgCript* is not empty
        i. *encryptedChar = stop flag* concatenated with *encryptedChar*
    g. End if
    h. Add *encryptedChar* to the end of *msgCript*
7. **End For**
8. Save *msgCript*

Figure 4. Encryption algorithm.

1. Read key image *img*
2. Calculate the maximum number of bis *n* needed to represent the *img* coordinates.
3. Read encrypted message *msgCrypt* using *n* bits for each coordinate.
4. Calculate *stop flag* as the maximum img dimension added plus one.
5. Initiate the decrypted message *msgDecrypt* as null.
6. Initiate *hasOp* as *false*
7. Initiate i = 1
8. **While** i < size of encrypted image **do**
    a. **If** msgCrypt(i) == 0
        i. i = i + 1
        ii. **While** *msgCrypt*(i) ≠ *stop flag* **do**
            1. *x = msgCrypt*(i)
            2. *y = msgCrypt*(i+1)
            3. *decryptedChar* = value of pixel *img(x,y)*
            4. **If** *hasOp == true*
                a. *decryptedChar = decryptedChar + corr*
                b. *hasOp = false*
            5. **End if**
            6. *msgDecrypt = decryptedChar* concatenated with *msgDecrypt*
            7. i = i + 2
            8. **If** i > size of *msgCrypt*
                a. *Break*
            9. **End if**
    b. **Else if** msgCrypt(i) == 1
        i. i = i + 1
        ii. *hasOp = true*
        iii. **If** *msgCrypt*(i) == 43
            1. *corr* = msgCrypt(i+1)*(-1)
        iv. **Else if** *msgCrypt*(i) == 45
            1. *corr* = msgCrypt(i+1)
        v. **End if**
        vi. i = i + 2
    c. **End if**
    d. i = i + 1
9. **End while**
10. Save *msgDecrypt*

Figure 5. Decryption algorithm.

IV. RESULTS

All algorithms presented in this paper were implemented using the MATLAB® program. Although slower than a program in C or C++ language, the MATLAB® program is favored when working with different image formats, because it automatically does the header parser for the user.

As the proposed optimization is intended to improve the encryption algorithm, the experiments will evaluate the original encryption algorithm and the optimized algorithm.

The experiments are divided into three groups. In the first group, 50 messages were randomly generated with size varying between 9,000 and 10,000 characters.

In the second group, the book of Genesis of the Christian Bible was employed. This book contains 187,071 characters in Portuguese.

In the last group, also employing the book of Genesis, comparative tests were made of the proposed algorithm with the original algorithm, AES and RSA algorithms.

The experiments used ten key-images of a public site [8]. Key-image details are shown in Table 3. All experiments were done in a computer with the following characteristics: Intel® Core 2 Duo 2.20 GHz, 4GB RAM, Windows 7 Home Premium 32 bits. Table 4 shows the results obtained in the first experiment group, while Table 5 shows the results obtained in the second experiment group.

1. Read original message file *msg*
2. Read key image *img*
3. Calculate the maximum number of bits *n* needed to represent *img* coordinates
4. Calculate stop flag value as the maximum dimension of *img* plus one.
5. Initiate encrypted message *msgCrypt as a null unsigned n integer file*.
6. Create listNonNull of valid ASCII values.
7. **For** each character *c* in *msg* **do**
    a. Return a *list* of coordinate pairs *(x,y)* where *img(x,y) = c*
    b. **If** *list* is different from *null*
        i. Choose a random position *k* in *list*
        ii. *encryptedChar = list(k)*
    c. **If not**
        i. Randomly choose a position in ListNonNull.
        ii. *temp* = [1 (43/45) *dist stop flag* 0]
        iii. Return a *list* of coordinate pairs *(x,y)* where *img(x,y) = c (+/-) dist*
        iv. Choose a random position *k* in *list*
        v. *encryptedChar = temp* concatenated with *list(k)*
    d. End if
    e. **If** size of *encryptedChar* == 2
        i. *encryptedChar = flag 0* concatenated with *encryptedChar*.
    f. **Else if** size of *encryptedChar* > 2 and *msgCript* is not empty
        i. *encryptedChar = stop flag* concatenated with *encryptedChar*
    g. End if
    h. Add *encryptedChar* to the end of *msgCript*
8. **End For**
9. Save *msgCript*

Figure 4. Optimized encryption algorithm.

TABLE III. SPECIFICATIONS OF THE KEY-IMAGES USED IN THE EXPERIMENTS

Key-image	Dimensions (pixels)	Size (bytes)
aerial.pgm	512x512	257K
boats.pgm	576x720	406K
bridge.pgm	512x512	257K
D108.pgm	640x640	401K
f16.pgm	512x512	257K
girl.pgm	576x720	406K
Lena.jpg	512x512	43K
peppers.pgm	512x512	257K
pp1209.pgm	512x512	257K
zelda.pgm	576x720	406K

As shown in these tables, the worst situation occurs when there are some empty values in the key-image (when the key-image does not have pixels with some intensity values). Arising from this situation two problems can occur. The first one is that the encryption algorithm becomes slower, because it spends so much time searching for the nearest pixel intensity. The second one is that the encrypted message size increases, because some extra flags are introduced (not shown in the tables). The optimized algorithm tries to improve the first situation, decreasing the encryption time. As shown in Table 5, the worst result occurs with the key-image pp1209.pgm, the encryption time is 1418.35 seconds (roughly 24 minutes) using the original algorithm, and 964.61 seconds (roughly 16 minutes) using the optimized algorithm, an 32% decrease in the encryption time. Overall, the optimized algorithm achieved 39.60% improvement in the first group and 9.62% improvement on the second group, over the original algorithm.

TABLE IV. RESULTS OBTAINED FROM GROUP 1 EXPERIMENT

key-image	Encryption time original(s)	Encryption time optimized (s)
aerial.pgm	2.8	2.3
boats.pgm	7.5	3.2
bridge.pgm	8.3	6.1
D108.pgm	46.0	33.0
f16.pgm	9.5	3.2
girl.pgm	5.8	3.3
Lena.jpg	1.65	1.55
peppers.pgm	3.0	1.9
pp1209.pgm	75.0	35.7
zelda.pgm	6.5	2.5

The third group comparisons were made with the best results of the proposed method shown in Table 5, the encryption process was obtained with the 7th key image. The results for this group are shown in Table 6. These results show that the AES and RSA methods are faster than the proposed algorithms (the original and the optimized). We can also notice that the optimized algorithm shows little improvement over the original algorithm in such cases.

TABLE V. RESULTS OBTAINED FROM GROUP 2 EXPERIMENT

key-image	Encryption Time original (s)	Encryption Time optimized (s)
aerial	78.80	78.52
boats	112.06	99.34
bridge	317.86	289.12
D108	763.47	686.54
f16	199.06	176.59
girl	100.93	94.94
lena	62.89	62.12
peppers	68.94	67.68
pp1209	1418.35	964.61
zelda	90.34	78.54

The AES and RSA algorithms are fast because they use non-linear substitution and transposition ciphers, respectively, that replace the original characters by other characters and shuffle the message. Otherwise, in these methods, it is not necessary to make random decisions about table positions and search for nearest values of pixel intensities that are used in the proposed method. Nevertheless, the original algorithm and the

optimized algorithm have an advantage over the two compared algorithms. Regardless of the number of times the AES and RSA algorithms are used to encrypt the same original message, they generate the same encrypted message. The same does not occur with the proposed algorithm. Each time the proposed algorithm and the optimized algorithm are used to encrypt the same original message, a different encrypted message is generated.

TABLE VI. RESULTS OBTAINED IN GROUP 3 EXPERIMENT

Algorithm	Encryption time (s)
AES	0.02
RSA	3.02
Original Algorithm	62.89
Optimized Algorithm	62.12

The improvement of the original algorithm over the original algorithm is small, because the best result from the second group was used in this group of experiment. The optimized algorithm improves the encryption time when the message has several missing ASCII values, a situation not presented on the best result of the second group of experiments.

V. DISCUSSION AND CONCLUSIONS

The main purpose of this paper was to propose an optimization over the cryptography method that uses an image as a cryptography key. To accomplish this task the use of a list of valid ASCII values is created in the beginning of the algorithm and it is used when we need to find a valid ASCII value to replace a missing one.

Ten key-images, each with different characteristics, were used in the experiments of the results section. Some of them have gray levels throughout the entire range 0-255 (high contrast images), while others have low peaks spaced in the histogram.

The experiments used random messages generated by a test program and the book Genesis of the Christian Bible. These messages were tested using both the original algorithm and the optimized algorithm and the results, compared.

The original method and the optimized, nevertheless, has a clear advantage of generating a different encrypted message when the same original message is encrypted. A comparison with AES and RSA algorithms show a better performance of these algorithms concerning encryption time. The optimized algorithm, nevertheless, show improvements in the encryption time over the original algorithm.

The main reason for a slower encryption time is that, when the algorithm must encrypt an ASCII value with no corresponding pixel intensity in the key-image, it has to search for a pixel with the nearest intensity value. This search time negatively impacts the encryption time. The solution proposed is, before the encryption starts, to create a list of valid ASCII values and choose a random value of this list when the algorithm needs to replace a missing ASCII value.

REFERENCES

[1] A. Menezes, P. V. Oorschot and S. Vantone, Handbook of Applied Cryptography, 1st ed., CRC Press, 2001.

[2] A. S. Tanenbaum, Computer Networks, 4th ed., Prentice Hall, 2008.

[3] M. Y. R. Gadelha, C. F. F. Costa Filho and M. G. F. Costa. Proposal of a Cryptography Method Using Gray Scale Digital Images. ICITST, 2012.

[4] ISO/IEC. ISO/IEC 8859-1: Latin Alphabet No. 1, 1997.

[5] National Institute of Standards and Technology. Advanced Encryption Standard, november, 2001.

[6] R. L. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, vol. 21, pp. 120-126, February 1978.

[7] National Institute Of Standards And Technology. Data Encryption Standard, October 1999.

[8] Mikhail Ramalho. Criptography using Images. http://gitorious.org/cryptography-using-images. (Access date: 25/9/2012)