






Research Article

Digital Image Watermarking for Image Integrity Verification and Tamper Correction

Anantha Rao Gottimukkala¹, Anita Pradhan¹, Naweem Kumar², Ashok Kumar Pradhan³, Ranjan K. Senapati⁴, Gandharba Swain^{1*}

¹Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, 522302, Andhra Pradesh, India

²School of Computer Science Engineering and Technology, Bennett University, Greater Noida, 201310, U.P., India

³Department of Computer Science and Engineering, SRM University-AP, 522240, Andhra Pradesh, India

⁴Department of ECE, VNR Vignana Jyothi Institute of Engineering & Technology, Bachupally, Nizampet (S.O), 500090, Hyderabad, Telangana, India

E-mail: gswain1234@gmail.com

Received: 19 December 2024; **Revised:** 3 January 2025; **Accepted:** 8 January 2025

Abstract: Images transmitted through internet can be easily tampered by the available image editing tools. This article proposes a Hamming code based watermarking approach for tamper localization and correction of images. The original image is divided into various blocks with 8 consecutive pixels. The 64 bits of the 8 pixels are arranged into an 8×8 matrix of bits. A modified (7,4) Hamming code (MHC) is applied on first 7 most significant bits (MSBs) of each row of the matrix. The first 4 MSBs are data bits. The next 3 bits are redundant bits. The watermark bits are calculated from the 4 MSBs and stored in 3 redundant bits. Furthermore, the column parity for the first 7 columns of the 8×8 matrix is computed and embedded in the least significant bits (LSBs) of the 7 rows. Thereafter the column parity of the first 7 bits of 8th column is stored in 8th bit location of 8th column. This technique can detect 1-bit error or 2-bit error if it occurs in one of the 8 pixels of the block. Experimental outcomes prove that this proposed scheme maintains 4.0 bits per pixel with 36.94 dB peak signal-to-noise ratio (PSNR) and 0.9781 structural similarity (SSIM).

Keywords: watermarking, tamper detection, data hiding, modified hamming code, logistic map

MSC: 65L05, 34K06, 34K28

1. Introduction

Digital image watermarking techniques are used for various purposes including copyright protection. Attackers can easily tamper a watermarked image (WI) using various photo editing tools [1]. Hence, the research on tamper detection and correction has been evolved. This is a great motivation for security researchers. A data hiding scheme can be assessed by four quality metrics, (i) hiding capacity (HC), (ii) imperceptibility, (iii) robustness, and (iv) security. Copyright protection and tamper detection are the important objectives of watermarking techniques. Cropping, rotation, increasing brightness, collusion, addition of salt and pepper (S and P) noise etc. are various attacks possible by attackers to disturb the watermark in a WI [2]. The watermarking schemes are of various types as discussed below [3]. As per the human perception, the

Copyright ©2025 Gandharba Swain, et al.

DOI: <https://doi.org/10.37256/cm.6220256272>

This is an open-access article distributed under a CC BY license
(Creative Commons Attribution 4.0 International License)

<https://creativecommons.org/licenses/by/4.0/>

watermarking schemes are 2 types, (i) visible, and (ii) invisible. Further, the invisible watermarking schemes are 2 types, (i) fragile, and (ii) robust. As per domain, the watermarking schemes are 2 types, (i) spatial, and (ii) frequency. As per documents, the watermarking schemes are 4 types, (i) video, (ii) audio, (iii) text, and (iv) image. As per applications, the watermarking schemes are 2 types, (i) source based, and (ii) destination based. Spatial domain techniques like block-wise watermarking, and chaotic map (CM) related watermarking are very well-known. These are discussed in the following paragraphs.

In spatial domain, the watermarking techniques use block by block or pixel by pixel for watermark embedding. CM can be used to improve the security of watermarks. Singh and Singh [4] developed a block-based approach. They used 2×2 magnitude blocks. From the 4 higher bit planes (HBPs) of 4 pixels, created a pair of recovery bits (RBs) and a pair of WBs. These generated bits are hidden in LSB positions. This technique identifies tampered blocks properly. Cao et al. [5] also developed a block-based technique. From m HBPs of a block, they generated WBs and RBs and hid in LSBs. The algorithm is secured because m is variable. Gull et al. [6] developed a 4×4 size block-based technique. They reset the 2 LSBs in all pixels as zeros and then calculated the average value of pixels in a block. The WBs are generated from these 8 bits of the block average. The average value is hidden in upper half block and WBs are stored in lower half block. Here tamper detection is happening properly. Feng et al. [7] discovered a 2×2 block-based technique. They applied Arnold map (AM) to improve the security and LSBs are replaced by the WBs. Bhalerao et al. [8] improved the security of their watermarking scheme using cryptographic hash function.

In Gul and Ozturk's [9] watermarking technique, 16 primary blocks are created from the image. A group of 4 blocks are partner blocks. A partner block is again splitted into 4×4 blocks. The WBs and RBs of a block are calculated in association with its partner blocks. In Sinhal et al.'s [10] color image watermarking technique, the block size is 2×4 . Each element of a block is a color component. From the 6 MSBs of these 8 color components, 12 WBs are created and hidden in LSBs. This technique achieves 99% tamper localization and 80% tamper correction. The above said techniques use non-overlapped blocks. An overlapped 3×3 size block-based technique is developed by Qin et al. [11]. The WBs and RBs are generated from 6 MSBs of all the pixels. WBs are embedded in middle pixel LSBs and RBs are hidden in other pixels. This technique efficiently corrects the tampered blocks. Kosuru et al. developed a Merkle tree-based approach [12]. Here WBs are computed from Merkle tree root. Although this approach detects tampered areas, but cannot correct them.

In Rawat and Raman's scheme [13] the image is scrambled by AM, the WBs and logistic map (LM) bits are XORed and the output bits are embedded in LSBs. In this scheme the tampered locations could not be identified. Botta et al. [14] used 7 HBPs to derive WBs and then hid them in LSBs. Prasad and Pal [15] calculated WBs from 5 MSBs and hidden in the pixel plying LM, remainder division and shift operations. In this scheme tampered blocks are identified accurately. Sahu [16] also plied LM for watermark embedding and tamper identification. This scheme sustains against various attacks. In Tong et al.'s [17] 2×2 block-based scheme the image is scrambled by CM, the WBs are computed from MSBs and stored in LSBs. This technique identifies and corrects the tampered blocks effectively. In Nazari et al.'s [18] 2×2 block-based scheme also the image is scrambled using CM, from MSBs of 4 pixels a variable length information array is generated and stored in LSBs. The security in these 2 techniques is improved by inclusion of CM. Sreenivas and Kamakshiprasad [19] also used 2×2 block-based watermarking scheme, wherein 12 WBs are computed using CM. Further these 12 bits are decreased to 4 to make space for RBs. LSBs are consumed to hold RBs and WBs. WBs and RBs are embedded in LSBs. Here tampered blocks are identified properly.

2. Related literature study and author's contribution

2.1 Related literature study

In 1950 Hamming code was initiated [20]. There can be many variants of this code, out of which (7,4) size is frequently used. The (7,4) size code considers 4 bits for data and 3 bits as parity/redundant. As depicted in Figure 1(a), this code adds 3 redundant bits r_1 , r_2 and r_3 to 4 data bits d_1 , d_2 , d_3 and d_4 . The redundant bits are computed from data bits. The bit r_1 is computed by exclusive or (XOR) operation of bits d_1 , d_2 , and d_4 . The bit r_2 is computed by XOR

operation of bits d_1 , d_3 , and d_4 . The bit r_3 is computed by XOR operation of bits d_2 , d_3 , and d_4 . Hamming code catches error bits by ensuring that each redundant bit along with data bits should possess an even parity. At the receiver, if parity is odd, then error is identified.

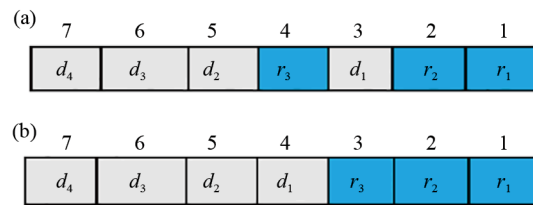


Figure 1. (a) The (7,4) Hamming code, (b) modified Hamming code

Chan and Chang [21] used rotation, torus automorphism and Hamming code for image authentication. First, they used Hamming code for computing parity check bits in each pixel. Then, they used Torus automorphism on parity bits to decide the suitability of a pixel to conceal the parity bits. The third activity is bit rotation operation before embedding to improve the security. It is claimed that tampered pixels could be rectified. Furthermore, it is declared that this method can eliminate burst errors. Authors in [22] computed the WBs from 4 MSBs using Hamming code. These WBs are then camouflaged in LSBs by simple bit substitution method. Furthermore, they used LM to improve the security of WBs. It has been claimed that the technique detects the tampered regions properly. From experimental results in Figure 1 it is evident that PSNR is 37 dB, it is acceptable. Wang et al. did a 3D object verification technique using Hamming code [23]. In this technique, watermark is created applying Hamming code and embedded using LSB substitution. During extraction also, the hamming code has been used for watermark verification. This scheme achieves less distortion and more security. Medical images are prone to errors when they are transmitted in Internet. Islam et al. [24] proposed a (8, 4) Hamming code-based approach to solve this problem. Furthermore, they implemented in a graphics processing unit (GPU) to make faster processing as per the demand of the real time requirement.

Trivedy and Pal [25] developed a watermarking scheme for integrity verification. They used LM to produce a key matrix and WBs. These WBs are stored in pixels after changing their value by ± 1 or ± 0 . This technique also detects the tampered regions properly. This technique produces imperceptible watermarked image and tolerant to various malicious attacks. But it has lower tamper detection capability. Authors in [26] also plied LM and Hamming code. In this scheme, the 3 WBs in a 2-pixel block are computed from 2 MSBs of the 2 pixels by plying LM and Hamming code. Further, these WBs have been embedded at 6 LSBs of the 2 pixels by applying pixel value differencing (PVD). This technique gives improved PSNR, and structural similarity (SSIM). Prasad and Pal [27] too did tamper detection at pixel level. In this technique a generator matrix is deployed to compute 3 WBs from 4 MSBs and the WBs are stored in LSBs. Due to the inclusion of LM, the security is improved. Authors in [28] developed a reversible scheme to verify integrity with tamper correction. They used Hamming code with regional binary pattern. At the extraction side Hamming code has been applied for tamper correction. This technique is secured against many attacks. This technique can also recognize tampered areas and check the image ownership. The advantages of this technique are: good perceptual quality, and high embedding capacity.

In literature, we can find that some data hiding schemes uses error detecting codes for detecting error bits at the extracted data at receiver side. Authors in [29] used Hamming code with odd parity to compute error positions and correct them. Here HC is 1 bits per pixel (bpp). Nguyen et al. [30] also used (5, 3) Hamming in their reversible data hiding. They have used 5-pixel blocks. Here a revised (5, 3) Hamming code has been introduced to recognize the suitable modification positions for hiding the message bits. The HC is very low i.e., only 1.2 bpp. The embedding positions play a vital role as a secret key and improves the security. Chen et al. [31] introduced a data hiding approach using Huffman coding, wherein the Huffman code compresses 4 MSBs and LSBs are stored in vacated MSB planes. Hence the vacated LSB planes can be utilized to hide data. This approach achieved higher HC. Authors in [32] developed a scheme which

is fragile in nature. The WBs are generated using parity bits in a tamper rectifying code and concealed in frequency sub-bands. The WBs have been plied as authentication bits for error correction.

Chen and Chang [33] used (7,4) Hamming code for achieving reversibility in data hiding over encrypted images. They embedded secret data in LSBs and designed prediction scheme to predict errors at MSBs. The MSBs can be modified to store the authentication bits which can be used to find errors in LSBs at the time of extraction. It is claimed that this technique achieves higher HC, and imperceptible stego-image (SI). But, embedding and extraction could be performed in very less time, so this technique suitable for real time cloud application. Wang et al. [34] performed data hiding using LSB substitution and Hamming code. They hid few bits at smoother regions and more bits at textured regions. They claimed that they achieved higher HC and imperceptible SI.

Wu et al. [35] proposed data hiding approach using an image transformation algorithm which uses (7,4) Hamming code. Here the original image (OI) is passed through the transformation phase to create the cover image. Thereafter the classified bits are hidden in OI by altering and flipping the bits in OI based on the syndrome derived using Hamming code. This technique also achieves higher HC. Basically, the data hiding techniques aims at higher HC, lower distortion and greater similarity between OI and SI. Recently, Khadse and Swain [36] did error detection and rectification over retrieved data from the SI at the receiver. Furthermore Kosuru et al. [37] used Hamming code for improving the accuracy in error detection and correction in extracted data.

2.2 Author's contribution

Many authors proposed watermarking techniques merely for tamper detection. But, correcting the tampered pixels is a very challenging and interesting task as well. This article proposes a watermarking technique based on modified version of Hamming code and parity bit (MHCPB). The important contributions are as listed below.

Figure 1a depicts (7,4) Hamming code. Here the 3 redundant bits known as parity bits, and these are computed from the 4 data bits. The parity bit r_1 is parity of bits at positions (3, 5, 7), r_2 is the parity bit of bits at positions (3, 6, 7) and r_3 is the parity bit of bits at positions (5, 6, 7). We modify these 3 calculations and keep the parity bits on LSB side as in Figure 1b, modified Hamming code (MHC). Note that as the positions of the 3 redundant bits are changed, their values are computed differently, r_1 is parity of bits at positions (4, 5, 7), r_2 is the parity of bits at positions (4, 6, 7) and r_3 is the parity of bits at positions (5, 6, 7).

We consider 8 pixels as a block and arrange the 64 bits into an 8×8 matrix of bits to apply the MHC on 7 bits of each row in order to identify and correct one-bit error in a row. Furthermore, we also compute parity bits column wise to locate and correct 2-bit error in one of the 8 rows of the matrix.

The detection and correction can be done very accurately. We have developed 2 correction mechanisms, (i) 1-bit error correction, and (ii) 2-bit error correction. HC, PSNR, and SSIM are not compromised.

Figure 1a The (7,4) Hamming code, Figure 1b modified Hamming code

The quality parameters such as SSIM, HC, and PSNR are not compromised to achieve the error correction.

3. The proposed MHCPB based watermarking technique

The architecture of MHCPB watermark hiding approach is depicted in Figure 2a. Similarly, the architecture of MHCPB watermark extraction and tamper detection procedure is depicted in Figure 2b. Furthermore, the detailed embedding procedure is mentioned in Section 3.1 and the detailed extraction procedure is mentioned in Section 3.2.

Figure 2a Architecture of watermark embedding procedure, Figure 2b Architecture of watermark extraction procedure

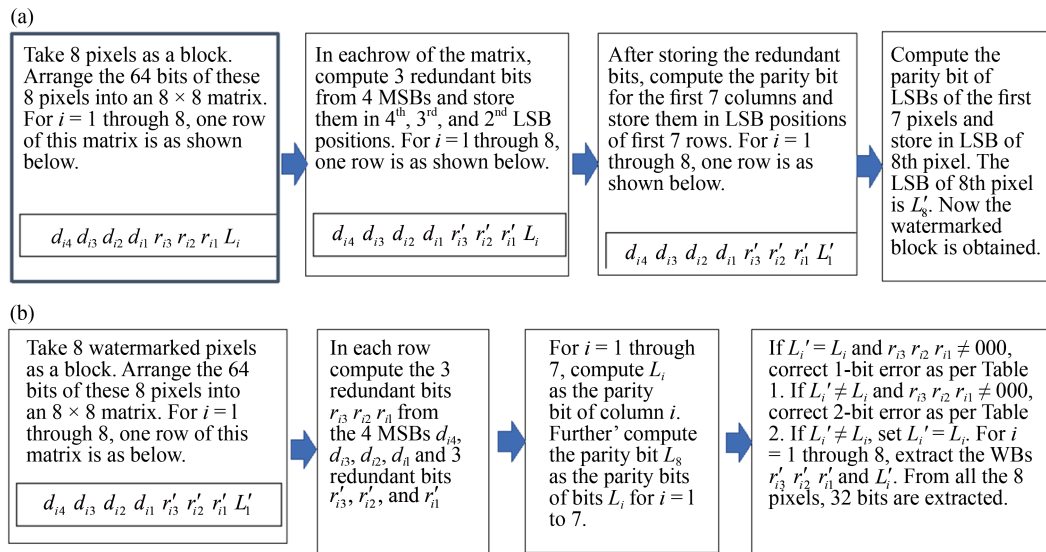


Figure 2. (a) Architecture of watermark embedding procedure, (b) Architecture of watermark extraction procedure

3.1 Watermark embedding

Divide the image into 8-pixel blocks in one of the following four ways, (i) 1×8 size block, pixels from one row and eight consecutive columns, (ii) 8×1 size block, pixels from one column and eight consecutive rows, (iii) 2×4 size block, pixels from 2 consecutive rows and 4 consecutive columns, and (iv) 4×2 size blocks, pixels from 4 consecutive rows and 2 consecutive columns. The watermark is embedded by the following steps. The eight pixels are represented in Figure 3.

Step 1: Each pixel is a byte or 8-bits. For $i = 1$ through 8, the bits of a pixel P_i can be designated as in Figure 4a.

Step 2: For $i = 1$ through 8, compute r'_{i1} , r'_{i2} , and r'_{i3} by Equation (1).

$$r'_{i1} = XOR(d_{i1}, d_{i2}, d_{i4}), r'_{i2} = XOR(d_{i1}, d_{i3}, d_{i4}), r'_{i3} = XOR(d_{i2}, d_{i3}, d_{i4}) \quad (1)$$

Step 3: Replace r_{i1} by r'_{i1} , r_{i2} by r'_{i2} , and r_{i3} by r'_{i3} . As a result, P_i is changed to P'_i as shown in Figure 4(b). Note that Figure 4a, b represent eight pixels for $i = 1$ through 8.

d_{14}	d_{13}	d_{12}	d_{11}	r_{13}	r_{12}	r_{11}	L_1
d_{24}	d_{23}	d_{22}	d_{21}	r_{23}	r_{22}	r_{21}	L_2
d_{34}	d_{33}	d_{32}	d_{31}	r_{33}	r_{32}	r_{31}	L_3
d_{44}	d_{43}	d_{42}	d_{41}	r_{43}	r_{42}	r_{41}	L_4
d_{54}	d_{53}	d_{52}	d_{51}	r_{53}	r_{52}	r_{51}	L_5
d_{64}	d_{63}	d_{62}	d_{61}	r_{63}	r_{62}	r_{61}	L_6
d_{74}	d_{73}	d_{72}	d_{71}	r_{73}	r_{72}	r_{71}	L_7
d_{84}	d_{83}	d_{82}	d_{81}	r_{83}	r_{82}	r_{81}	L_8

Figure 3. The 64 bits of 8 pixels in an 8×8 matrix

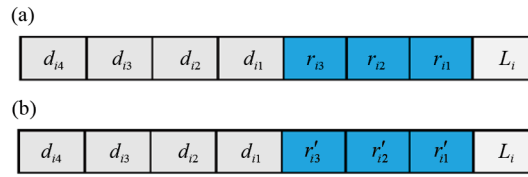


Figure 4. (a) The eight bits of a pixel, P_i , and (b) The eight bits of a pixel, P'_i

Step 4: For $i = 1$ to 8, calculate L'_i using Equations (2-9).

$$L'_1 = XOR \text{ of bits } d_{i4}, \text{ for } i = 1 \text{ to } 8 \quad (2)$$

$$L'_2 = XOR \text{ of bits } d_{i3}, \text{ for } i = 1 \text{ to } 8 \quad (3)$$

$$L'_3 = XOR \text{ of bits } d_{i2}, \text{ for } i = 1 \text{ to } 8 \quad (4)$$

$$L'_4 = XOR \text{ of bits } d_{i1}, \text{ for } i = 1 \text{ to } 8 \quad (5)$$

$$L'_5 = XOR \text{ of bits } r'_{i3}, \text{ for } i = 1 \text{ to } 8 \quad (6)$$

$$L'_6 = XOR \text{ of bits } r'_{i2}, \text{ for } i = 1 \text{ to } 8 \quad (7)$$

$$L'_7 = XOR \text{ of bits } r'_{i1}, \text{ for } i = 1 \text{ to } 8 \quad (8)$$

$$L'_8 = XOR \text{ of bits } L'_i, \text{ for } i = 1 \text{ to } 7 \quad (9)$$

Step 5: For $i = 1$ through 8, replace L_i of Figure 4b by L'_i in P'_i . As a result, we get watermarked pixel P_i^w as in Figure 5. Thus, we got the 8-pixels of the watermarked block as in Figure 6.



Figure 5. The eighth bits of watermarked pixel, P_i^w

d_{14}	d_{13}	d_{12}	d_{11}	r'_{13}	r'_{12}	r'_{11}	L'_1
d_{24}	d_{23}	d_{22}	d_{21}	r'_{23}	r'_{22}	r'_{21}	L'_2
d_{34}	d_{33}	d_{32}	d_{31}	r'_{33}	r'_{32}	r'_{31}	L'_3
d_{44}	d_{43}	d_{42}	d_{41}	r'_{43}	r'_{42}	r'_{41}	L'_4
d_{54}	d_{53}	d_{52}	d_{51}	r'_{53}	r'_{52}	r'_{51}	L'_5
d_{64}	d_{63}	d_{62}	d_{61}	r'_{63}	r'_{62}	r'_{61}	L'_6
d_{74}	d_{73}	d_{72}	d_{71}	r'_{73}	r'_{72}	r'_{71}	L'_7
d_{84}	d_{83}	d_{82}	d_{81}	r'_{83}	r'_{82}	r'_{81}	L'_8

Figure 6. The 64 bits of eight pixels of the watermarked block

Figure 7 depicts the flow diagram for watermark embedding procedure for quick reference by the readers.

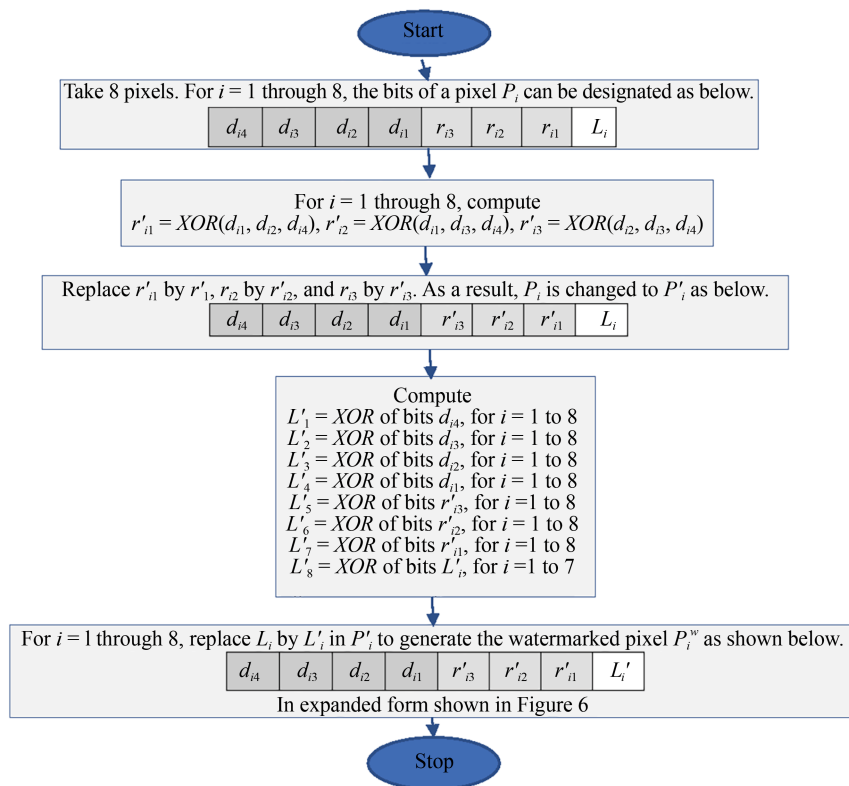


Figure 7. Flow diagram for watermark embedding procedure

3.2 Watermark extraction, tamper detection, and correction

Divide the watermarked image into 8-pixel blocks in same way as we did in watermark embedding procedure. The watermark retrieval and tamper correction are illustrated in below steps.

Step 1: For $i = 1$ through 8, the bits of a watermarked pixel P_i^w are as designated in Figure 5.

Step 2: For $i = 1$ through 8, compute r_{i1} , r_{i2} and r_{i3} by Equation (10).

$$r_{i1} = \text{XOR}(r'_{i1}, d_{i1}, d_{i2}, d_{i4}), \quad r_{i2} = \text{XOR}(r'_{i2}, d_{i1}, d_{i3}, d_{i4}), \quad r_{i3} = \text{XOR}(r'_{i3}, d_{i2}, d_{i3}, d_{i4}) \quad (10)$$

Step 3: Compute L_i , for $i = 1$ to 8 by Equations (11)-(18).

$$L_1 = \text{XOR of bits } d_{i4}, \text{ for } i = 1 \text{ to } 8 \quad (11)$$

$$L_2 = \text{XOR of bits } d_{i3}, \text{ for } i = 1 \text{ to } 8 \quad (12)$$

$$L_3 = \text{XOR of bits } d_{i2}, \text{ for } i = 1 \text{ to } 8 \quad (13)$$

$$L_4 = \text{XOR of bits } d_{i1}, \text{ for } i = 1 \text{ to } 8 \quad (14)$$

$$L_5 = \text{XOR of bits } r'_{i3}, \text{ for } i = 1 \text{ to } 8 \quad (15)$$

$$L_6 = \text{XOR of bits } r'_{i2}, \text{ for } i = 1 \text{ to } 8 \quad (16)$$

$$L_7 = \text{XOR of bits } r'_{i1}, \text{ for } i = 1 \text{ to } 8 \quad (17)$$

$$L_8 = \text{XOR of bits } L'_i, \text{ for } i = 1 \text{ to } 7 \quad (18)$$

Step 4: For $i = 1$ through 8, if $r_{i3}r_{i2}r_{i1} = 000$, it means that there is no error in the first 7 bits of all the pixels. Suppose there is no error in 7 bits of all the pixels, but for some value of i , $L_i \neq L'_i$, then there is error in LSB bit. To correct it set $L'_i = L_i$.

Step 5: For $i = 1$ to 8, if $L_i = L'_i$, but for some value of i , $r_{i3}r_{i2}r_{i1} \neq 000$ then there exists single bit error in that pixel block. Find the tampered bit location and correct it using Table 1. In Table 1, there are seven rows and three columns. The first column specifies a condition. If a certain condition out of seven conditions satisfies, then the error position is found in second column of same row. The corrective action is mentioned in third column of same row. For example, suppose the condition $r_{i3}r_{i2}r_{i1} = 101$ satisfies, then the error is at d_{i2} . The corrective action to be taken is “invert d_{i2} ”. Furthermore, this correction procedure is also described by means of an if-else statement in Figure 8.

Table 1. One-bit error identification and correction

Condition	Error position	Correction
if $r_{i3}r_{i2}r_{i1} = 111$	error at d_{i4}	invert d_{i4}
if $r_{i3}r_{i2}r_{i1} = 110$	error at d_{i3}	invert d_{i3}
if $r_{i3}r_{i2}r_{i1} = 101$	error at d_{i2}	invert d_{i2}
if $r_{i3}r_{i2}r_{i1} = 100$	error at r'_{i3}	invert r'_{i3}
if $r_{i3}r_{i2}r_{i1} = 011$	error at d_{i1}	invert d_{i1}
if $r_{i3}r_{i2}r_{i1} = 010$	error at r'_{i2}	invert r'_{i2}
if $r_{i3}r_{i2}r_{i1} = 001$	error at r'_{i1}	invert r'_{i1}

Tamper correction logic-1
if $r_{i3} r_{i2} r_{i1} = 111$, then there is error at d_{i4} , so invert d_{i4} else if $r_{i3} r_{i2} r_{i1} = 110$, then there is error at d_{i3} , so invert d_{i3} else if $r_{i3} r_{i1} r_{i1} = 101$, then there is error at d_{i2} , so invert d_{i2} else if $r_{i3} r_{i2} r_{i1} = 100$, then there is error at r'_{i3} , so invert r'_{i3} else if $r_{i3} r_{i2} r_{i1} = 011$, then there is error at d_{i1} , so invert d_{i1} else if $r_{i3} r_{i2} r_{i1} = 010$, then there is error at r'_{i2} , so invert r'_{i2} else if $r_{i3} r_{i2} r_{i1} = 001$, then there is error at r'_{i1} , so invert r'_{i1} else // no error case, no error correction required end

Figure 8. One-bit tamper correction procedure

Tamper correction logic-2
if $r_{i3} r_{i2} r_{i1} = 001$, if $L_1 \neq L'_1$ and $L_2 \neq L'_2$, then there is error at d_{i4} and d_{i3} , so invert both the bits d_{i4} and d_{i3} , end if $L_3 \neq L'_3$ and $L_5 \neq L'_5$, then there is error at d_{i2} and r'_{i3} , so invert both the bits d_{i2} and r'_{i3} , end if $L_4 \neq L'_4$ and $L_6 \neq L'_6$, then there is error at d_{i1} and r'_{i2} , so invert both the bits d_{i1} and r'_{i2} , end else if $r_{i3} r_{i2} r_{i1} = 010$, if $L_1 \neq L'_1$ and $L_3 \neq L'_3$, then there is error at d_{i4} and d_{i2} , so invert both the bits d_{i4} and d_{i2} , end if $L_2 \neq L'_2$ and $L_5 \neq L'_5$, then there is error at d_{i3} and r'_{i3} , so invert both the bits d_{i3} and r'_{i3} , end if $L_4 \neq L'_4$ and $L_7 \neq L'_7$, then there is error at d_{i1} and r'_{i1} , so invert both the bits d_{i1} and r'_{i1} , end else if $r_{i3} r_{i2} r_{i1} = 011$, if $L_1 \neq L'_1$ and $L_5 \neq L'_5$, then there is error at d_{i4} and r'_{i3} , so invert both the bits d_{i4} and r'_{i3} , end if $L_2 \neq L'_2$ and $L_3 \neq L'_3$, then there is error at d_{i3} and d_{i2} , so invert both the bits d_{i3} and d_{i2} , end if $L_6 \neq L'_6$ and $L_7 \neq L'_7$, then there is error at r'_{i2} and r'_{i1} , so invert both the bits r'_{i2} and r'_{i1} , end else if $r_{i3} r_{i2} r_{i1} = 100$, if $L_1 \neq L'_1$ and $L_4 \neq L'_4$, then there is error at d_{i4} and d_{i1} , so invert both the bits d_{i4} and d_{i1} , end if $L_2 \neq L'_2$ and $L_6 \neq L'_6$, then there is error at d_{i3} and r'_{i2} , so invert both the bits d_{i3} and r'_{i2} , end if $L_3 \neq L'_3$ and $L_7 \neq L'_7$, then there is error at d_{i2} and r'_{i1} , so invert both the bits d_{i2} and r'_{i1} , end else if $r_{i3} r_{i2} r_{i1} = 101$, if $L_1 \neq L'_1$ and $L_6 \neq L'_6$, then there is error at d_{i4} and r'_{i2} , so invert both the bits d_{i4} and r'_{i2} , end if $L_2 \neq L'_2$ and $L_4 \neq L'_4$, then there is error at d_{i3} and d_{i1} , so invert both the bits d_{i3} and d_{i1} , end if $L_5 \neq L'_5$ and $L_7 \neq L'_7$, then there is error at r'_{i3} and r'_{i1} , so invert both the bits r'_{i3} and r'_{i1} , end else if $r_{i3} r_{i2} r_{i1} = 110$, if $L_3 \neq L'_3$ and $L_4 \neq L'_4$, then there is error at d_{i2} and d_{i1} , so invert both the bits d_{i2} and d_{i1} , end if $L_5 \neq L'_5$ and $L_6 \neq L'_6$, then there is error at r'_{i3} and r'_{i2} , so invert both the bits r'_{i3} and r'_{i2} , end if $L_1 \neq L'_1$ and $L_7 \neq L'_7$, then there is error at d_{i4} and r'_{i1} , so invert both the bits d_{i4} and r'_{i1} , end else if $r_{i3} r_{i2} r_{i1} = 111$, if $L_2 \neq L'_2$ and $L_7 \neq L'_7$, then there is error at d_{i3} and r'_{i1} , so invert both the bits d_{i3} and r'_{i1} , end if $L_3 \neq L'_3$ and $L_6 \neq L'_6$, then there is error at d_{i2} and r'_{i2} , so invert both the bits d_{i2} and r'_{i2} , end if $L_4 \neq L'_4$ and $L_5 \neq L'_5$, then there is error at d_{i1} and r'_{i3} , so invert both the bits d_{i1} and r'_{i3} , end else // no error, no correction is required, end

Figure 9. Two-bit tamper correction procedure

Step 6: For $i = 1$ to 8, if for any value of i , $L_i \neq L'_i$ and $r_{i3}r_{i2}r_{i1} \neq 000$, then assuming that the LSBs in pixels (i.e., L'_i for $i = 1$ through 8) are not tampered and only 2 bits in a single pixel out of 8 pixels are tampered, we can find the error locations and correct as per Table 2 or Figure 9. There are four columns in Table 2. The first column specifies 7 conditions, and the second column specifies 3 sub-conditions for each condition mentioned in column 1. If a certain condition is satisfied, then out of 3 sub-conditions one will be satisfied. The error position is noted in third column of same row, and the corrective action is mentioned in fourth column of the same row. For example, suppose the condition $r_{i3}r_{i2}r_{i1} = 110$ is satisfied, and out of 3 sub-conditions, the second one $L_5 \neq L'_5$ and $L_6 \neq L'_6$ is satisfied, then the error is at r'_{i3} and r'_{i2} . The corrective action to be taken is “invert both the bits r'_{i3} and r'_{i2} ”. Figure 9 mentions the same correction mechanism in a procedural way using nested if-else structure. In this if-else ladder statement, under each main condition there are 3 sub-conditions nested. For each nested sub-condition, the correction over 2 error bits are mentioned. So that there is a total of 21 sub-conditions under these 7 conditions. This implies that there are 21 possible ways of 2-bit error detection and correction.

Step 7: For $i = 1$ through 8, if $L_i = L'_i$ and $r_{i3}r_{i2}r_{i1} = 000$, then there exists no error in pixel P_i^w . For $i = 1$ through 8, extract the watermarked bits r'_{i3} , r'_{i2} , r'_{i1} and L'_i . From all the eight pixels 32 bits are extracted.

Figure 10 depicts the above procedure for tamper correction and watermark bits extraction in a flow diagram for quick reference by the readers.

Table 2. Error position identification and correction for 2-bit error in any one of the 8 pixels

Condition	Sub-condition	Error positions	Corrective action
if $r_{i3}r_{i2}r_{i1} = 001$	if $L_1 \neq L'_1$ and $L_2 \neq L'_2$	error at d_{i4} and d_{i3}	invert both the bits d_{i4} and d_{i3}
	if $L_3 \neq L'_3$ and $L_5 \neq L'_5$	error at d_{i2} and r'_{i3}	invert both the bits d_{i2} and r'_{i3}
	if $L_4 \neq L'_4$ and $L_6 \neq L'_6$	error at d_{i1} and r'_{i2}	invert both the bits d_{i1} and r'_{i2}
if $r_{i3}r_{i2}r_{i1} = 010$	if $L_1 \neq L'_1$ and $L_3 \neq L'_3$	error at d_{i4} and d_{i2}	invert both the bits d_{i4} and d_{i2}
	if $L_2 \neq L'_2$ and $L_5 \neq L'_5$	error at d_{i3} and r'_{i3}	invert both the bits d_{i3} and r'_{i3}
	if $L_4 \neq L'_4$ and $L_7 \neq L'_7$	error at d_{i1} and r'_{i1}	invert both the bits d_{i1} and r'_{i1}
if $r_{i3}r_{i2}r_{i1} = 011$	if $L_1 \neq L'_1$ and $L_5 \neq L'_5$	error at d_{i4} and r'_{i3}	invert both the bits d_{i4} and r'_{i3}
	if $L_2 \neq L'_2$ and $L_3 \neq L'_3$	error at d_{i3} and d_{i2}	invert both the bits d_{i3} and d_{i2}
	if $L_6 \neq L'_6$ and $L_7 \neq L'_7$	error at r'_{i2} and r'_{i1}	invert both the bits r'_{i2} and r'_{i1}
if $r_{i3}r_{i2}r_{i1} = 100$	if $L_1 \neq L'_1$ and $L_4 \neq L'_4$	error at d_{i4} and d_{i1}	invert both the bits d_{i4} and d_{i1}
	if $L_2 \neq L'_2$ and $L_6 \neq L'_6$	error at d_{i3} and r'_{i2}	invert both the bits d_{i3} and r'_{i2}
	if $L_3 \neq L'_3$ and $L_7 \neq L'_7$	error at d_{i2} and r'_{i1}	invert both the bits d_{i2} and r'_{i1}
if $r_{i3}r_{i2}r_{i1} = 101$	if $L_1 \neq L'_1$ and $L_6 \neq L'_6$	error at d_{i4} and r'_{i2}	invert both the bits d_{i4} and r'_{i2}
	if $L_2 \neq L'_2$ and $L_4 \neq L'_4$	error at d_{i3} and d_{i1}	invert both the bits d_{i3} and d_{i1}
	if $L_5 \neq L'_5$ and $L_7 \neq L'_7$	error at r'_{i3} and r'_{i1}	invert both the bits r'_{i3} and r'_{i1}
if $r_{i3}r_{i2}r_{i1} = 110$	if $L_3 \neq L'_3$ and $L_4 \neq L'_4$	error at d_{i2} and d_{i1}	invert both the bits d_{i2} and d_{i1}
	if $L_5 \neq L'_5$ and $L_6 \neq L'_6$	error at r'_{i3} and r'_{i2}	invert both the bits r'_{i3} and r'_{i2}
	if $L_1 \neq L'_1$ and $L_7 \neq L'_7$	error at d_{i4} and r'_{i1}	invert both the bits d_{i4} and r'_{i1}
if $r_{i3}r_{i2}r_{i1} = 111$	if $L_2 \neq L'_2$ and $L_7 \neq L'_7$	error at d_{i3} and r'_{i1}	invert both the bits d_{i3} and r'_{i1}
	if $L_3 \neq L'_3$ and $L_6 \neq L'_6$	error at d_{i2} and r'_{i2}	invert both the bits d_{i2} and r'_{i2}
	if $L_4 \neq L'_4$ and $L_5 \neq L'_5$	error at d_{i1} and r'_{i3}	invert both the bits d_{i1} and r'_{i3}

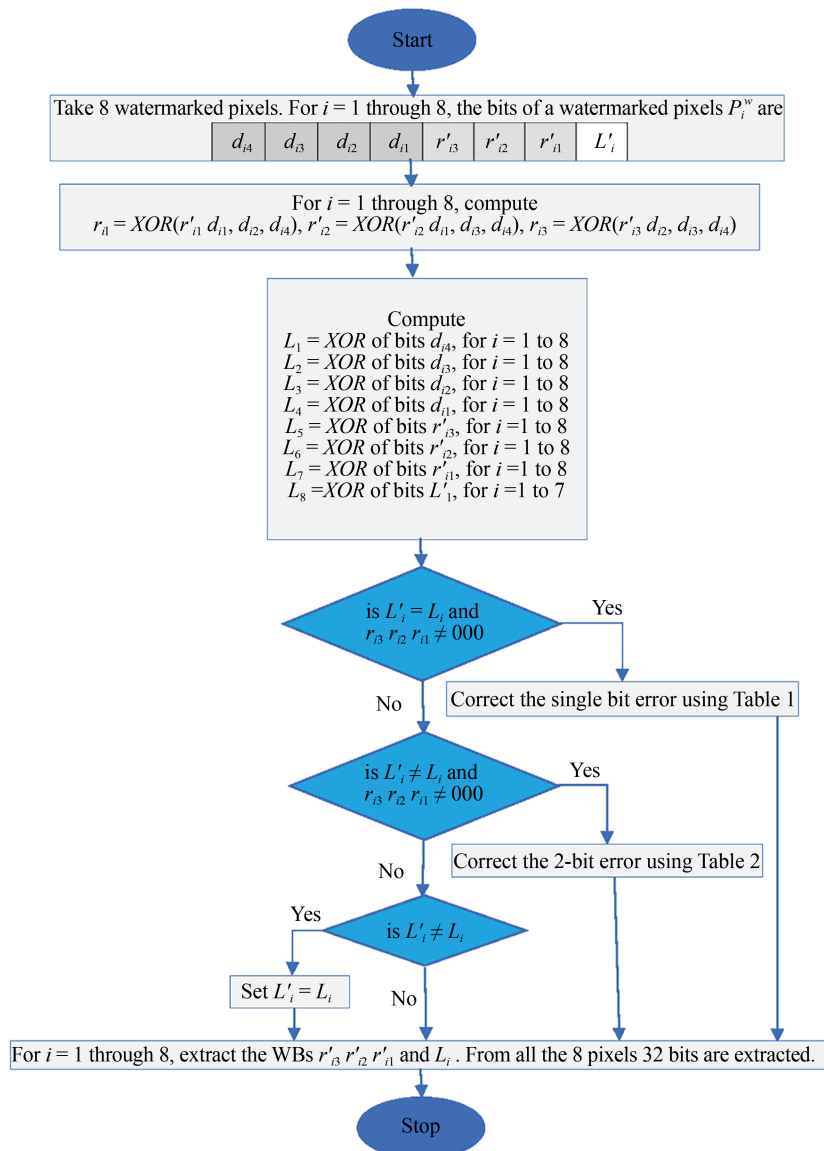


Figure 10. The flow diagram for tamper detection, tamper correction and watermark extraction

4. Examples of watermark embedding, and watermark extraction

Figure 11 describes an example of watermark embedding in a step-by-step manner. It takes a block of 8 pixels. Converts their pixel values to 8 binary bits and applies the embedding procedure in step by step manner. Figure 12 represents an example of watermark retrieval with no error. It takes a watermarked pixel block, converts to binary and extracts the embedded WBs. Here the watermarked block is not erroneous. Figure 13 represents an example of watermark extraction with error. It takes a watermarked pixel block, wherein one of the eight pixels has become erroneous during transit from sender to receiver. Here the error positions are detected and the correction logic is applied.

Example of watermark embedding

The values of 8 pixels of the block are 55, 56, 57, 58, 75, 76, 77, and 78. The embedding procedure has been applied on these 8 pixels step by step. Each pixel is converted to 8 bits binary and the embedding process is illustrated below.

Step 1

$P_1 = 55 = 00110111_2$	$d_{14} = 0$	$d_{13} = 0$	$d_{12} = 1$	$d_{11} = 1$	$r_{13} = 0$	$r_{12} = 1$	$r_{11} = 1$	$L_1 = 1$
$P_2 = 56 = 00111000_2$	$d_{24} = 0$	$d_{23} = 0$	$d_{22} = 1$	$d_{21} = 1$	$r_{23} = 1$	$r_{22} = 0$	$r_{21} = 0$	$L_2 = 0$
$P_3 = 57 = 00111001_2$	$d_{34} = 0$	$d_{33} = 0$	$d_{32} = 1$	$d_{31} = 1$	$r_{33} = 1$	$r_{32} = 0$	$r_{31} = 0$	$L_3 = 1$
$P_4 = 58 = 00111010_2$	$d_{44} = 0$	$d_{43} = 0$	$d_{42} = 1$	$d_{41} = 1$	$r_{43} = 1$	$r_{42} = 0$	$r_{41} = 1$	$L_4 = 0$
$P_5 = 75 = 01001011_2$	$d_{54} = 0$	$d_{53} = 1$	$d_{52} = 0$	$d_{51} = 0$	$r_{53} = 1$	$r_{52} = 0$	$r_{51} = 1$	$L_5 = 1$
$P_6 = 76 = 01001100_2$	$d_{64} = 0$	$d_{63} = 1$	$d_{62} = 0$	$d_{61} = 0$	$r_{63} = 1$	$r_{62} = 1$	$r_{61} = 0$	$L_6 = 0$
$P_7 = 77 = 01001101_2$	$d_{74} = 0$	$d_{73} = 1$	$d_{72} = 0$	$d_{71} = 0$	$r_{73} = 1$	$r_{72} = 1$	$r_{71} = 0$	$L_7 = 1$
$P_8 = 78 = 01001110_2$	$d_{84} = 0$	$d_{83} = 1$	$d_{82} = 0$	$d_{81} = 0$	$r_{83} = 1$	$r_{82} = 1$	$r_{81} = 1$	$L_8 = 0$

Step 2

$r'_{i1} = \text{XOR}(d_{i1}, d_{i2}, d_{i4})$	$r'_{11} = 0, r'_{21} = 0, r'_{31} = 0, r'_{41} = 0, r'_{51} = 0, r'_{61} = 0, r'_{71} = 0, r'_{81} = 0$
$r'_{i2} = \text{XOR}(d_{i1}, d_{i3}, d_{i4})$	$r'_{12} = 1, r'_{22} = 1, r'_{32} = 1, r'_{42} = 1, r'_{52} = 1, r'_{62} = 1, r'_{72} = 1, r'_{82} = 1$
$r'_{i3} = \text{XOR}(d_{i2}, d_{i3}, d_{i4})$	$r'_{13} = 1, r'_{23} = 1, r'_{33} = 1, r'_{43} = 1, r'_{53} = 1, r'_{63} = 1, r'_{73} = 1, r'_{83} = 1$

Step 3

For $i = 1$ through 8, replacing r_{i1} by r'_{i1} , r_{i2} by r'_{i2} , and r_{i3} by r'_{i3} , we get

$$P'_1 = (00111101)_2, P'_2 = (00111100)_2, P'_3 = (00111101)_2, P'_4 = (00111100)_2$$

$$P'_5 = (01001101)_2, P'_6 = (01001100)_2, P'_7 = (01001101)_2, P'_8 = (01001100)_2$$

Step 4

Using equations 2 to 9
 $L'_1 = \text{XOR}$ of bits d_{i4} , for $i = 1$ to 8, $L'_2 = \text{XOR}$ of bits d_{i3} , for $i = 1$ to 8
 $L'_3 = \text{XOR}$ of bits d_{i2} , for $i = 1$ to 8, $L'_4 = \text{XOR}$ of bits d_{i1} , for $i = 1$ to 8
 $L'_5 = \text{XOR}$ of bits r'_{i3} , for $i = 1$ to 8, $L'_6 = \text{XOR}$ of bits r'_{i2} , for $i = 1$ to 8
 $L'_7 = \text{XOR}$ of bits r'_{i1} , for $i = 1$ to 8, $L'_8 = \text{XOR}$ of bits L'_i , for $i = 1$ to 7

$$L'_1 = 0, L'_2 = 0$$

$$L'_3 = 0, L'_4 = 0$$

$$L'_5 = 0, L'_6 = 0$$

$$L'_7 = 0, L'_8 = 0$$

Step 5

For $i = 1$ to 8, replace L_i by L'_i in P'_i , we get watermarked pixel P_i^w

$$P_1^w = (00111100)_2, P_2^w = (00111100)_2, P_3^w = (00111100)_2, P_4^w = (00111100)_2$$

$$P_5^w = (01001100)_2, P_6^w = (01001100)_2, P_7^w = (01001100)_2, P_8^w = (01001100)_2$$

The watermarked pixel values are 60, 60, 60, 60, 76, 76, 76, and 76.

Figure 11. Example of watermark embedding

Example of watermark extraction with no error

The watermarked pixel values are 60, 60, 60, 60, 76, 76, 76, and 76. These values are converted to 8-bit binary and the extraction procedure is applied in a step by step manner. The operation in each step is as depicted below.

Step 1

$P_1^w = (00111100)_2$	$d_{14} = 0$	$d_{13} = 0$	$d_{12} = 1$	$d_{11} = 1$	$r'_{13} = 1$	$r'_{12} = 1$	$r'_{11} = 0$	$L'_1 = 0$
$P_2^w = (00111100)_2$	$d_{24} = 0$	$d_{23} = 0$	$d_{22} = 1$	$d_{21} = 1$	$r'_{23} = 1$	$r'_{22} = 1$	$r'_{21} = 0$	$L'_2 = 0$
$P_3^w = (00111100)_2$	$d_{34} = 0$	$d_{33} = 0$	$d_{32} = 1$	$d_{31} = 1$	$r'_{33} = 1$	$r'_{32} = 1$	$r'_{31} = 0$	$L'_3 = 0$
$P_4^w = (00111100)_2$	$d_{44} = 0$	$d_{43} = 0$	$d_{42} = 1$	$d_{41} = 1$	$r'_{43} = 1$	$r'_{42} = 1$	$r'_{41} = 0$	$L'_4 = 0$
$P_5^w = (01001100)_2$	$d_{54} = 0$	$d_{53} = 1$	$d_{52} = 0$	$d_{51} = 0$	$r'_{53} = 1$	$r'_{52} = 1$	$r'_{51} = 0$	$L'_5 = 0$
$P_6^w = (01001100)_2$	$d_{64} = 0$	$d_{63} = 1$	$d_{62} = 0$	$d_{61} = 0$	$r'_{63} = 1$	$r'_{62} = 1$	$r'_{61} = 0$	$L'_6 = 0$
$P_7^w = (01001100)_2$	$d_{74} = 0$	$d_{73} = 1$	$d_{72} = 0$	$d_{71} = 0$	$r'_{73} = 1$	$r'_{72} = 1$	$r'_{71} = 0$	$L'_7 = 0$
$P_8^w = (01001100)_2$	$d_{84} = 0$	$d_{83} = 1$	$d_{82} = 0$	$d_{81} = 0$	$r'_{83} = 1$	$r'_{82} = 1$	$r'_{81} = 0$	$L'_8 = 0$

Step 2

For $i = 1$ through 8, r_{i1} , r_{i2} , and r_{i3} are computed

$r_{i1} = \text{XOR}(r'_{i1}, d_{i1}, d_{i2}, d_{i4})$	$r_{11} = 0, r_{21} = 0, r_{31} = 0, r_{41} = 0, r_{51} = 0, r_{61} = 0, r_{71} = 0, r_{81} = 0$
$r_{i2} = \text{XOR}(r'_{i2}, d_{i1}, d_{i3}, d_{i4})$	$r_{12} = 0, r_{22} = 0, r_{32} = 0, r_{42} = 0, r_{52} = 0, r_{62} = 0, r_{72} = 0, r_{82} = 0$
$r_{i3} = \text{XOR}(r'_{i3}, d_{i2}, d_{i3}, d_{i4})$	$r_{13} = 0, r_{23} = 0, r_{33} = 0, r_{43} = 0, r_{53} = 0, r_{63} = 0, r_{73} = 0, r_{83} = 0$

Step 3

<p>Using equations 11 to 18</p> <p>$L_1 = \text{XOR}$ of bits d_{14}, for $i = 1$ to 8, $L_2 = \text{XOR}$ of bits d_{13}, for $i = 1$ to 8</p> <p>$L_3 = \text{XOR}$ of bits d_{12}, for $i = 1$ to 8, $L_4 = \text{XOR}$ of bits d_{11}, for $i = 1$ to 8</p> <p>$L_5 = \text{XOR}$ of bits r'_{i3}, for $i = 1$ to 8, $L_6 = \text{XOR}$ of bits r'_{i2}, for $i = 1$ to 8</p> <p>$L_7 = \text{XOR}$ of bits r'_{i1}, for $i = 1$ to 8, $L_8 = \text{XOR}$ of bits L'_i, for $i = 1$ to 7</p>	<p>$L_1 = 0, L_2 = 0$</p> <p>$L_3 = 0, L_4 = 0$</p> <p>$L_5 = 0, L_6 = 0$</p> <p>$L_7 = 0, L_8 = 0$</p>
--	---

Step 4

For $i = 1$ to 8, if $r_{i3} r_{i2} r_{i1} = 000$, it means that there is no error in the first 7 bits of all the pixels. For $i = 1$ through 8, furthermore, if $L_i = L'_i$, so there is no error in LSBs too.

Step 5

For $i = 1$ to 8, if $L_i = L'_i$, but for some value of i , $r_{i3} r_{i2} r_{i1} \neq 000$. This condition does not satisfy, so there does not exist any single bit error.

Step 6

For $i = 1$ to 8, if for any value of i , $L_i = L'_i$ and $r_{i3} r_{i2} r_{i1} \neq 000$. This condition does not satisfy, so there exists no 2-bit error. For $i = 1$ through 8, extract the watermarked bits $r_{i3} r_{i2} r_{i1}$ and L'_i . From all the eight pixels 32 bits are extracted. Thus, the extracted WBs are 1100 1100 1100 1100 1100 1100 1100 1100.

Figure 12. Example of watermark extraction

Example of watermark extraction with error correction

The watermarked pixel values are 60, 60, 60, 60, 76, 76, 76, and 76. The first pixel is P_1^w . Its actual value in binary is $(00111100)_2$. But during transit from sender to receiver, the first bit and third bit became erroneous. The first bit reached as 1, and the third bit has reached as 0. So, the pixel P_1^w has reached with 2-bit error as $(10011100)_2$.

Step 1

$P_1^w = (10011100)_2$	$d_{14} = 1$	$d_{13} = 0$	$d_{12} = 0$	$d_{11} = 1$	$r'_{13} = 1$	$r'_{12} = 1$	$r'_{11} = 0$	$L'_1 = 0$
$P_2^w = (00111100)_2$	$d_{24} = 0$	$d_{23} = 0$	$d_{22} = 1$	$d_{21} = 1$	$r'_{23} = 1$	$r'_{22} = 1$	$r'_{21} = 0$	$L'_2 = 0$
$P_3^w = (00111100)_2$	$d_{34} = 0$	$d_{33} = 0$	$d_{32} = 1$	$d_{31} = 1$	$r'_{33} = 1$	$r'_{32} = 1$	$r'_{31} = 0$	$L'_3 = 0$
$P_4^w = (00111100)_2$	$d_{44} = 0$	$d_{43} = 0$	$d_{42} = 1$	$d_{41} = 1$	$r'_{43} = 1$	$r'_{42} = 1$	$r'_{41} = 0$	$L'_4 = 0$
$P_5^w = (01001100)_2$	$d_{54} = 0$	$d_{53} = 1$	$d_{52} = 0$	$d_{51} = 0$	$r'_{53} = 1$	$r'_{52} = 1$	$r'_{51} = 0$	$L'_5 = 0$
$P_6^w = (01001100)_2$	$d_{64} = 0$	$d_{63} = 1$	$d_{62} = 0$	$d_{61} = 0$	$r'_{63} = 1$	$r'_{62} = 1$	$r'_{61} = 0$	$L'_6 = 0$
$P_7^w = (01001100)_2$	$d_{74} = 0$	$d_{73} = 1$	$d_{72} = 0$	$d_{71} = 0$	$r'_{73} = 1$	$r'_{72} = 1$	$r'_{71} = 0$	$L'_7 = 0$
$P_8^w = (01001100)_2$	$d_{84} = 0$	$d_{83} = 1$	$d_{82} = 0$	$d_{81} = 0$	$r'_{83} = 1$	$r'_{82} = 1$	$r'_{81} = 0$	$L'_8 = 0$

Step 2

For $i = 1$ through 8, r_{i1} , r_{i2} , and r_{i3} are computed

$r_{i1} = \text{XOR}(r'_{i1}, d_{i1}, d_{i2}, d_{i4})$	$r_{11} = 0, r_{21} = 0, r_{31} = 0, r_{41} = 0, r_{51} = 0, r_{61} = 0, r_{71} = 0, r_{81} = 0$
$r_{i2} = \text{XOR}(r'_{i2}, d_{i1}, d_{i3}, d_{i4})$	$r_{12} = 0, r_{22} = 0, r_{32} = 0, r_{42} = 0, r_{52} = 0, r_{62} = 0, r_{72} = 0, r_{82} = 0$
$r_{i3} = \text{XOR}(r'_{i3}, d_{i2}, d_{i3}, d_{i4})$	$r_{13} = 0, r_{23} = 0, r_{33} = 0, r_{43} = 0, r_{53} = 0, r_{63} = 0, r_{73} = 0, r_{83} = 0$

Step 3

<p>Using Equations 11 to 18</p> <p>$L_1 = \text{XOR}$ of bits d_{i4}, for $i = 1$ to 8, $L_2 = \text{XOR}$ of bits d_{i3}, for $i = 1$ to 8</p> <p>$L_3 = \text{XOR}$ of bits d_{i2}, for $i = 1$ to 8, $L_4 = \text{XOR}$ of bits d_{i1}, for $i = 1$ to 8</p> <p>$L_5 = \text{XOR}$ of bits r'_{i3}, for $i = 1$ to 8, $L_6 = \text{XOR}$ of bits r'_{i2}, for $i = 1$ to 8</p> <p>$L_7 = \text{XOR}$ of bits r'_{i1}, for $i = 1$ to 8, $L_8 = \text{XOR}$ of bits L'_i, for $i = 1$ to 7</p>	<p>$L_1 = 1, L_2 = 0$</p> <p>$L_3 = 1, L_4 = 0$</p> <p>$L_5 = 0, L_6 = 0$</p> <p>$L_7 = 0, L_8 = 0$</p>
--	---

Step 4

For $i = 1$ to 8, in all the eight cases if $r_{i3} r_{i2} r_{i1} = 000$, does not satisfy. That means there is some error.

Step 5

For $i = 1$ through 8, if $L_i = L'_i$ is not true for $i = 1$, and 3. Also but $r_{i3} r_{i2} r_{i1} = 000$, does not satisfy for $i = 1$. There is some error.

Step 6

For $i = 1$ to 8, inspect the conditions ($L_i \neq L'_i$ and $r_{i3} r_{i2} r_{i1} \neq 000$). For $i = 1$ and 3, the first condition, and for $i = 1$, the 2nd condition does not satisfy, so there exists 2-bit error. As per Table 2 correction logic if $r_{i3} r_{i2} r_{i1} = 010$ and $L_1 \neq L'_1$ and $L_3 \neq L'_3$, then there is error at d_{i4} and d_{i2} , so invert both the bits d_{i4} and d_{i2} . Thus, after correction, $P_1^w = (00111100)_2$. Now we can check that $L_i = L'_i$ satisfies when $i = 1$ and 3. For $i = 1$ to 8, extract the watermarked bits $r_{i3} r_{i2} r_{i1}$ and L'_i . From all the eight pixels 32 bits are extracted. Thus, the extracted WBs are 1100 1100 1100 1100 1100 1100 1100 1100.

Figure 13. Example of watermark extraction with error correction

5. Experimental results

This proposed MHCPB technique is coded in MATLAB. The used laptop computer comprises of 8 GB RAM and *i5* processor. Figure 14 lists 8 OIs taken from SIPI image repository [38] and Figure 15 depicts the corresponding WIs. The PSNR and SSIM values are mentioned under each WI. The merit of this technique is proved by measuring PSNR, HC, SSIM, and accuracy (ACC). Time for embedding (ET), and time for extraction (ExT) are also considered for evaluating the efficacy of this technique.

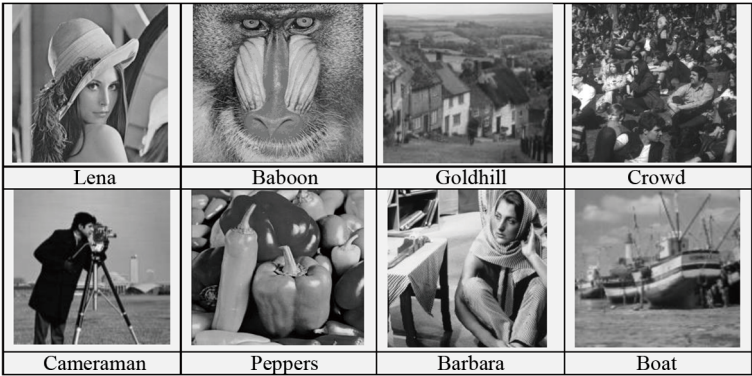


Figure 14. The original images

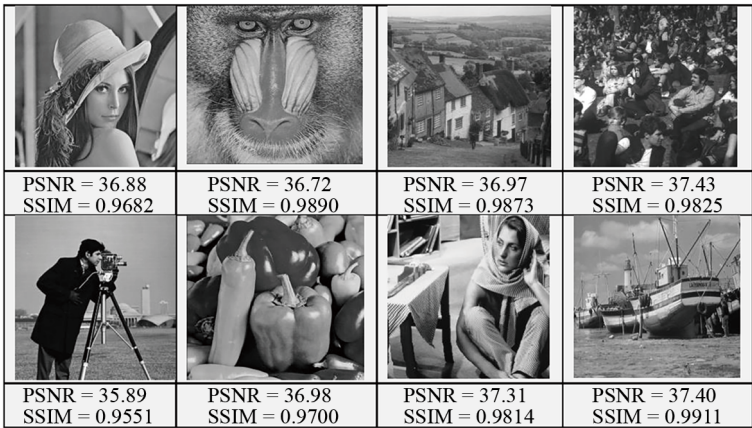


Figure 15. The watermarked images with PSNR and SSIM values given below each image

We have experimented with grey images, wherein each pixel is 1 byte or 8 bits. This technique can also be applied over color images. Each pixel of a color image is 3 bytes. Each component (Red, Green, Blue) is 8 bits or one byte. Each component of the color image shall be treated as a pixel equivalent in grey image, and the computation can be performed. Equation (19) measures the PSNR (distortion in WI) in decibels (dB). PSNR is computed to check the distortion after the WBs are embedded.

$$PSNR = 10 \times \log_{10} \frac{m \times n \times 255 \times 255}{\sum_{i=1}^m \sum_{j=1}^n (P_{ij} - Q_{ij})^2} \quad (19)$$

HC represents the count of message bits hidden in the whole image and bpp stands for the average HC per pixel. Equation (20) estimates the SSIM between OI and WI [26]. Here \bar{P} denotes pixel mean value of OI, \bar{Q} denotes pixel mean value of WI, σ_{pq} denotes variance between WI and OI. σ_p^2 and σ_q^2 represent variances over OI and WI accordingly. There are 2 constants c_1 and c_2 , plied to guarantee non-zero values for the denominator. The 2 constants $c_1 = (K_1 \times 255)^2$, and $c_2 = (K_2 \times 255)^2$, where $K_1 \ll 1$, and $K_2 \ll 1$. The SSIM value will be 1 when WI and OI are the same. The purpose of computing SSIM is to check the resemblance of WI with OI after WBs are hidden. The ACC estimates the extent to which we identify the tampered pixels. It is computed in Equation (21) using the count of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) [26].

$$SSIM = \frac{(2\bar{P}\bar{Q} + c_1)(2\sigma_{pq} + c_2)}{(\bar{P}^2 + \bar{Q}^2 + c_1)(\sigma_p^2 + \sigma_q^2 + c_2)} \quad (20)$$

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (21)$$

Table 3 records the efficacy metrics values of this MHCPB technique. From Table 3 we have noticed that the PSNR values do not fall below 36 dB, so it is acceptable. The HC is higher, and it is 4 bpp. The average SSIM value is 0.9781. It indicates that WI is 97.81% structurally similar to OI. The ACC = 0.9999 indicates that the number of tampered pixels is identified with 99.99% accuracy. The watermark embedding time is 16.42 s, and extraction time is 17.45 s.

Table 3. Efficacy measurement of MHCPB technique

Images	SSIM	ACC	PSNR (dB)	HC (bpp)	ET (secs)	ExT (secs)
Lena	0.9682	0.9999	36.88	4.0	17.34	18.07
Baboon	0.9890	0.9999	36.72	4.0	19.61	17.76
Goldhill	0.9873	0.9999	36.97	4.0	17.29	20.41
Crowd	0.9825	0.9999	37.43	4.0	18.03	18.29
Cameraman	0.9551	0.9999	35.89	4.0	14.45	14.70
Pepper	0.9700	0.9999	36.98	4.0	14.31	16.18
Barbara	0.9814	0.9999	37.31	4.0	16.01	16.57
Boat	0.9911	0.9999	37.40	4.0	14.36	17.62
Average	0.9781	0.9999	36.94	4.0	16.42	17.45

Figure 16 represents the tampered Baboon images with tampering % from 5% to 45%. A pixel is tampered if its value is changed in any of the bits. The tampered pixels are localized and made white color for easy reference by the readers. It is shown for only one image. In fact, it is performed for all the test images.

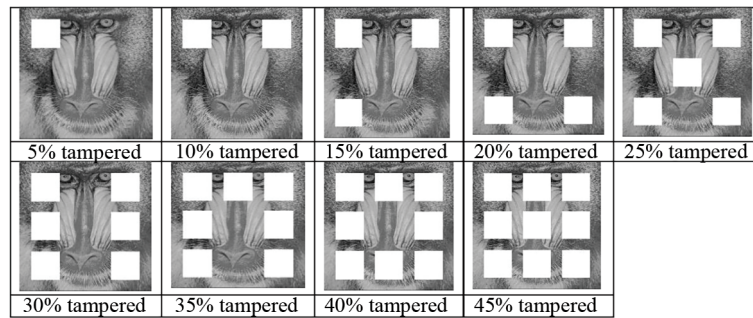


Figure 16. Tampered Baboon images (the tampered location pixels are made white color for reference)

Table 4 depicts the mean value of bpp, PSNR, SSIM, and ACC over 8 images with various tampering percentages starting from 0% to 45%. Figure 17a depicts the tampering percentage versus PSNR, Figure 17b depicts the tampering percentage versus SSIM, and Figure 17c depicts the tampering percentage versus ACC. These plots indicate that if the tampering is raised beyond 45%, still the PSNR can be at least 36 dB. the SSIM will be maintained more than 0.97, and the ACC will be maintained more than 0.99.

Table 4. Efficacy at various tampering rates

Efficacy parameter	Tampering rate									
	0%	5%	10%	15%	20%	25%	30%	35%	40%	45%
bpp	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
PSNR	36.94	36.82	36.73	36.63	36.53	36.44	36.35	36.27	36.19	36.11
SSIM	0.9781	0.9761	0.9740	0.9731	0.9722	0.9718	0.9714	0.9711	0.9708	0.9705
ACC	0.9999	0.9991	0.9982	0.9975	0.9966	0.9957	0.9947	0.9938	0.9929	0.9921

Table 5. Comparison with existing techniques

Technique	SSIM	ACC	PSNR	bpp
Nazari et al. [18]	0.9928	0.9845	36.50	1.66
Chang et al. [22]	0.9844	0.9969	37.88	3.0
Prasad and Pal [26]	0.9994	0.9995	42.09	1.5
Prasad and Pal [27]	0.9861	0.9990	37.94	3.0
Proposed MHCPB	0.9781	0.9999	36.94	4.0

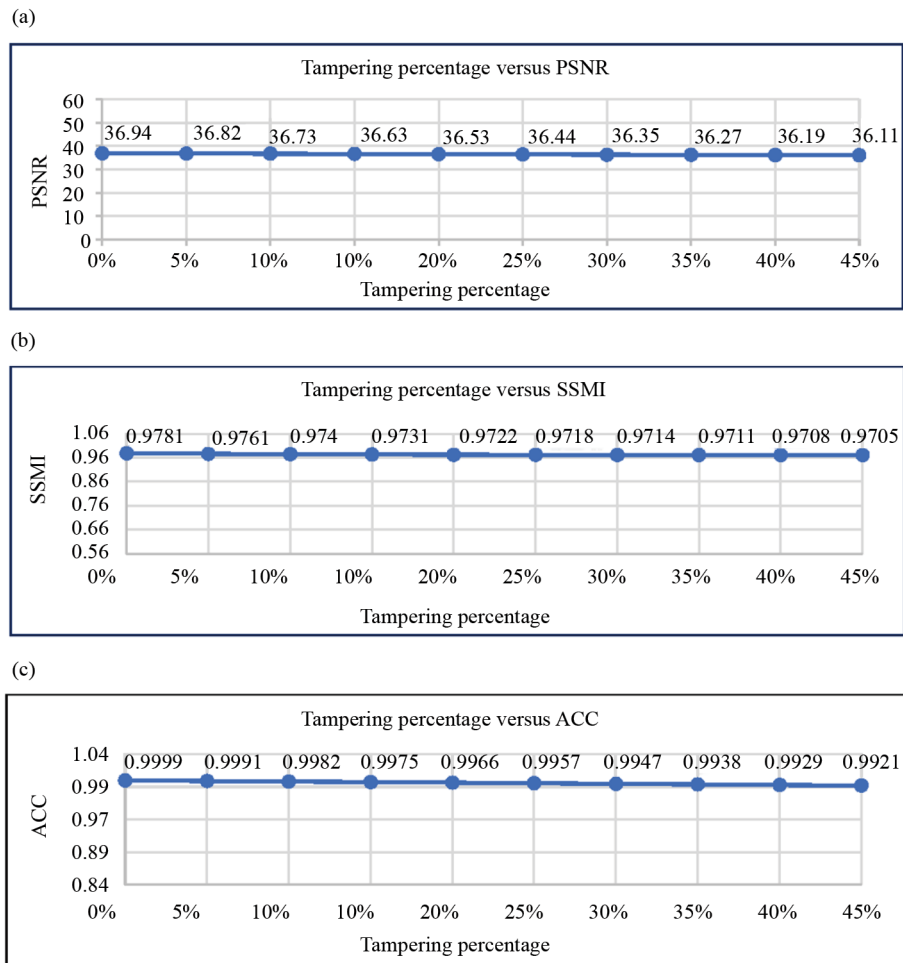


Figure 17. (a) tampering % versus PSNR, (b) Tampering % versus SSIM, (c) Tampering % versus ACC

Table 5 represents the average estimates of different parameter values of our MHCPB technique with that of Nazari et al. [18], Chang et al. [22], Prasad et al. [26] and Prasad et al. [27]. These three existing techniques are based on Hamming code. Furthermore, Figure 18 shows a bar graph comparing the bpp and PSNR values of existing techniques with MHCPB technique. Figure 19 shows a bar graph comparing the SSIM and ACC values of existing techniques and MHCPB technique. From Figure 18 it has been noted that the bpp of the MHCPB technique is surely over the existing techniques, but PSNR is moderate. Although the PSNR of our MHCPB scheme is moderate it is unacceptable range of 30-40 dB. From Figure 19, it can be noticed that the SSIM value of the MHCPB technique is lesser than existing schemes, but it is acceptable. But the ACC of MHCPB scheme is more than the existing techniques, it is 0.9999.

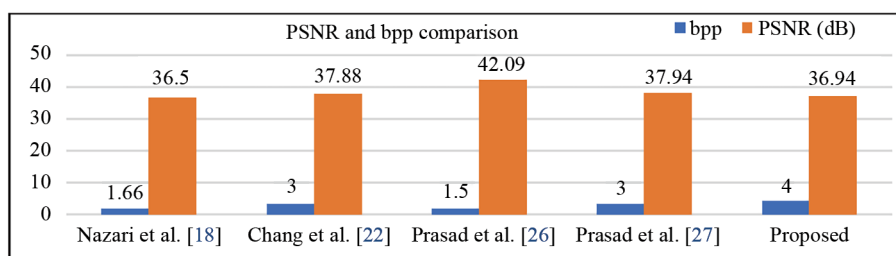


Figure 18. bpp and PSNR comparison with related techniques

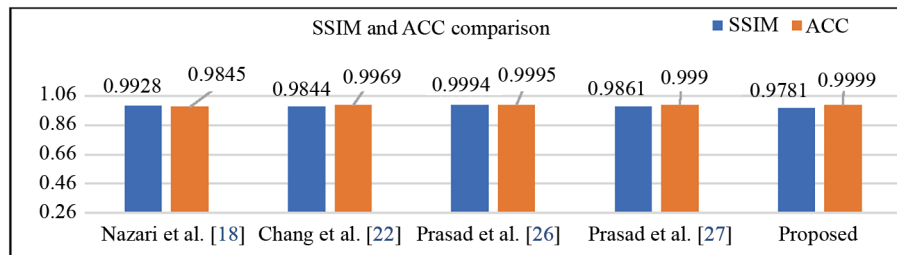


Figure 19. SSIM and ACC comparison with related techniques

6. Conclusions

The MHCPB tamper detection and correction scheme has been proposed for images. The OI is logically fragmented into 8-pixel blocks. Here the 64 bits of these 8 pixels could be arranged into an 8×8 matrix of bits. A modified (7,4) Hamming code is proposed. This Hamming code is applied on first 7 bits of each row of matrix. The first 4 MSBs are assumed as data and next 3 bits are treated as redundant bits to store the WBs. The WBs are calculated using 4 MSBs in a non-traditional way and stored in 3 redundant bit positions. Furthermore, the column parity for the first 7 columns of the 8×8 matrix is computed and stored in the first 7-bit locations of the 8th column. Thereafter the column parity of the first 7 bits of 8th column is stored in 8th bit location of 8th column. This technique can efficiently detect 1-bit error in every pixel. Furthermore, it can also detect 2-bit error if it occurs in only one of the 8 pixels of the block. Experimental results reveal that this proposed technique possesses higher payload i.e., 4.0 bpp with an acceptable PSNR value 36.94 dB. The estimated SSIM is 0.9781. It indicates that WI is almost 97.81% similar to the OI. We got ACC value 0.9921 (at 45% tampering percentage). It implies that tampered pixels are recognized with 99.21% accuracy. This technique cannot handle error in more than 2 bits. To do this the error correction mechanism is to be changed appropriately. This is a future research direction.

Acknowledgments

The authors are thankful to the KLEF management for providing the necessary facilities to complete this research work.

Conflict of interest

The authors declare that there is no conflict of interest.

References

- [1] Abd Warif NB, Wahab AWA, Idris MYI, Ramli R, Salleh R, Shamshirband S, et al. Copy-move forgery detection: Survey, challenges, and future directions. *Journal of Network and Computer Applications*. 2016; 75: 259-278.
- [2] Agarwal N, Singh AK, Singh PK. Survey of robust and imperceptible watermarking. *Multimedia Tools and Applications*. 2019; 78: 8603-8633.
- [3] Singh L, Singh AK, Singh PK. Secure data hiding techniques: A survey. *Multimedia Tools and Applications*. 2020; 79: 15901-15921.
- [4] Singh D, Singh SK. Effective self-embedding watermarking scheme for image tampered detection and localization with recovery capability. *Journal of Visual Communication and Image Representation*. 2016; 38: 775-789.

- [5] Cao F, An B, Wang J, Ye D, Wang H. Hierarchical recovery for tampered images based on watermarking self-embedding. *Displays*. 2017; 46: 52-60.
- [6] Gull S, Loan NA, Parah SA, Sheikh JA, Bhat GM. An efficient watermarking technique for tamper detection and localization of medical images. *Journal of Ambient Intelligence and Humanized Computing*. 2020; 11: 1799-1808.
- [7] Feng B, Li X, Jie Y, Guo C, Fu H. A novel semi-fragile digital watermarking scheme for scrambled image authentication and restoration. *Mobile Networks and Applications*. 2020; 25: 82-94.
- [8] Bhalerao S, Ansari IA, Kumar A. A secure image watermarking for tamper detection and localization. *Journal of Ambient Intelligence and Humanized Computing*. 2021; 12: 1057-1068.
- [9] Gul E, Ozturk S. A novel triple recovery information embedding approach for self-embedded digital image watermarking. *Multimedia Tools and Applications*. 2020; 79: 31239-31264.
- [10] Sinhal R, Ansari IA, Ahn CW. Blind image watermarking for localization and restoration of color images. *IEEE Access*. 2020; 8: 200157-200169.
- [11] Qin C, Ji P, Zhang X, Dong J, Wang J. Fragile image watermarking with pixel-wise recovery based on overlapping embedding strategy. *Signal Processing*. 2017; 138: 280-293.
- [12] Kosuru SNVJD, Swain G, Kumar N, Pradhan A. Image tamper detection and correction using Merkle tree and remainder value differencing. *Optik*. 2022; 261: 169212.
- [13] Rawat S, Raman B. A chaotic system based fragile watermarking scheme for image tamper detection. *International Journal of Electronics and Communications (AEU)*. 2011; 65: 840-847.
- [14] Botta M, Cavagnino D, Pomponiu V. A successful attack and revision of chaotic system based fragile watermarking scheme for image tamper detection. *International Journal of Electronics and Communications (AEU)*. 2015; 69(1): 242-245.
- [15] Prasad S, Pal AK. A secure fragile watermarking scheme for protecting integrity of digital images. *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*. 2020; 44: 703-727.
- [16] Sahu AK. A logistic map based blind and fragile watermarking for tamper detection and localization in images. *Journal of Ambient Intelligence and Humanized Computing*. 2022; 13: 3869-3881.
- [17] Tong X, Liu Y, Zhang M, Chen Y. A novel chaos-based fragile watermarking for image tampering detection and self-recovery. *Signal Processing: Image Communication*. 2013; 28: 301-308.
- [18] Nazari M, Sharif A, Mollaeefar M. An improved method for digital image fragile watermarking based on chaotic maps. *Multimedia Tools and Applications*. 2017; 76: 16107-16123.
- [19] Sreenivas K, Kamakshiprasad V. Improved image tamper localization using chaotic maps and self-recovery. *Journal of Visual Communication and Image Representation*. 2017; 49: 164-176.
- [20] Hamming RW. Error detecting and error correcting codes. *Bell System Technical Journal*. 1950; 29(2): 147-160.
- [21] Chan CS, Chang CC. An efficient image authentication method based on Hamming code. *Pattern Recognition*. 2007; 40: 681-690.
- [22] Chang CC, Chen KN, Lee CF, Liu LJ. A secure fragile watermarking scheme based on chaos-and-Hamming code. *Journal of Systems and Software*. 2011; 84(9): 1462-1470.
- [23] Wang JT, Chang YC, Yu CY, Yu SS. Hamming code based watermarking scheme for 3D model verification. *Mathematical Problems in Engineering*. 2014; 2014: 241093.
- [24] Islam MS, Kim CH, Kim JM. A GPU-based (8, 4) Hamming decoder for secure transmission of watermarked medical images. *Cluster Computing*. 2015; 18: 333-341.
- [25] Trivedy S, Pal AK. A logistic map-based fragile watermarking scheme of digital images with tamper detection. *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*. 2017; 41: 103-113.
- [26] Prasad S, Pal AK. A tamper detection suitable fragile watermarking scheme based on novel payload embedding strategy. *Multimedia Tools and Applications*. 2020; 79: 1673-1705.
- [27] Prasad S, Pal AK. Hamming code and logistic-map based pixel-level active forgery detection scheme using fragile watermarking. *Multimedia Tools and Applications*. 2020; 79: 20897-20928.
- [28] Pal P, Jana B, Bhaumik J. An image authentication and tampered detection scheme exploiting local binary pattern along with Hamming error correcting code. *Wireless Personal Communications*. 2021; 121: 939-961.
- [29] Jana B, Giri D, Mondal SK. Dual image based reversible data hiding scheme using (7,4) Hamming code. *Multimedia Tools and Applications*. 2018; 77(1): 763-785.
- [30] Nguyen D, Le HD. A reversible data hiding scheme based on (5, 3) Hamming code using extra information on overlapped pixel blocks of grayscale images. *Multimedia Tools and Applications*. 2021; 80: 13099-13120.

- [31] Chen CC, Chang CC, Chen K. High capacity reversible data hiding in encrypted image based on Huffman coding and differences of high nibbles of pixels. *Journal of Visual Communication and Image Representation*. 2021; 76: 1-10.
- [32] Ramos AM, Artiles JAP, Chaves DPB, Pimentel C. A fragile image watermarking scheme in DWT domain using chaotic sequences and error-correction codes. *Entropy*. 2023; 25(508): 1-23.
- [33] Chen K, Chang CC. Real-time error-free reversible data hiding in encrypted images using (7,4) Hamming code and most significant bit prediction. *Symmetry*. 2019; 11(51): 1-17.
- [34] Wang Y, Tang M, Wang Z. High-capacity adaptive steganography based on LSB Hamming code. *Optik*. 2020; 213(164685): 1-9.
- [35] Wu X, Yang CN, Liu YW. High capacity partial reversible data hiding by Hamming code. *Multimedia Tools and Applications*. 2020; 79: 23425-23444.
- [36] Khadse DB, Swain G. Data hiding and integrity verification based on quotient value differencing and Merkle tree. *Arabian Journal for Science and Engineering*. 2023; 48: 1793-1805.
- [37] Kosuru SNVJD, Pradhan A, Basith KA, Sonar R, Swain G. Digital image steganography with error correction on extracted data. *IEEE Access*. 2023; 11: 80945-80957.
- [38] University of Southern California (USC) Signal and Image Processing Institute. USC-SIPI image database. Available from: <http://sipi.usc.edu/database/database.php?volume=misc> [Accessed 15th December 2022].