Separable High Capacity Reversible Data Hiding Algorithm for Encrypted Images

Iyad Jafar
Computer Engineering Department,
The University of Jordan,
Jordan
iyad.jafar@ju.edu.jo

Khalid Darabkh
Computer Engineering Department,
The University of Jordan,
Jordan
k.darabkeh@ju.edu.jo

Fahed Jubair
Computer Engineering Department,
The University of Jordan,
Jordan
f.jubair@ju.edu.jo

Abstract: *This paper presents a separable Reversible Data Hiding Algorithm for Encrypted Images (RDHEI) that consists of three phases. The encryption phase in the algorithm circularly shifts the columns in the image by a random amount, blocks the image into equal and regular blocks and maps them to irregular blocks generated based on Hilbert filling curve, and finally complements a random subset of the blocks. The embedding phase is essentially an adapted version of the modification of the prediction errors algorithm that is applied to each block in the encrypted image independently. In the decryption phase, and since the algorithm is separable, the user can extract the data only, decrypt the image only, or can perform both actions depending on the type of keys he has. When compared to a very similar and recent algorithm, performance evaluation proved the ability of the proposed algorithm in increasing the embedding capacity with reasonable quality of the directly decrypted image. In terms of the security, the analytical and quantitative assessment showed the superiority of the proposed algorithm in protecting the encrypted image.*

Keywords: *Encryption, embedding capacity, privacy, reversible data hiding.*

Received April 23, 2021, accepted December 2, 2021
https://doi.org/10.34028/iajit/19/5/13

1. Introduction

With the wide spread and use of digital multimedia objects, copyright protection, tamper detection and privacy have become major security concerns. In practice, encryption, digital forensics, and secret sharing [19, 21] are some of the technologies that have been employed to protect multimedia objects. Data Hiding is another concept in which secret data is embedded imperceptibly in multimedia objects such as text, audio, images and videos [2]. When digital images are considered as the host or cover objects, a special class of data hiding algorithms evolved in order to losslessly recover both the data and the cover image once the data is extracted. Such algorithms are referred to as Reversible Data Hiding (RDH) algorithms and they are useful in applications that do not tolerate changes in the cover image such as artistic works, medical and military images [15].

Many RDH algorithms have been proposed in the literature to maximize the amount of data that can be embedded, improve the quality of the stego image, and guarantee reversibility. These algorithms can be grouped into four basic fundamental categories, namely, lossless compression [3, 5], Difference Expansion (DE) [1, 17], Pixel Value Ordering (PVO) [8, 10] and Histogram Shifting (HS) [6, 14].

Given the challenge of optimizing the design of RDH algorithms in terms of embedding capacity, quality of stego image and complexity, the emergence of cloud technology has imposed a new and pressing factor in the design of RDH algorithms [13, 21]. Recently, cloud services allow users to store and manipulate their images. For management and storage utilization purposes, it might be useful to embed data in the images that reside on the cloud for different purposes such as image annotation and authentication. However, using the images to store the data of a third party jeopardizes the privacy of the content owner. One workaround is to allow the content owner to encrypt the image using one of the known encryption schemes such as permutation, stream cipher, DES, or AES before data is embedded into the image in a reversible manner. RDH algorithms designed for this purpose are usually referred to as Reversible Data Hiding in Encrypted Images (RDHEI).

Several RDHEI algorithms have been proposed in the literature. One way to classify these algorithms is based on separability. Effectively, an RDHEI algorithm is considered either joint or separable. In the joint methods, data extraction and image recovery are performed jointly. In other words, extracting the data requires decrypting the image first. Many algorithms have been proposed in this area [7, 20, 23]. The embedding capacity in such algorithms is usually low and the quality of the restored image is degraded with the possibility of generating errors in the extracted data, especially at high payloads. On other hand, the separable algorithms [9, 11, 22, 24] are designed such that data extraction and image decryption can be performed separately.

In this paper, a separable reversible data hiding algorithm in the encrypted image domain is proposed. The encryption process in the proposed algorithm relies on circularly shifting the columns of the image, blocking the image into $B\times1$ blocks, mapping the blocks to irregularly-shaped blocks generated from Hilbert pattern, complementing the pixels in a set of randomly selected blocks, and finally permuting these blocks. For data embedding, the data hider scans the blocks and order their pixels based on their linear index and then uses the modification of prediction error approach to embed the data. The algorithm showed impressive performance in increasing the embedding capacity and the privacy of the encrypted image when compared to a related state-of-the-art RDHEI algorithm with a slight and acceptable degradation in the quality of the directly-decrypted image at high payloads.

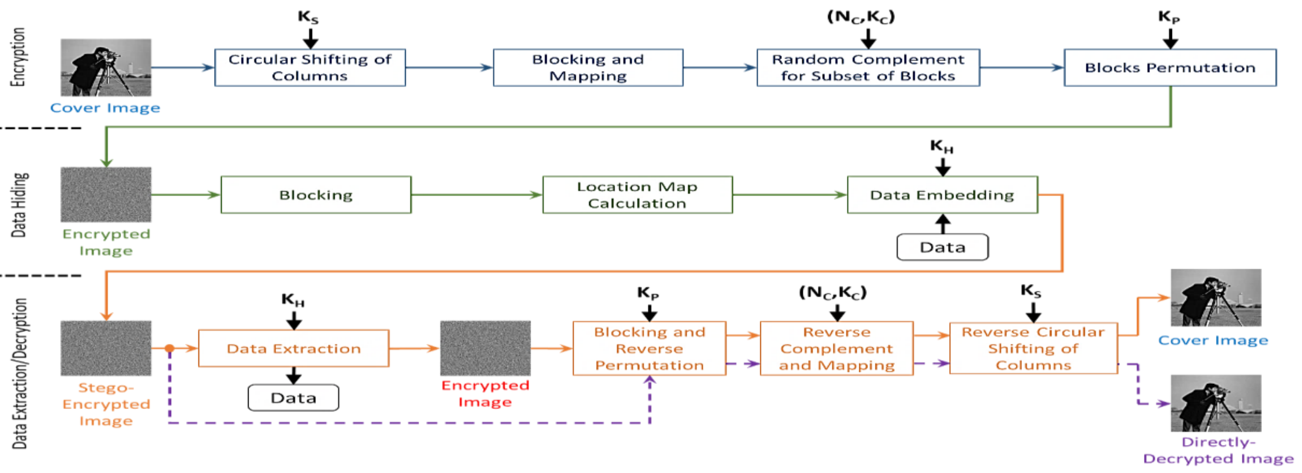


Figure 1. Framework of the proposed algorithm.

The rest of the paper is organized as follows. In section 2, we review two algorithms that are strongly related to the proposed algorithm. We present the details of the proposed algorithm in section 3 and evaluate its performance in section 4. We conclude the paper in section 5.

2. Related Work

Shui *et al.* [17] proposed a separable permutation-based algorithm that encrypts the image using Fisher-Yates permutation after partitioning the image into non-uniform blocks obtained through traversing the Hilbert curve. Additionally, another key is used to scramble and permute pixels in each block. The authors argued that using such permutation to encrypt the image is more secure, analytically and visually, than stream cipher, in addition to fact that permutation does not alter the local correlation between pixels within the block and this has direct impact on increasing the embedding capacity. For data embedding, the algorithm relied on PVO [8] to embed the data through prediction error expansion of the minimum and maximum values in each block.

The algorithm showed impressive results in terms of the quality of the directly decrypted image. This is due to the fact that only two pixels are modified in each block in the worst case. However, the embedding capacity is quite limited especially when the block size is increased. Technically, for an image with $M\times N$ pixels that is blocked into blocks each with B pixels, the maximum number of pixels that can be used for embedding is $2\times M\times N/B$ pixels which implies that only $(2/B\times100)\%$ of the pixels are possibly utilized in embedding.

Nonetheless, using only two pixels from each block for embedding has the advantage of Pushing The Peak-Signal-To-Noise Ratio (PSNR) of the directly-decrypted image in this algorithm to high values which gets better as the block size increases since fewer pixels are modified. In terms of security, the algorithm relies on using two keys to perform permutation of blocks and pixels and avoided using other forms of encryption in order to preserve the correlation of the pixels. The permutation is expected to affect some of the security measures in the encrypted image; however, since the pixels' values are not modified, the entropy of the image, which is an important security measure, after encryption is not affected.

3. The Proposed Algorithm

The framework of the proposed algorithm is shown in Figure 1. Similar to other RDHEI algorithms, there are three parties involved in the framework; the content owner, the data hider and the receiver. The algorithm is composed of three phases: encryption, data hiding and decryption and data extraction. The following subsections detail the different phases in the proposed algorithm.

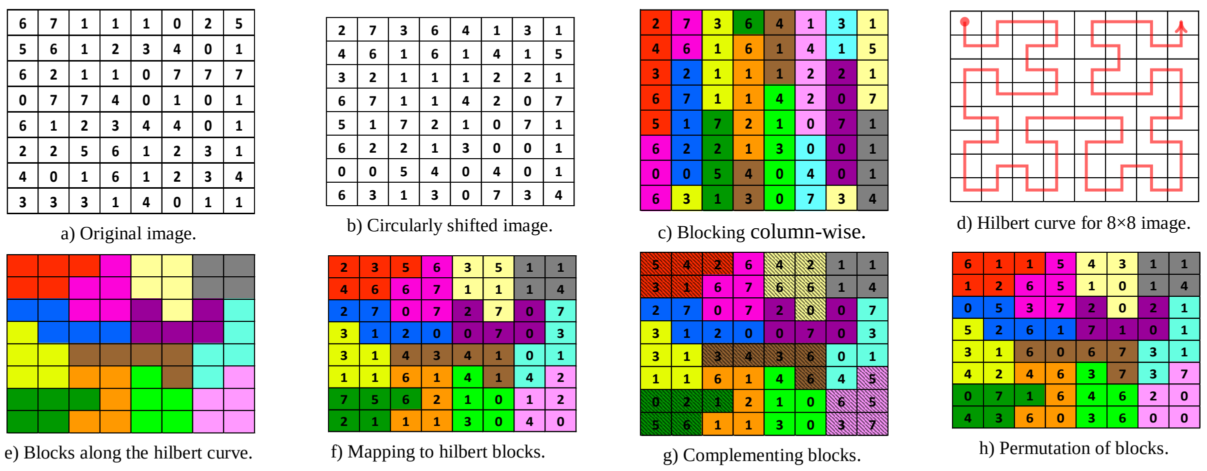

a) Original image.
b) Circularly shifted image.
c) Blocking column-wise.
d) Hilbert curve for 8×8 image.
e) Blocks along the hilbert curve.
f) Mapping to hilbert blocks.
g) Complementing blocks.
h) Permutation of blocks.

Figure 2. Example on encrypting a simple image in the proposed algorithm.

3.1. Encryption Phase

For an n-bit $M\times N$ Cover Image (CI), the content owner starts the encryption phase by considering the columns in the image. Using the shifting key (K_S), N random values are generated to form the set $S=\{S_k, 0\leq k\leq N-1\}$. The range of the values in S is $[0,M-1]$, where M is the number of rows in the image, which is the number of pixels in each column. Afterwards, the pixels in each column are circularly shifted using S such that the new position of pixel $Q_{k,i}$ in column C_K is given by

$$q'_{k,i} = (q_{k,i} + s_k)\ mod\ (M-1) \quad (1)$$

The purpose of this step, when compared to [16], create some sort of visual confusion in the encrypted image to avoid scrambling and permuting pixels within the block, when the image is blocked later. Figure 2-a) shows a hypothetical 8×8 3-bit image that we will use throughout this subsection to explain the encryption process. Figure 2-b) shows the original image that has been shifted column-wise circularly when S is {3, 0, 1, 3, 4, 5, 2, 1}.

The next step in the encryption phase is block permutation. Technically, the circularly shifted image is blocked column-wise into $B\times1$ blocks. This blocking generates the set G of N_B blocks, where $N_B=[M\times N/B]$, such that $G=\{g_k, 0\leq k\leq N_B-1\}$. The ordering and numbering of the blocks in G is based on column-wise scanning. For example, the image in Figure 2-c) is blocked into 13 blocks such that the red block is block number 1 while the gray block is block number 13. Each element in G is basically a B-tuple that contains the values of the pixels in the block when they are read column-wise such that $gk=(v_{k,0}, v_{k,1}, v_{k,2}, \ldots, v_{k,B\text{-}1})$. For instance, the tuple that represents the second pink block g_2 in Figure 2-c) is (6, 0, 6, 7, 6).

Next, and similar to [16], the Hilbert curve for the $M\times N$ image is generated and the pixels along the path of the curve are grouped into B-pixel blocks to generate the set $H=\{h_k, 0\leq k\leq N_B-1\}$. The blocks in H are ordered and numbered based on their location along the Hilbert curve. For example, the red block in Figure 2-e) is block h_1, the dark-pink block is h_2, and so on.

Initially, each element in H is an empty B-tuple, $hk=(x_{k,0}, x_{k,1}, x_{k,2}, \ldots, x_{k,B\text{-}1})$, or less for last block in case $M\times N$ is not an integer multiple of B. The order of the elements inside h_k is based on their column-major in the image. Figure 2-d) shows the Hilbert curve for an 8×8 image and Figure 2-e) shows the blocks in H when the block size is 5. Having the two sets, G and H, blocks in G are mapped to those in H in the same order, i.e., $g_k \rightarrow h_k$. The elements in the h_k tuples are filled with the values in g_k such that $v_{k,I}\longrightarrow x_{k,i}$. Figure 2-f) shows how the blocks in H are filled with values from the tuples found in G.

In order to improve the security of the final encrypted image, the next step in encryption is to use the complement key K_C to form the set T that contains N_C random values in the range $[0, N_{B\text{-}1}]$ The set T identifies the block numbers in H that are randomly selected and processed by complementing their pixels such that the new pixel value of the i^{th} pixel in the h_k block is calculated using.

$$x'_{k,i} = 2^n - x_{k,i}, \forall\, k\in T \quad (2)$$

This step has the effect of flipping the pixels in N_C randomly selected blocks, which is expected to increase the security of the encrypted image to a certain extent, while maintaining the relative spatial correlation between the pixels within the block. Figure 2-g) demonstrates the effect of applying the complement step on a subset of the blocks (diagonally-patterned blocks) in the image in Figure 2-f) where n here is assumed to be 3 since the image in Figure 2-a) is assumed to be a 3-bit image.

For example, consider the original block h_1=(2,4,3,6,5) in Figure 2-f) which is selected as one of the blocks to be complemented to become (5, 3, 4, 1, 2). It is clear how this simple complement operation has preserved the absolute difference between successive values which is an important factor in the embedding phase as we will discuss in the following subsection.

The last step in the encryption phase is to use the permutation key K_P to generate K_B randomly-ordered unique values in the range $[0, N_B - 1]$ to form the set U and then use it to permute and scramble the blocks in H such that the contents of each block h_k is mapped to $h_{U(k)}$ to form the encrypted image *EI*. In case $M \times N$ is not an integer multiple of B, the last block is not permuted and the size of set U is $N_B - 1$ and the range of values in it becomes $[0, N_B - 2]$. Figure 2-h) demonstrates this last step when applied to the image in Figure 2-g) using U={4, 7, 11, 10, 3, 1, 8, 12, 2, 5, 9, 6}.

3.2. Data Embedding Phase

Once the Encrypted Image (EI) is obtained from the encryption phase, the content owner can safely hand it to the data hider in order to start the data embedding phase. Since the correlation between pixels in the same block is somehow preserved, the embedding operation should process the encrypted image block-wise. Accordingly, and after encrypting the data to be embedded using the data hiding key K_H, the embedding phase starts by generating the Hilbert curve for an $M \times N$ image. Afterwards, and knowing the block size B, the image is blocked into B-pixel blocks by scanning the pixels in *EI* along the generated path to obtain the set $H=\{h_k, 0 \le k \le N_B - 1\}$. such that each element in h_k is a B-tuple that contains the pixels values in the block ordered by column-major, i.e., $hk = (v_{k,0}, vx_{k,1}, v_{k,2}, \ldots, v_{k,B-1})$. Similar to the blocking performed during encryption, the blocks in H are numbered according to their order along the Hilbert path.

Next, the blocked image is scanned and the blocks are checked to determine whether they contain pixels with values of 0 and/or 2^n -1 as these values may result in underflow/overflow if data is embedded. These blocks have to be excluded from data embedding. In order to record the overflow/underflow blocks, the set location map (L) is created such that $L=\{l_k, 0 \le k \le N_B - 1\}$, where l_k is basically a one bit that is set to 1 in case h_k is an overflow/underflow block; otherwise, l_k it is zero. Figure 3-a) shows the blocked encrypted while Figure 3-b) shows the blocks that are excluded from data embedding (horizontally-patterned) due to overflow.

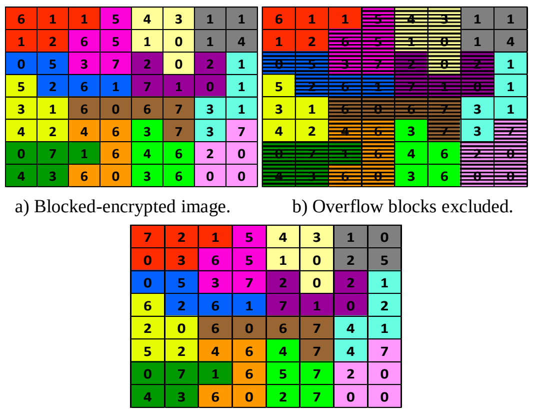
For the remaining blocks, they are scanned in order to embed the given data. For every block that can be used in embedding, $h=(v_0,v_1,v_2, \ldots, v_{B-1})$, the pixels are manipulated to store the given bits as follows. For pixels v_i, $i \in [1, B\text{-}2]$, the prediction is calculated using.

$$p_i = \lfloor (v_{i-1} + v_{i+1})/2 \rfloor \tag{3}$$

Where $\lfloor x \rfloor$ is the floor operator, and hence , the prediction error is:

$$pe_i = v_i - p_i \tag{4}$$

Accordingly, the pixel value in the block is then modified and updated to embed one bit, if possible, using where b is the bit to embedded. Effectively, prediction errors of 0 and -1 are utilized to embed the data and they are either left unchanged when b is 0 or incremented/decremented by 1 when b is 1. Other prediction errors are incremented or decremented by 1 based on their polarity in order to open space for embedding data in prediction errors of 0 and -1.


a) Blocked-encrypted image. b) Overflow blocks excluded.

c) Stego-encrypted image.

Figure 3. Data embedding steps.

$$v_i = \begin{cases} p_i + pe_i + b\,;\ pe_i = 0 \\ p_i + pe_i - b\,;\ pe_i = -1 \\ p_i + pe_i + 1\,;\ pe_i > 0 \\ p_i + pe_i - 1\,;\ pe_i < -1 \end{cases} \tag{5}$$

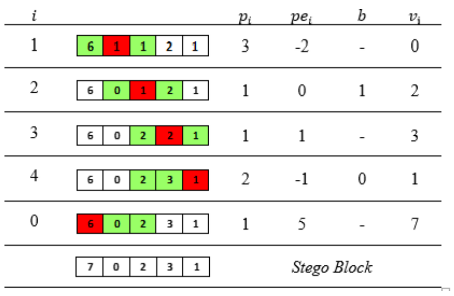
For the last pixel in the block, v_{B-1}, Equations (4), and (5) are used to perform embedding; however, the prediction is calculated using.

$$p_{B-1} = \lfloor (v_{B-2} + v_{B-3})/2 \rfloor \tag{6}$$

In order to maximize the usage of all pixels in the block for data embedding, the first pixel v_0 is also considered for data embedding after processing pixel v_{B-1} using the same steps but the prediction for this pixel is calculated such that

$$p_0 = \lfloor (v_1 + v_2)/2 \rfloor \tag{7}$$

To demonstrate the idea, let's consider the first block, the red block h=(6, 1, 1, 2, 1), in the image in Figure 3-b) and assume that the data to be embedded is a stream of bits that alternates between 1 and 0, i.e., d= (1, 0, 1, 0,….). Figure 4 shows the embedding steps for processing all pixels in the first block. In each step, the pixel under consideration is shaded with red while the pixels that are used to calculate the prediction are shaded with green. For example, the value of v_1 in this block is 1 while the neighboring pixels have values of 6 and 1, thus the p_1 is $\lfloor (6+1)/2 \rfloor$ which is 3 and the prediction error pe_1 is -2. According to (5), this pixel will be decremented by 1 and no bit is embedded. So, the new pixel value v_1 is 0. The same process is applied to the remaining pixels, except in calculating the prediction for the last and first pixels in the block. Figure 3-c) shows the stego-encrypted image when all of the blocks are considered for the data embedding.



i		p_i	pe_i	b	v_i
1	🟩6 🟥1 🟩1 2 1	3	-2	-	0
2	6 🟩0 🟥1 🟩2 1	1	0	1	2
3	6 0 🟩2 🟥2 🟩1	1	1	-	3
4	6 0 🟩2 🟩3 🟥1	2	-1	0	1
0	🟥6 🟩0 🟩2 3 1	1	5	-	7
	7 0 2 3 1	*Stego Block*			

Figure 4. Embedding example for one block (Pixel being processed is in red while pixels used for prediction are in green).

Depending on the size of the data to be embedded, all or subset of the blocks and pixels within the last block might be used for data embedding. In order to extract the data correctly and ensure the reversibility, the data hider should provide the receiver with address of the last pixel A_e that was used for embedding. Additionally, the receiver should have the location map L in order skip the overflow blocks during data extraction. This information forms the Overhead (OH) to the embedding operation that has to be communicated to the receiver. Alternatively, and similar to how most RDH algorithms treat the overhead, the embedding phase reserves the first N_{OH} pixels to store A_e and L. Effectively, the location map is first compressed to reduce its size to S_L bits. Then, the Least Significant Bits (LSBs) of the first N_{OH} pixels along the Hilbert curve are replaced with A_e, S_L and L. These LSBs are pre-appended to the data to be embedded and stored in the image during embedding. The number of pixels to be used for this purpose is:

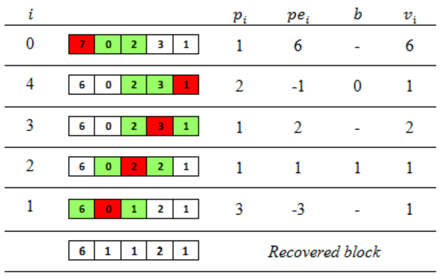
$$N_{OH} = \lceil log_2(M \times N) \rceil + S_L + L \tag{8}$$

The value $\lceil log_2(M \times N) \rceil$ is basically the number of bits required to store A_e. Since the size of the location map after compression will vary from image to image, and thus S_L, we opt to set and fix S_L to $\lceil log_2(M \times N) \rceil$ bits as well for any image.

3.3. Data Extraction and Image Decryption Phase

In this phase, the receiver may perform three different operations depending on the available keys as depicted in Figure 1. In case the receiver has the data hiding key K_h only, then he can only extract the data from the stego-encrypted image to obtain the encrypted image and the embedded data.

Effectively, data extraction starts by generating the Hilbert curve for the $M{\times}N$ stego-encrypted image and reading the LSBs of the first $2\lceil log_2(M \times N) \rceil$ pixels along the path. The first half of these bits is A_e while the second half is S_L which is used to read the following S_L bits that represent the location map. After decompressing the location map, the stego-encrypted image is blocked using B and the data extraction proceeds starting from the block that contains the pixel with address A_e. If the corresponding bit of a block is 1 in the location map, then the block skipped. Otherwise, restoring the pixels' values in the block and extracting the bits is as follows.



i		p_i	pe_i	b	v_i
0	🟥7 🟩0 🟩2 3 1	1	6	-	6
4	6 0 🟩2 🟩3 🟥1	2	-1	0	1
3	6 0 🟩2 🟥3 🟩1	1	2	-	2
2	6 🟩0 🟥2 🟩2 1	1	1	1	1
1	🟩6 🟥0 🟩1 2 1	3	-3	-	1
	6 1 1 2 1	*Recovered block*			

Figure 5. Extraction example for one block (Pixel being processed is in red while pixels used for prediction are in green).

For a block in the stego-encrypted image, h=(v_0,v_1,v_2, …, $v_{B\text{-}1}$), processing starts by considering the first pixel v_0 and calculating the prediction and prediction errors, p_0 and pe_0, using (7) and (4), respectively. Based on the prediction error value, the original value of v_0 in the encrypted image is restored using

$$v_i = \begin{cases} p_i + pe_i \;;\; pe_i \in \{-1,0\} \\ p_i + pe_i - 1 \;;\; pe_i > 0 \\ p_i + pe_i + 1 \;;\; pe_i < -1 \end{cases} \tag{9}$$

While the embedded bit, if any, is extracted such that

$$b = \begin{cases} 0 \;;\; pe_i \in \{-1,0\} \\ 1 \;;\; pe_i \in \{-2,1\} \\ None \;;\; otherwise \end{cases} \tag{10}$$

Next, exaction moves to the last pixel in the block $v_{B\text{-}1}$, calculates the prediction using (6) and restores the original value and the embedded bits using (9) and (10), respectively. For the remaining pixels, they are processed in a reverse order, i.e., from $v_{B\text{-}2}$ to v_1, with the prediction calculated using (3). This process repeats for all blocks until the first block is processed.

Once all data is extracted, the first N_{OH} bits of the data are stored back into the first N_{OH} pixels in the Hilbert path and the embedded data is decrypted using K_H. Figure 5 demonstrates the extraction process for the red block in the stego-encrypted image in Figure 3-c). It is clear in this example how the original values in the first block in the encrypted image in Figure 2-e) is restored and how the embedded bits are extracted correctly.

The second case is when the receiver has the encryption keys only, i.e., K_P, N_C, K_C and K_S. In this case, the receiver can only decrypt the stego-encrypted image to obtain the directly-decrypted image. Specifically, and after blocking the image over the Hilbert path into N_B blocks, the receiver decrypts the stego-encrypted image by simply reversing the operations performed during encryption.

Technically, the key K_P is used to generate the N_B randomly-ordered unique values in the range [0, N_B-1] to form the set U that is used to reverse the permutation step such that block $h_{U(k)}$ is mapped to h_k, except for the last block when $M{\times}N$ is not and integer multiple of B. Next, and using K_C and N_C, the blocks that were complemented during encryption are identified and the their pixels are complemented using (2). Afterwards, the Hilbert blocks are mapped to B×1 blocks and in the same order. Finally, the key K_S is used to generate the set $S=\{S_k, 0 \leq k \leq N - 1\}$ to reverse the shifting operation performed on the columns such that the original position of pixel $q_{k,i}$ in column C_k is given by

$$q_{k,i} = (q'_{k,i} - s_k)\ mod\ (M-1) \tag{11}$$

The result of this decryption process is the directly-decrypted image which is slightly different from the original cover image due to the fact that the embedded data has not been extracted. As a matter of fact, improving the quality of this directly-decrypted image is one important metric that RDHEI algorithms compete in.

The third case is when the receiver has both the data hiding and encryption keys. In this case, the receiver can extract the data first from the stego-encrypted image using the steps we discussed in the first case. Afterwards, the receiver uses the encryption keys to decrypt the image to recover the original image by following the steps discussed in the second case. The output is an image that exactly matches the original cover image as shown in Figure 1.

4. Experimental Results

4.1. Evaluation Setup and Performance Metrics

In this section, we evaluate the performance of the proposed algorithm and compare it with Shui *et al.*'s [17] algorithm. The MATLAB code for both algorithms along with the evaluation code can be found on (shorturl.at/gqrs9) In the evaluation, we used six 512×512 test images, namely; Clock, Cat, Bird, Tank, Butterfly and Cameraman [18, 21] which are shown in Figure 6.

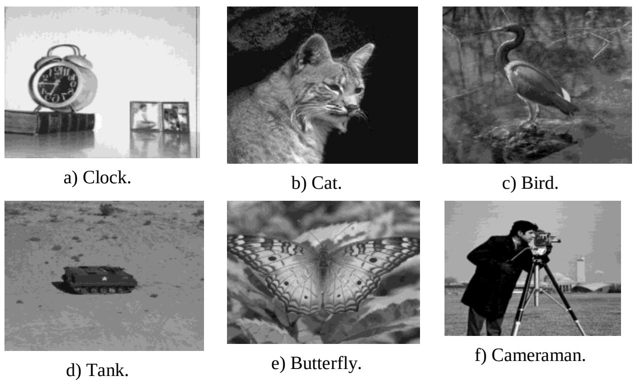


Figure 6. Test images.

The evaluation considers the embedding capacity (bits) and the quality of the directly decrypted images using the peak PSNR. To evaluate the security of the encrypted images, four different metrics are considered; the PSNR value between the cover image and the encrypted image, the entropy, Number Of Changing Pixel Rate (NPCR) and the Unified Averaged Changed Intensity (UACI) [4, 12]. The last three metrics are usually used as indicators for the resistibility to differential attacks.

4.2. Evaluation Under Maximum Embedding Capacity

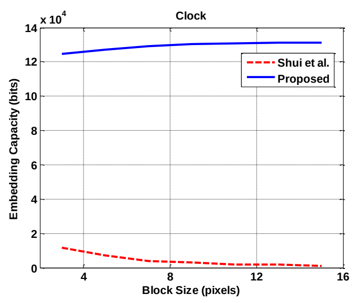
The first experiment in the evaluation considers assessing the maximum amount of data that can be embedded in the test images when the block size B is varied is varied between 3 and 15 pixels.

Figure 7 shows the maximum embedding capacity obtained using the two algorithms for different block size. It is evident how the proposed algorithm outperforms [17] significantly in terms the embed data. This result is expected given the fact that [17] uses only two pixels in each block for embedding; hence, and as we mentioned earlier, only (2/B×100)% of the pixels in each block is considered for embedding. So, as the block size increases, the embedding capacity decreases.

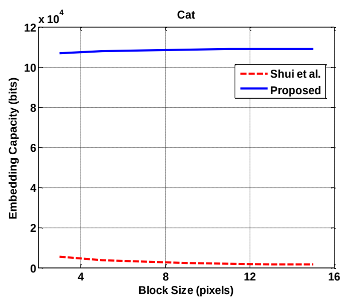
On the contrary, the proposed algorithm uses all pixels in the blocks for embedding and this gives the proposed algorithm the potential to achieve higher capacities even when the block size is increased. Ignoring the overhead and assuming there are no overflow blocks, the embedding capacity in the proposed algorithm could be as high as M×N.

Despite the impressive increase in the embedding capacity for the proposed algorithm, the directly-decrypted image is expected to have relatively lower PSNR values when compared to Shui *et al.*'s [17] algorithm since the proposed algorithm affects all pixels in the block. Effectively, and in the worst case scenario, if we assume that all pixels in the proposed algorithm are either incremented or decremented by 1 during data embedding, then this implies that the lower bound for the PSNR value is $10log_{10}255^2$ which 48.13 dB. On the other hand, the lower PSNR bound in the worst case scenario in [16] is $10log_{10}(\frac{B\times 255^2}{2})$ if we assume that two pixels in each block are modified by 1. For example, when B is 3, the lower PSNR bound is 49.89 dB.

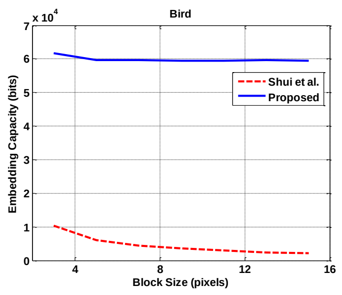
Figure 8 shows the PSNR values for the six test images in both algorithms when the block is varied. It is clear how the PSNR value in Shui *et al.*'s [17] algorithm increases as the block size increases. However, the PSNR of the directly-decrypted image is always lower in the proposed algorithm but it is always above 48.13 dB. Nonetheless, this difference in the PSNR values between the two algorithms under maximum embedding capacity slightly affects the visual quality of the decrypted image and this can be traded for the increased EC_{max} Figure 9 shows the original image Clock and the directly decrypted versions in both algorithms when the block size is 3 and 15. The visual appearance of the images in both algorithms is very similar but with advantage of having significantly larger embedding capacity in the proposed algorithm.



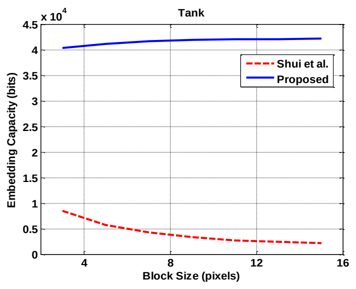
a) Clock.



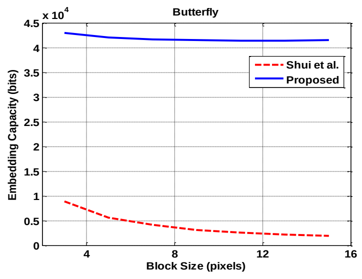
b) Cat.



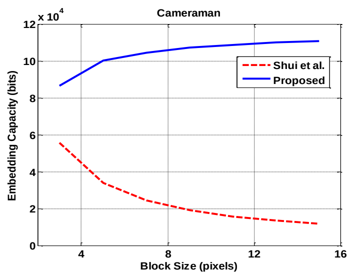
c) Bird.



d) Tank.



e) Butterfly.



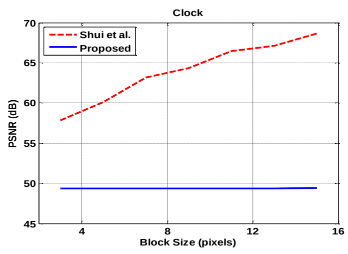
f) Cameraman.

Figure 7. Effect of block size on maximum embedding capacity for the six test images.

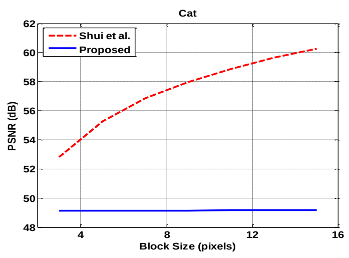
4.3. Security Analysis

One major concern in RDHEI algorithms is the security level of the encrypted image as it will be exploited by the data hider to embed the data. Thus, it is important that the encryption in RDHEI algorithms protects the privacy of the content owner. This subsection evaluates the security of encrypted images in both algorithms using the security metrics presented in 4.1.

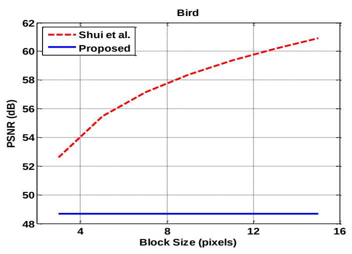
Analytically, the security of encryption algorithms mainly depends on the size of key space. The larger the key space, the more secure the algorithm is. For Shui *et al.*'s [17] algorithm, there are two keys; one for permuting the blocks and the other for permuting the pixels within the block. Hence, if we assume that the size of each key is r bits, then the key space for Shui *et al.*'s [17] algorithm is $min(2^r, N_B!) \times \min(2^r, B!)$, where N_B and B are the number of blocks and the number of pixels in the block, respectively. On the other hand, the proposed algorithm uses four different keys to shift the columns, complement subset of the blocks, and permute the blocks. So, the overall key space size will be $min(2^r, M \times N) \times \min(2^r, N_B!) \times \min(2^r \times N_B, \sum_{i=0}^{N_B} \binom{N_B}{i})$.



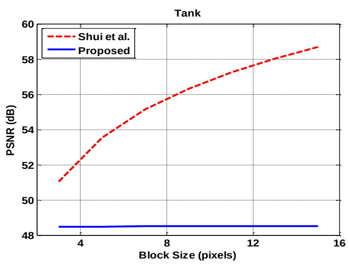
a) Clock.



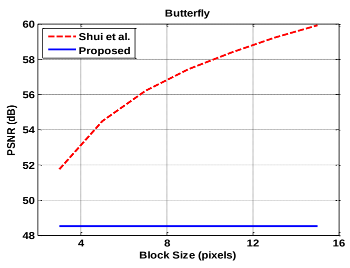
b) Cat.



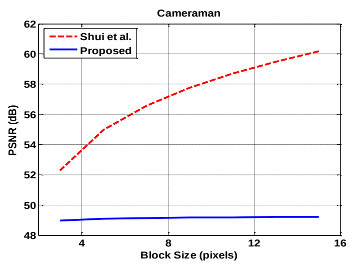
c) Bird.



d) Tank.

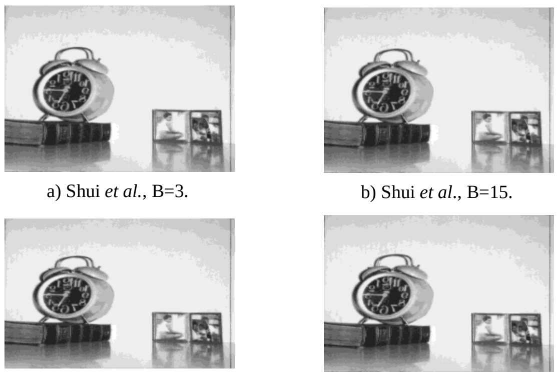


e) Butterfly.



f) Cameraman.

Figure 8. PSNR values of the directly-decrypted version for the six test images.



a) Shui *et al.*, B=3. b) Shui *et al.*, B=15.

c) Proposed, B=3. d) Proposed, B=15.

Figure 9. Directly-decrypted.

Figure 10 shows the size of the keys in both algorithms (log scale) as a function of block size for a 512×512 image and when the size for any key is 64 bits. It is apparent how the overall key size in the proposed algorithm is much higher than that in Shui *et al*.'s [17] algorithm, which indicates higher security since it implies that an attacker who attempts to decrypt the image through brute-force attack will need to spend large time. For example, the average time required to decrypt the image in the proposed and Shui *et al.* [17] algorithms was 3.04 and 0.24 seconds, respectively, when a PC with Intel® Core i7 2 GHz processor and 16 GB of RAM, respectively, when the block size is 7 pixels. So, the time required to decrypt the image in Shui *et al*.'s [17] algorithm through brute-force attack is years, while it is 2.5424× 10^{44} is 7.0755 ×10^{14} years in the proposed algorithm.

Visually, Figure 11 compares the encrypted images in Shui *et al.* [17] and proposed algorithms when the block size is 3 and 15 with N_C in the proposed algorithm set such that 75% of the blocks are complemented. Both algorithms were capable of scrambling the image content in an unperceivable manner with higher visual confusion observed for smaller blocks. However, the proposed algorithm was capable of changing the major tone in the image due to complementing the pixels in some of the blocks.

Quantitatively, Table 1 lists the PSNR, entropy, NPCR and UACI metrics of the encrypted six test images for both algorithms when the block size is 7 and the same permutation key K_P is used to permute the Hilbert blocks in both algorithms while a different key is used to permute the pixels with the blocks in Shui *et al*.'s [17] algorithm. In the proposed algorithm, N_C is specified such that 75% of the blocks are complemented. Investigating the numbers in Table 1 reveals that the PSNR values of the encrypted image in the proposed algorithm are always lower than those in Shui *et al*.'s [17] algorithm. This reflects lower similarity between the encrypted and original images in the proposed algorithm. The same conclusion can be made by considering the NPCR and UACI metrics. The numbers in Table 1 shows the superiority of the proposed algorithm which produced encrypted images with higher values for these metrics for all test images.

As for the entropy, which measures the randomness in the encrypted image, the values of Shui *et al*.'s [17] algorithm are effectively the entropy value of the original cover image since this algorithm only permutes the blocks and the pixels within the blocks without affecting their values during encryption. On the other hand, the proposed algorithm does not only shift the columns and permute the blocks, but it also complements the pixels' values in a randomly selected blocks using N_C and K_C keys. This has the effect of changing a subset of the pixels in the image; thus increasing the entropy as shown in Table 1.

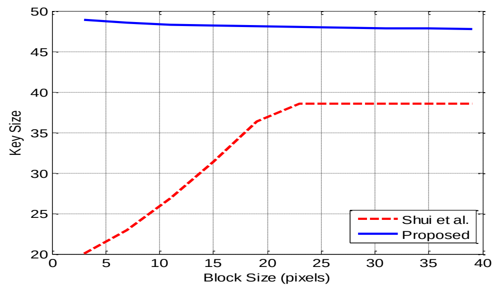
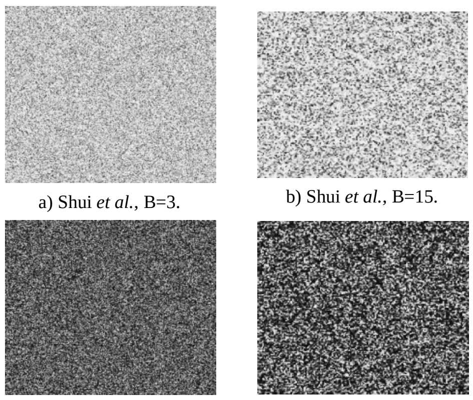


Figure 10. Key size as function of block size in both algorithms (log scale).



a) Shui *et al.*, B=3. b) Shui *et al.*, B=15.

c) Proposed, B=3. d) Proposed, B=15.

Figure 11. Encrypted images.

5. Conclusions

Privacy protection for images uploaded to cloud services has become an issue; especially when these images are subject to data embedding for different purposes. In this papers, we presented a separable reversible data hiding algorithm for encrypted images. The algorithm is essentially based on permutation in order to preserve the correlation between pixels which has direct impact on the embedding capacity. The contribution of the algorithm in terms of encryption lies in encrypting the image through specific scrambling of the image columns and blocks and complementing pixels' values in subset of the blocks which helps in embedding larger amount of data through the modification of prediction errors in the encrypted image. The results of the algorithm verified its ability in increasing embedding capacity as well as the security and privacy of the encrypted image. The security of the proposed algorithm can be further enhanced by considering classifying the blocks based on their texture and using blocks with low texture for data embedding while blocks with high texture are encrypted using Advanced Encryption Standard (AES) or Data Encryption Standard (DES).

Table 1. Values of security metrics for the test images in both algorithms.

	PSNR		**Entropy**		**NPCR**		**UACI**	
Image	**Prop.**	**Shui *et al.*[17]**	**Prop.**	**Shui *et al.*[17]**	**Prop.**	**Shui *et al.*[17]**	**Prop.**	**Shui *et al.*[17]**
Clock	5.87	9.98	7.12	6.71	99.37	97.92	42.85	22.89
Cat	5.66	9.91	5.79	5.19	96.43	86.09	43.95	24.34
Bird	9.89	16.57	6.91	6.22	99.48	98.18	27.74	11.41
Tank	17.47	17.47	5.68	5.05	98.30	95.31	9.44	8.57
Butterfly	12.18	12.57	6.86	6.61	99.15	98.76	20.11	19.07
Cameraman	9.15	9.27	7.40	7.05	99.54	98.98	28.40	26.33

References

[1] Alattar A., "Reversible Watermark Using Difference Expansion of Quads," *in Proceeding of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Motreal, pp. 377-380, 2004.

[2] Alsaidi A., Al-Lehaibi K., Alzahrani H., AlGhamdi M., and Gutub A., "Compression Multi-Level Crypto Stego Security of Texts Utilizing Colored Email Forwarding," *Journal of Computer Science and Computational Mathematics*, vol. 8, no. 3, pp. 33-42, 2018.

[3] Celik M., Sharma G., and Tekalp A., "Lossless Generalized-LSB Data Embedding," *IEEE transactions on image processing*, vol. 14, no. 2, pp. 253-266, 2005.

[4] Chen G., Mao Y., and Chui C., "A Symmetric Image Encryption Scheme Based on 3D Chaotic Cat Maps," *Chaos, Solitons and Fractals*, vol. 21, no. 3, pp. 749-761, 2004.

[5] Fridrich J., Goljan M., and Du R., "Lossless Data Embedding New Paradigm in Digital Watermarking," *Eurasip Journal on Advances in Signal Processing*, vol. 2002, no. 2, pp. 1-12, 2002.

[6] Hong W., Chen T., and Shiu C., "Reversible Data Hiding for High Quality Images Using Modification of Prediction Errors," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1833-1842, 2009.

[7] Hong W., Chen T., and Wu H., "An Improved Reversible Data Hiding in Encrypted Images Using Side Match," *IEEE Signal Processing Lett*, vol. 19, no. 4, pp. 199-202, 2012.

[8] Kumar R. and Jung K., "Enhanced Pairwise IPVO-Based Reversible Data Hiding Scheme Using Rhombus Context," *Information Sciences*, vol. 536, pp. 101-119, 2020.

[9] Long M., Zhao Y., Zhang X., and Peng F., "A Separable Reversible Data Hiding Scheme for Encrypted Images Based on Tromino Scrambling and Adaptive Pixel Value Ordering," *Signal Processing*, vol. 176, pp. 107703, 2020.

[10] Li X., Li J., and Li B., "High-Fidelity Reversible Data Hiding Scheme Based on Pixel-value-ordering and Prediction-Error Expansion," *Signal Processing*, vol. 93, no. 1, pp. 198-205, 2013.

[11] Ma K., Zhang W., Zhao X., Yu N., and Li F., "Reversible Data Hiding in Encrypted Images By Reserving Room Before Encryption," *IEEE Transactions on Information Forensics and security*, vol. 8, no. 3, pp. 553-562, 2013.

[12] Mao Y., Chen G., and Lian S., "A Novel Fast Image Encryption Scheme Based on 3D Chaotic Baker Maps," *International Journal of Bifurcation and chaos*, vol. 14, no. 10, pp. 3613-3624, 2003.

[13] Marawan M., AlShahwan F., Sifou F., Kartit A., and Ouhamne H., "Improving The Security of Cloud-Based Medical Image Storage," *Engineering Letters*, vol. 27, no. 1, 2019.

[14] Ni Z., Shi Y., Ansari N., and Su W., "Reversible Data Hiding," *IEEE Transactions on Circuits and Systems for video technology*, vol. 16, no. 3, pp. 354-362, 2006.

[15] Puech W., Chaumont M., and Strauss O., "A Reversible Data Hiding Method for Encrypted Images," *in Proceeding of the Security, Forensics, Steganography, and Watermarking of Multimedia Contents X*, pp. 534-542, 2008.

[16] Shi Y., Li X., Zhang X., Wu H., and Ma B., "Reversible Data Hiding: Advances in the Past Two Decades," *IEEE Access*, vol. 4, pp. 3210-3237, 2016.

[17] Shiu C., Chen Y., and Hong W., "Reversible Data Hiding in Permutation-based Encrypted Images with Strong Privacy," *KSII Transactions on Internet and Information Systems*, vol. 13, no. 2, pp. 1020-1042, 2019.

[18] The USC-SIPI Image Database [Online]. Available: http://sipi.usc.edu/database, Last Visited 2021.

[19] Thahab A., "A Novel Secure Video Steganography Technique Using Temporal Lifted Wavelet Transform and Human Vision Properties," *The International Arab Journal for Information Technology*, vol. 17, no. 2, pp. 147-153, 2020.

[20] Wang W., "A Reversible Data Hiding Algorithm Based on Bidirectional Difference Expansion," Multimedia Tools and Applications, vol. 79, no. 9, pp. 890- 896, 2020.

[21] Waterloo Image Repository [Online]. Available: http://links.uwaterloo.ca/Repository.html, Last Visited, 2021.

[22] Xu C., Zhang Y., and Gu Z., "A Novel Color Image Encryption Method Based on Sequence Cross Transformation and Chaotic Sequences," *Engineering Letters*, vol. 28, no. 4, 2020.

[23] Zhang X., "Reversible Data Hiding in Encrypted Image," *IEEE Signal Processing Letters*, vol. 18, no. 4, pp. 255-258, 2011.

[24] Zhang X., Qian Z., Feng G., and Ren Y., "Efficient Reversible Data Hiding in Encrypted Images," *Journal of Visual Communication and Image Representation*, vol. 25, no. 2, pp. 322-328, 2014.

Iyad Jafar received the B.S. degree in Electrical Engineering from The University of Jordan in 2001, the M.Sc. degree in Electrical Engineering from the Illinois Institute of Technology in 2004, and the Ph.D. degree in Computer Engineering from Wayne State University in 2008. He is currently working as a professor and acting chair in the Department of Computer Engineering at the University of Jordan. His research interests are in signal and image processing, pattern recognition, and computer networks.

Khalid Darabkh Received the PhD degree in Computer Engineering from the University of Alabama in Huntsville in 2007 with honors. He is currently a professor in the Computer Engineering Department at the University of Jordan. He authored and co-authored of at least a hundred eighty highly esteemed research articles. He is among World's Top 2% Scientists List compiled by Stanford University in 2020 and 2021. He serves on the Editorial Board of Telecommunication Systems, published by Springer, Computer Applications in Engineering Education, published by John Wiley & Sons, and Journal of High Speed Networks, published by IOS Press. Additionally, he serves as a TPC member of highly reputable IEEE conferences such as GLOBECOM, ICC, LCN, VTC-Fall, PIMRC, ISWCS, ATC, ICT, and IAEAC. He is engaged in research mainly on Internet of things, Software-defined networks, vehicular networks, flying ad-hoc networks, Fog networking, Full duplex cognitive radio networks, queuing systems and networks, multimedia transmission, channel coding, steganography and watermarking, as well as innovative and interactive learning environments.

Fahed Jubair graduated from Purdue University in 2014 with a Ph.D. degree in Electrical and Computer Engineering. He received his B.Sc. degree from the University of Jordan in 2007. Dr. Jubair is currently an assistant professor of Computer Engineering at the University of Jordan. His main research interests include optimizing compilers, parallel computing, heuristic algorithms, and machine learning.