Original Research Paper

A Key Dependent Encryption Algorithm Based on Multiple Bitwise-Shuffling and XOR Variable-Length Partitions

Abdelfatah Aref Tamimi

Department of Computer Science, Al-Zaytoonah University of Jordan, P.O. Box 130, Amman 11733, Jordan

Article history
Received: 26-11-2015
Revised: 21-03-2016
Accepted: 26-03-2016

Email: drtamimi@zuj.edu.jo

Abstract: This new algorithm employs shuffling procedures combined with variable-length key-dependent XOR and S-box substitutions to perform lossless image encryption. This algorithm was implemented and tested by performing different permutations of shuffling, XOR encryption and S-box substitution. Empirical analysis using different types of test images of different sizes showed that this new algorithm is effective and resistant to statistical attacks. The idea presented by this algorithm may be generalized to apply to input data other than images and may be combined with other encryption methods.

Keywords: Shuffling, Cryptography, Stream Cipher, Partition Cipher, S-Box

Introduction

Cryptography is an essential field of computer security used for protecting the confidentiality, integrity and availability of information. Wang and Wang (2012) have introduced an image encryption algorithm based on couple multiple chaotic systems. Security analysis showed that the key space of this algorithm is large enough to make brute-force attacks infeasible. The simulation results also showed that the scheme has high performance, so it has a potential value in the field of image encryption and image transmission.

Kaipa *et al.* (2014) used linear Eigen values and several mathematical operations to introduce nonlinearity to the linear transformation-based cryptosystem using byte substitution over GF(2^8) and variable length sub-key groups. They also conducted performance evaluation of the method.

Ali and Makhzoum (2012) increased the efficiency of the decryption process compared to an existing public key Luc cryptosystems algorithm by using Divide-By-Prime computation.

Do and Song (2014) formed a protocol capable of recovering the encrypted streaming media data only if a key is shared and stored using All or Nothing Transform (AONT). It was based on XOR threshold Secret Sharing where the user acquires the Recovery Share (Privilege Manager Group) and XOR Share (User) to be distributed by Data Owner. Moreover, collusion attacks are avoided by realizing the management of access privileges and distribution of decryption privileges using a Privilege Manager Group.

Chengqing and Liu (2013) showed that, based on some properties of a composite function composed of modulo addition and the XOR operation, a known-plaintext attack and an improved chosen-plaintext attack can be provided to determine an equivalent secret key. The cryptanalysis highlighted attack vulnerability in some encryption schemes based on multiple combination of modulo addition and XOR operations.

Backes and Pfitzmann (2008) presented a Dolev-Yao model with XOR with a cryptographic realization secure against passive attacks if the surrounding protocol additionally guarantees that no incorrect conversion of XORs back into other types attempted, except for the restrictions on passive attacks and correct type conversions.

A bitwise XOR operation is normally used as a part of a more complex encryption algorithm. Numerous variations of the use of XOR in image encryption can be found in the literature. In the Advanced Encryption Standard (AES), which was adopted by many official and commercial organizations worldwide to encrypt sensitive data of various formats, XOR is used as a step in the encryption procedure for effectively combining data being encrypted with the encryption key. Al-Husainy (2012) designed an algorithm that combines XOR encryption with a rotation operation for effective image encryption. Nag *et al.* (2011) used an affine transform combined with XOR encryption to perform image encryption. Chatzichristofis *et al.* (2014) effectively encrypted images using the recursive attributes of the XOR filter.

Examples on applying the four steps of AES, including the use of S-box substitution, are available.

Many encryption algorithms based on AES were also developed. However, El-Fishawy and Abu Zaid (2007) and Sivakumar and Venkatesan (2014) and has shown that AES has limitations on some image and multimedia specific requirements, so other encryption algorithms need to be developed.

A new algorithm is presented, which performs lossless encryption in three steps. These steps are shuffling, partitioning with XOR groups operation and S-box substitution. The first step performs shuffling on the image as a stream. The second step partitions the result into variable length blocks of key-dependent sizes and performs XOR encryption on these blocks. The third and final step performs byte substitution using a lookup table. The algorithm was implemented and different combinations of these steps were tested. Analysis of these steps showed different effectiveness of the cipher with different combinations of these steps; where using all three steps produced the most secure encryption of the different combinations.

The New Algorithm

This algorithm takes an image and a key as input and it works in three steps as follows. It starts with bitwise shuffling of the image stream. Then, it partitions the resulting image into variable size blocks with key-dependent sizes, applying a checksum technique to each block to produce a new code. Finally, it applies byte substitution using a lookup table called S-box. The encryption and decryption steps of the algorithm are illustrated in Fig. 1, where the three encryption steps are independent. Any combination of these three steps may be performed, where the decryption performs the inverse of the applied steps in reverse order.

Bitwise Shuffling

In the first step of the algorithm, the image is regarded as a stream of bytes and the encryption performed is both key dependent and data dependent. One specific bit, call it *bitLoc*, is chosen by a key-dependent function. A shuffle vector is constructed by listing the numbers of bytes which have the value of bit number *bitLoc* equal to one, followed by the numbers of bytes which have the value of bit number *bitLoc* equal to zero. This vector gives the new locations of the input bytes; which are numbered left to right. This step is repeated for several iterations. Each iteration uses a different *bitLoc* and applies the same steps to the new image that resulted from the preceding iteration. Let the bytes of an ($n \times m$)-byte image be numbered left to right, row by row, starting with number ($j = 0$) for the first byte of the first row and ending with number ($j = nm$-1) for the last byte of the last row. The encryption algorithm using bitwise shuffling is as in the algorithm designed by Yahya and Abdalla (2008; 2009).

The Bitwise Shuffle step uses a key-dependent function to specify the location of the shuffle bit (*bitLoc*), where $0 \leq bitLoc \leq 7$. It runs for a given number of iterations; *k*. Then, a shuffle vector is constructed by listing the numbers of bytes with the value of bit number *bitLoc* equal to one, followed by the numbers of bytes with the value of bit number *bitLoc* equal to zero. This vector gives a mapping that specifies the new location of each byte in the image. In each one of the subsequent iterations, a different *bitLoc* is chosen and the same steps are applied to the image that resulted from the preceding iteration. Note that the value of *bitLoc* can be represented by three bits, which requires a total length to $3k$ bits to represent all values of *bitLoc*. The algorithm outlining this step is given below.

```
For i = 0 to k-1
    bitLoc = Key[i]
    D = Vector where D[j] is the value of bit (bitLoc) of
        the j^th byte of the current image
    S0 = Vector containing numbers of current image
        bytes (j) that have (D[j] = = 0)
    S1 = Vector containing numbers of current image
        bytes (j) that have (D[j] = = 1)
    Shuffle = Concatenation of S1 with S0
    Substitute the bytes of current image so that the new
        location of byte (j) is byte (Shuffle[j])
    Replace bit (bitLoc) of the j^th byte with D[j] in each
        byte of current image
End For
```

Stream Partition

In the stream partition step, including the XOR encryption operation, the image is regarded as a stream of bytes and then it is divided into groups (one-dimensional blocks). Let the input image have *n* bytes and let the key have *b* bytes referred to as *key*[0] through *key*[*b*-1]. The image is divided into approximately $n / \sum_{i=0}^{b-1} key[i]$ groups of bytes. Group number *j* will consist of *key*[*i*] bytes where $j = (b \times c + i)$ for some non-negative integer *c*.

For example, with a key of 16 bytes where the value of its *key*[5] = 70, there will be groups in the image consisting of 70 bytes each, namely: The groups numbered 5, 21, 37, 53, 69, *etc*.

XOR Groups Operation

Each of the above groups is encrypted with XOR as follows. Suppose the bytes of group number *j* are *G*[0] through *G*[*key*[*i*]-1]. Then, the encrypted values will be:

$$\begin{aligned} &G'[0] = \left(G[0] \text{ XOR } key[i]\right) \textit{ and} \\ &G'[p] = \left(G[p] \text{ XOR } G'[p-1]\right) \text{ for } 0 < p \leq key[i]\text{-1} \end{aligned} \quad (1)$$

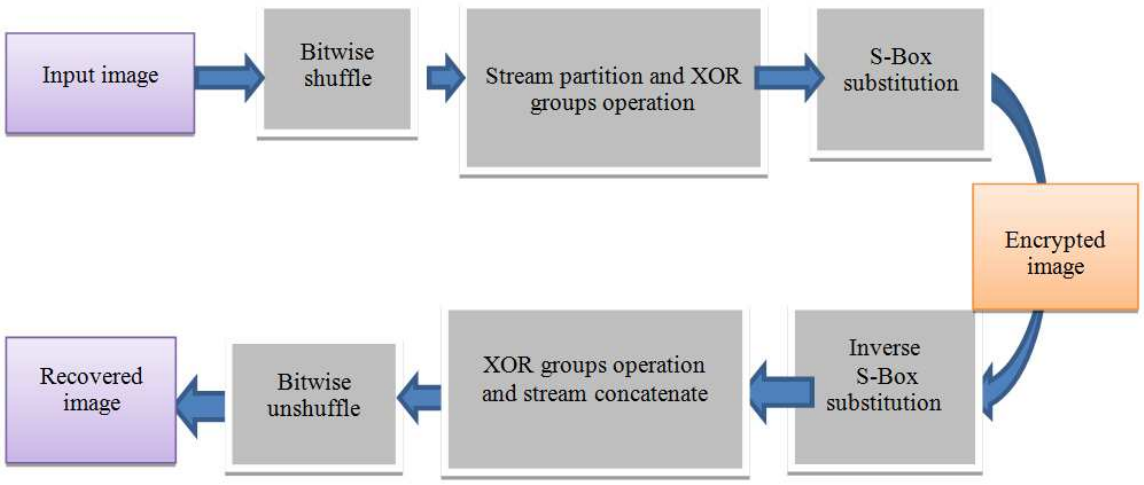
Input image
Bitwise shuffle
Stream partition and XOR groups operation
S-Box substitution
Encrypted image
Inverse S-Box substitution
XOR groups operation and stream concatenate
Bitwise unshuffle
Recovered image

Fig. 1. Diagram showing the steps of the algorithm

Each group is encrypted similarly but independently of the other groups. This is the same techniques used in Internet checksum but with variable group sizes.

S-Box Substitution

The two-dimensional substitution table, known as S-box, is constructed to perform two transformations: multiplicative inverse and affine transformation. This nonlinear key-dependent substitution was presented as a single step in each iteration of the AES algorithm. However, in the new algorithm presented here, this substitution is performed at most two times. It is either applied before, after, or both before and after the XOR encryption operation. It is applied to the entire image; block by block. The S-box substitution in this algorithm may also be skipped if needed. If S-box substitution is skipped, another substitution or shuffling operation should be applied in addition to the XOR encryption, so that no encrypted group will remain intact or in the same location inside the image as produced by the XOR encryption operation.

The decryption algorithm is similar to the encryption algorithm, where each of the above steps can be easily inverted. This decryption restores the original image without any loss.

Example

Let the input be: (234, 124, 29, 35, 245, 44, 189, 222) with *key* = (3, 5). This input is represented by the binary numbers in the first column of Table 1. Suppose the first step; Bitwise Shuffle, runs for 3 iterations and specifies the locations of the shuffle bit (*bitLoc*) to be 0, 4 and 3. In the first iteration, *bitLoc* = 0. Based on the values of this *bitLoc*, *D* = 00111010, *S0* = (0, 1, 5, 7) and *S1* = (2, 3, 4, 6), which make the shuffle vector (2, 3, 4, 6, 0, 1, 5, 7). This vector gives the new locations of the input bytes; which are numbered left to right. Then, the input after the shuffle substitution will be as shown in the second column of Table 1. Now, *D* is used for replacing each bit (*bitLoc*) to give the result shown in the third column of Table 1. This result is used by the second iteration, where *bitLoc* = 4. Now, *D* = 10110101, *S0* = (1, 4, 6) and *S1* = (0, 2, 3, 5, 7), which make the shuffle vector (0, 2, 3, 5, 7, 1, 4, 6) and its result is shown in the fourth column of Table 1. Placing the bits from *D* into this result gives the result shown in the fifth column of Table 1. Similarly, the third iteration has *bitLoc* = 3, producing *D* = 10111011 resulting in the sixth and seventh columns of Table 1. The decimal representation of this result is (28, 181, 124, 206, 235, 53, 237, 58).

After that, stream partition is performed with XOR operations. To demonstrate this step for this simple example, let *key*[0] = 3 and *key*[1] = 5, so the 8-byte input is partitioned into two group; one with 3 bytes and the other with the remaining 5 bytes. Performing the XOR groups operation gives the result (31, 5, 238, 203, 5, 238, 219, 54). Finally, S-box substitution is performed producing a result dependent on the values of S-box, such as (192, 107, 40, 31, 107, 40, 185, 5).

The inverse of this example starts with this result and works back to extract the original input as follows. First, Inverse S-box is applied to this result to obtain the vector: (31, 5, 238, 203, 5, 238, 219, 54). Then, XOR groups operation and concatenation produce the vector: (28, 181, 124, 206, 235, 53, 237, 58). After that, the vector *D* = 10111011 is extracted from bit number 5, which is the *bitLoc* taken from the *key*. This gives *S0, S1* and shuffle vector as obtained in the last (i.e., third) iteration of the Bitwise Shuffle step of the encryption. The unshuffle operation is performed and then the values of bit 5 in each byte is replaced with its corresponding value from *D* restoring the result of the second Bitwise Shuffle iteration. Similarly, the result from the first Bitwise Shuffle iteration is restored and then the original input vector is produced in the same manner.

Table 1. Example data obtained by applying Bit Shuffling

Original data	1st Iteration (*bitLoc* = 0)	1st Replace D	2nd Iteration (*bitLoc* = 4)	2nd Replace D	3rd Iteration (*bitLoc* = 3)	3rd Replace D
11101010	00011101	00011100	00011100	00011100	00011100	00011100
01111100	00100011	00100010	11110101	11100101	10111101	10110101
00011101	11110101	11110101	10111101	10111101	01111100	01111100
00100011	10111101	10111101	01111100	01111100	11001110	11001110
11110101	11101010	11101011	11011110	11001110	11101011	11101011
00101100	01111100	01111100	00100010	00110010	00111101	00110101
10111101	00101100	00101101	11101011	11101011	11100101	11101101
11011110	11011110	11011110	00101101	00111101	00110010	00111010

Implementation and Analysis

The security of the new algorithm comes from combining the three encryption operations; using bitwise shuffling, XOR encryption and S-box byte substitution. If XOR encryption is used alone, the encryption may become vulnerable to brute-force and plaintext attacks. Using S-box substitution alone could make the encryption vulnerable to statistical attacks. A combination of these three encryption operations will provide significant resistance to all of these types of attacks.

If one or more bits in the key are changed, it causes a different grouping in the XOR step and the XOR values are changed. In addition, let an S-box of size 16×16 bytes be used in the S-box substitution step. This S-box has 2,048 different entries where each of these entries consists of 8 bits. This makes the total number of permutations for this step is 2^{11}. Consequently, for an image of ten or more kilobytes input to any combination of these two encryption operations, a brute-force attack is impossible.

The algorithm was applied to 50 images of various types and sizes ranging from 2 to 30 kilobytes (kB). When different combinations were used with the same image, they produced different encrypted images. In addition, analysis using histograms, correlation and Peak Signal to Noise Ratio (PSNR) showed properties of the algorithm that strongly resist statistical attacks.

As seen in Fig. 2, the histograms of the images encrypted with any combination of the operations of the new algorithm were relatively uniform and different from the histograms of the original images. They gave little indication that may help statistical attacks. As seen in the figure, using two steps of encryption (partition and shuffling) produced histogram slightly different from the one using partition alone, where using all three steps of the algorithm gave the best results.

The mean squared error for two images, stored in matrices A and B, is computed as follows:

$$MSE = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(A[i,j]-B[i,j]\right)^2 \qquad (2)$$

PSNR is computed as:

$$PSNR = 10\ \log_{10}\left(\frac{MAX^2}{MSE}\right) \qquad (3)$$

Where:
MAX = The maximum pixel value of the image
$PSNR$ = Measurement unit is the decibel (dB)

A lower PSNR value is desired for encrypted images since it indicates more noise and, therefore, more resistance to attacks.

Figure 3 shows PSNR computed for encrypted images resulting from encrypting the original image using different combinations of partitioning, XOR and S-box encryptions: Partitioning with XOR alone, partitioning with XOR followed by shuffling and applying all three steps. As it appears in the figure, the PSNR values of these methods were similar. The average PSNR values for the results are shown in the second column of Table 2. The average PSNR value was the highest (i.e., best) when applying all three steps, where using only one or two steps produced a close average result. The average was slightly lower when two steps were used and the lowest (worst) when only one step is used.

The correlation, r, between two images, stored in matrices A and B, is computed as follows, where $\bar{A}$ and $\bar{B}$ are mean values for matrices A and B, respectively:

$$r = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(A[i,j]-\bar{A})(B[i,j]-\bar{B})}{\sqrt{\left(\sum_{i=1}^{m}\sum_{j=1}^{n}(A[i,j]-\bar{A})^2\right)\left(\sum_{i=1}^{m}\sum_{j=1}^{n}(B[i,j]-\bar{B})^2\right)}} \qquad (4)$$

A lower correlation value between an image and its encryption indicates less resemblance between them, which provides more resistance to attacks.

The correlation value computed for encrypted images resulting from encrypting the original image using different combinations of XOR and S-box encryptions is shown in Fig. 4. As seen in the figure, using XOR without S-box generally gave the highest (worst) value of all four encryption combinations, while other encryption combinations gave values similar to each other. This observation is supported by the average correlation value computed for each operation combination, shown in the third column of Table 2, where the average was taken for the absolute values of correlation for the sample images.

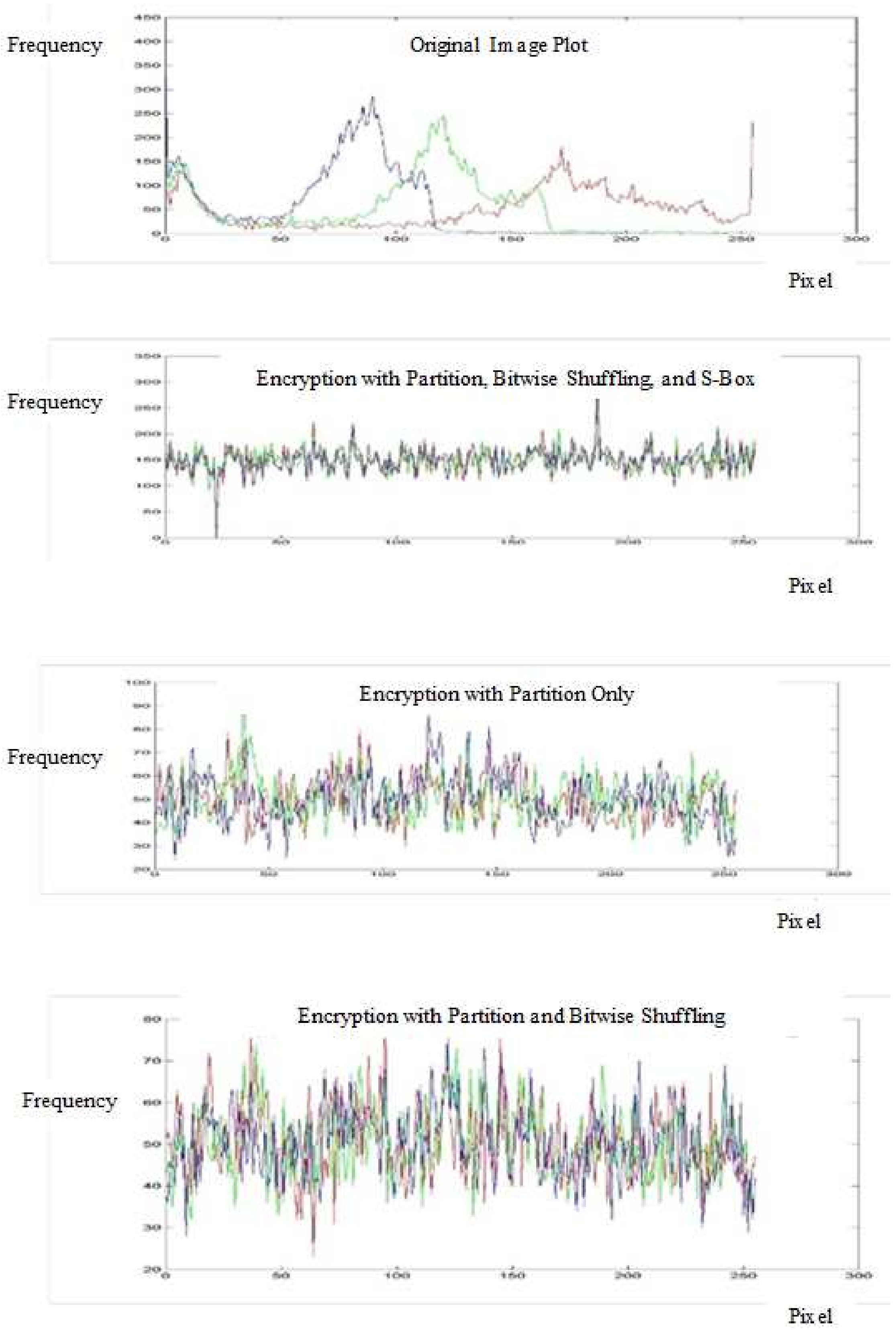

Fig. 2. Histograms of image encrypted with the combination of the operation of the new algorithms

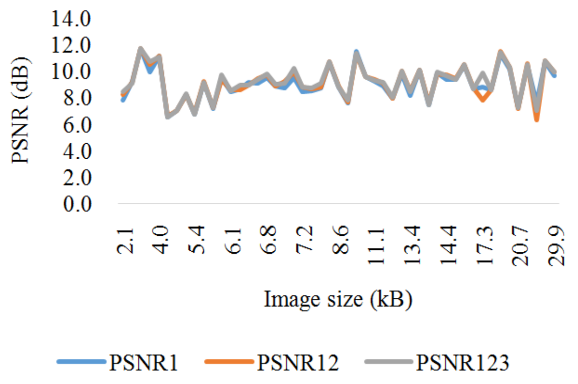

Fig. 3. PSNR resulting from different combined operations

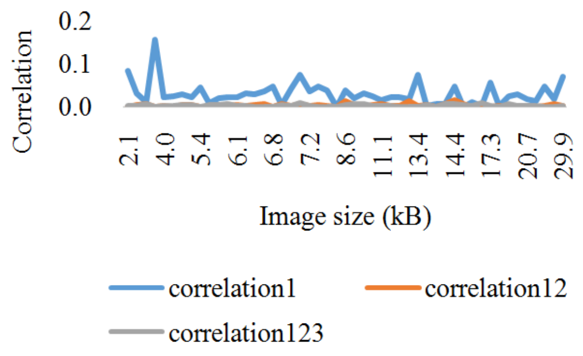

Fig. 4. Correlation resulting from different combined operations

Table 2. Average values with different combinations of encryption operations (50 images)

Operation combination	PSNR	Correlation	Entropy	MSSIM
Partition	9.07	0.032	7.967	0.0203
Partition and shuffling	9.11	0.005	7.968	0.0188
Partition, shuffling and S-box	9.21	0.003	7.965	0.0177

The average correlation computed when applying S-box once was lower than the average computed when S-box was used twice and did not make much difference whether S-box was applied before or after XOR.

The randomness of pixel values can be measured with entropy, where entropy is computed as follows:

$$H = -\sum_{i=1}^{MAX} (P(i)\log_2(P(i))) \quad (5)$$

Where:

MAX	= The maximum pixel value of the image
$P(i)$	= The probability of the occurrence of pixel value i

A higher entropy indicates higher randomness and, consequently, higher resistance to statistical attacks. The average entropy value for the original sample images was 2.65707. Using any of the three encryption steps produced clearly higher entropy than the original image, as seen from the plotted values in Fig. 5. It can be observed from the figure that the entropy results when using one, two or all three steps of the algorithm were relatively similar. These results are supported by the average entropy values shown in the third column of Table 1. Even though the high values of entropy indicated high randomness and resistance to attacks, they did not differentiate among the use of one, two, or all three steps of the algorithm. This is due to the relatively small size of sample space of pixel values (0 to 255). Examining the difference between the average entropy values from the use of one, two, or all three steps, obtained from the fourth column of Table 2, it can be seen that the maximum difference between any two values is 0.037%. This is a negligible small value that does not show which combination of steps is better than the others. Therefore, other statistical analysis methods must be used, such as PSNR, correlation and SSIM.

Structural Similarity (SSIM) index is a method for measuring the perceived quality of digital images and videos or measuring the similarity between two images. It is a perception-based model that considers image degradation as perceived change in structural information, while also incorporating important perceptual phenomena. It may be used in measuring image quality based on an initial distortion-free image as reference. SSIM is designed to improve on traditional methods like Peak Signal-to-Noise Ratio (PSNR) and Mean Squared Error (MSE) by focusing on structural information rather than absolute error. SSIM is computed as in Equation 1:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (6)$$

Where:

- μ_x the average of x
- μ_y the average of y
- σ_x^2 the variance of x
- σ_y^2 the variance of y
- σ_{xy} the covariance of x and y
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ two variables to stabilize the division with weak denominator
- L the dynamic range of the pixel-values (typically this is 2^{bpp} -1), bpp is bits per pixel
- $k_1 = 0.01$ and $k_2 = 0.03$ by default

For a single overall quality measure of an image, the Mean SSIM (MSSIM) value for two images, X and Y, with M local windows is computed as in Equation 7:

$$MSSIM(X,Y) = \frac{1}{M}\sum_{j=1}^{M} SSIM(x_j, y_j) \quad (7)$$

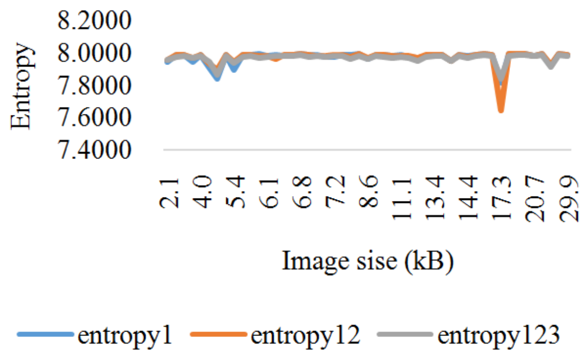

Fig. 5. Entropy resulting from different combined operations

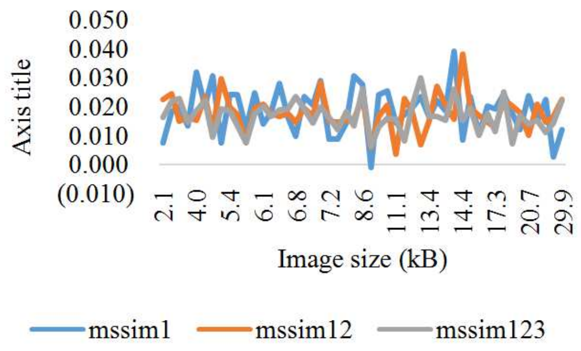

Fig. 6. MSSIM resulting from different combined operations

A low MSSIM for encrypted images is desired since it indicates more degradation in the encrypted image, i.e., less resemblance to the original image. The MSSIM values for different combinations of the encryption steps are illustrated in Fig. 6. The MSSIM values using one, two or all three steps were not visibly different in the figure. However, the average of these values, computed for all sample images, was lower (i.e., better) when using two steps rather than one step and it was the best when using all three steps of the algorithm. These averages are shown in the last column of Table 2.

Overall, all PSNR values were high and all correlation and MSSIM values were low. This indicates resistance to statistical attacks. The different statistical analysis results agreed in showing the increase in encryption effectiveness with each added step of the algorithm.

Conclusion

A new encryption algorithm was presented. The new algorithm employs shuffling procedures combined with variable-length key-dependent XOR and S-box substitutions to perform lossless image encryption. Analysis of different combinations of these encryptions showed that applying all three produced the best results compared to using only one or two encryptions.

Statistical analysis using histograms, PSNR, correlation, entropy and MSSIM showed the algorithm is not vulnerable to statistical attacks. In addition, the huge number of possible keys combined with a huge number of possible substitutions makes a brute-force attack on the algorithm impossible.

Acknowledgement

The author likes to thank and acknowledge Dr. Ayman Abdalla for his suggestions to improve the quality of the paper.

Funding Information

This study was supported by Al-Zaytoonah University of Jordan.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Al-Husainy, M.A.F., 2012. A novel encryption method for image security. Int. J. Secur. Applic., 6: 1-8.

Ali, Z.M. and N.M.A. Makhzoum, 2012. Computation of private key based on divide-by-prime for Luc cryptosystems. J. Comput. Sci., 8: 523-527. DOI: 10.3844/jcssp.2012.523.527

Backes, M. and B. Pfitzmann, 2008. Limits of the BRSIM/UC soundness of Dolev-Yao-style XOR. Int. J. Inf. Secur., 7: 33-54. DOI: 10.1007/s10207-007-0040-z

Chatzichristofis, S.A., O. Marques, M. Lux and Y. Boutalis, 2014. Image encryption using the recursive attributes of the eXclusive-OR filter on cellular automata. J. Cellular Automata, 9: 125-137. DOI: 10.1007/978-3-642-33350-7_35

Chengqing, L. and Y. Liu, 2013. Breaking a chaotic image encryption algorithm based on modulo addition and XOR operation. Int. J. Bifurcat. Chaos, 23: 1350075-1350087. DOI: 10.1142/S0218127413500752

Do, J.M. and Y.J. Song, 2014. Secure streaming media data management protocol. Int. J. Secur. Applic., 8: 193-202.

El-Fishawy, N. and O.M. Abu Zaid, 2007. Quality of encryption measurement of bitmap images with RC6, MRC6 and Rijndael block cipher algorithms. Int. J. Net. Sec., 5: 241-251.

Kaipa, A.N.R., V.V. Bulusu, R.R. Koduru and D.P. Kavati, 2014. A hybrid cryptosystem using variable length sub key groups and byte substitution. J. Comput. Sci., 10: 251-254. DOI: 10.3844/jcssp.2014.251.254

Nag, A., J.P. Khan, S. Singh, S. Biswas and D. Sarkar *et al*., 2011. Image encryption using affine transform and XOR operation. Proceedings of the International Conference on Signal Processing, Communication, Computing and Networking Technologies, Jul. 21-22, IEEE Xplore Press, Thuckafay, pp: 309-312. DOI: 10.1109/ICSCCN.2011.6024565

Sivakumar, T. and R.A. Venkatesan, 2014. A novel approach for image encryption using dynamic SCAN pattern. IAENG Int. J. Comput. Sci., 41: 91-101.

Wang, X.Y. and T. Wang, 2012. A novel algorithm for image encryption. Int. J. Modern Phys. B, 26: 1250175-1250184. DOI: 10.1142/S0217979212501755

Yahya, A. and A.M. Abdalla, 2009. An AES-based encryption algorithm with shuffling. Proceedings of the International Conference Security and Management, (SAM' 09), pp: 113-116.

Yahya, A.A. and A.M. Abdalla, 2008. A shuffle image-encryption algorithm. J. Comput. Sci., 4: 999-1002. DOI: 10.3844/jcssp.2008.999.1002