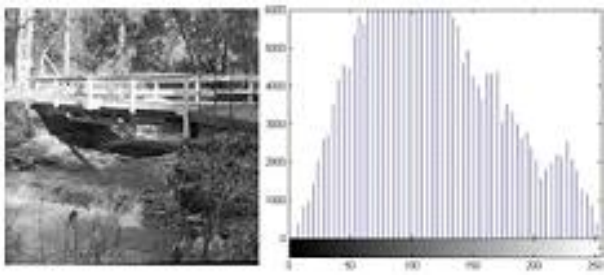

(a) (b)

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)