electronics

MDPI

Article

A Simple and Efficient Data Hiding Method with Error Detection and Correction

Hengxiao Chi [1], Jui-Chuan Liu [1,*], Chin-Chen Chang [1,*] and Ji-Hwei Horng [2]

[1] Department of Information Engineering and Computer Science, Feng Chia University, Taichung 40724, Taiwan; hx9704@gmail.com
[2] Department of Electrical Engineering, National Quemoy University, Kinmen 89250, Taiwan; horng@email.nqu.edu.tw
* Correspondence: p1200318@o365.fcu.edu.tw (J.-C.L.); alan3c@gmail.com (C.-C.C.)

Abstract: In recent years, data hiding techniques have emerged as a prominent area of research due to the increasing demand for information security. Existing data hiding techniques typically do not prioritize the verification of the correctness of the extracted data, nor do they attempt to rectify any erroneous information extracted. This paper presents a simple yet effective method by which a recipient can perform error detection and correction on extracted data. Prior to data embedding, a preprocess is applied to secret data. After converting the secret data to a data stream, a data hider categorizes adjacent and identical bits within the data stream into groups. The number of these bits is then recorded as an indicator to extract information and perform error detection and correction. The number of bits in each group is adjusted to ensure that it is an odd number. Moreover, the collected indicator data stream is encoded using (7,4) Hamming code to compact the recorded information. Finally, the concatenation of the encoded indicator and the encoded secret is embedded into the multiple least significant bits (LSBs) of the original image pixels. As for the receiver, error correction can be applied based on the indicator after the extraction of the encoded data. Experimental results demonstrate that this method allows the data receiver to detect and correct errors while maintaining a certain level of embedding capacity and image quality. In terms of image quality, this method exhibits superior performance compared to existing research when the embedded data volume is relatively small.

Keywords: data hiding; error detection; error correction

check for updates

Citation: Chi, H.; Liu, J.-C.; Chang, C.-C.; Horng, J.-H. A Simple and Efficient Data Hiding Method with Error Detection and Correction. *Electronics* **2024**, *13*, 2018. https://doi.org/10.3390/electronics13112018

Academic Editor: Andreas Mauthe

Received: 22 April 2024
Revised: 15 May 2024
Accepted: 16 May 2024
Published: 22 May 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the advancement of technologies, such as Virtual Reality (VR), Augmented Reality (AR), and cloud computing, there is a growing need for faster data transfer speeds and lower latency, so researchers have turned their attention to the development of Fifth Generation (5G) communication technology. The research and development of 5G technology have significantly enhanced the speed and efficiency of information transmission. As 5G technology continues to evolve and becomes more widely spread, the demand for information security has also steadily increased. Data hiding (DH) technology consistently remains a popular and enduring topic in the field of information security.

Data hiding technology can be broadly categorized into two major classes based on the extent of recoverability of cover images: reversible data hiding (RDH) [1–7] and irreversible data hiding (IRDH) [8–17]. As the names suggest, RDH techniques allow for the lossless restoration of the cover image after extracting the hidden data, whereas IRDH techniques introduce irreversible distortions to the cover image during the embedding of secret data. RDH techniques, in their pursuit of lossless cover image recovery, allocate extra space during the data embedding process to store auxiliary information required for image restoration. IRDH is a better choice when there is no special requirement for the quality

of cover images. IRDH techniques compromise image quality in exchange for increased capacity to embed data.

However, whether it is reversible or irreversible data hiding technology, most existing DH methods have primarily been designed under the ideal assumption that the stego image remains unaltered during transmission. There are relatively few methods that enable the recipient to perform correctness checks and error correction on the extracted data. In digital communications, the reliability of data transmission is critical, and it is worth sacrificing some embedding capabilities to introduce redundant information in order to detect and correct errors.

In the field of communication, various techniques have been developed to detect and correct errors in data transmission. These techniques include parity code, Hamming code [18–20], cyclic redundancy check (CRC) [21–23], and many more. Parity code is one of the earliest error-checking codes, originally used in telegraph communication during the early 20th century. It ensures that the number of ones in the data is either odd or even by adding a parity bit (odd or even parity) to the data. This way, if a bit error occurs during transmission or storage, parity codes can detect it. In the mid-20th century, Hamming code was developed and has become widely used as an error-correcting technique. Hamming code arranges the original data bits by following specific rules to generate and append redundant check bits. At the receiving end, these check bits are used to detect errors. If only one error occurs in the data, the receiving end can analyze the check bits to determine the location of the error and enable the ability to correct it. CRC code employs a polynomial encoding technique by attaching a polynomial checksum to the data for error detection.

This paper introduces a straightforward yet effective IRDH technique that allows the receiver to identify errors in the extracted data and correct erroneous bits. The data hider begins by grouping the secret data and recording the number of bits in each group as an indicator. Then, it encodes the secret data, ensures that each group has an odd number of bits, and that there is more than a bit. This is performed to ensure that the principle of majority votes can be applied to validate data when extracting secret data. In case a group contains only one bit, the single bit is duplicated, resulting in having three bits of data instead. To facilitate information extraction, verification, and error correction, the indicators are further encoded using a (7,4) Hamming code to guard against indicator errors. Subsequently, the receiver first extracts the indicators and verifies their accuracy. The extracted data streams are then grouped according to the indicators. Since all the elements within a group should be the same, errors can be easily detected and the incorrect data can be restored using the voting principle. The redundant elements can then be eliminated according to the indicators to obtain the final secret data.

The main contributions of this paper are as follows:

1. The copy encoding method is simpler to use than other methods and can detect errors more easily;
2. The use of voting strategy can effectively correct errors;
3. When the secret data encoded is not greater than 10×10^5 bits, the peak signal-to-noise ratio (PSNR) of the stego image generated by the proposed method is higher than other schemes;
4. It takes little time to perform encoding and decoding.

Overall, experimental results demonstrate the feasibility of the proposed algorithm, exhibiting high error-detection rate, high error-correction rate, high image visual quality and high total embedding capacity.

The remaining sections of this paper are organized as follows: Section 2 briefs the related works. Section 3 presents a detailed description of the proposed algorithm. Section 4 describes the experimental results and performance evaluation of the proposed approach. Section 5 presents the conclusions and outlines future research directions.

2. Related Works

In this section, we will provide a brief overview of the (7,4) Hamming code [18–20], and the data hiding method that combines quotient value differencing (QVD) and Hamming code capable of error detection and correction proposed by Kosuru et al. [16].

2.1. The (7,4) Hamming Code

Hamming Code is a coding technique used to detect and correct errors. Although there are other, more complex error correction codes available in modern coding technology, Hamming codes remain an important component of many systems that require error detection and correction because of their simplicity, effectiveness, and reliability. At the sender's end, the (7,4) Hamming code generates three bits of parity information for every four bits of data. The parity bits are used to check whether the four bits of data have changed during the transmission process. If a change has occurred, these three bits of parity information indicate the location of the data changed. When a receiver receives the data, the data are divided into groups of seven bits, allowing for the analysis of the three-bit parity codes in each group to verify the integrity of the data. If there is only one incorrect bit in a group, the receiver can utilize the three-bit parity code associated with the group to identify the location of the error and correct it. The specific operations of parity coding and error location identification are depicted in Figure 1.

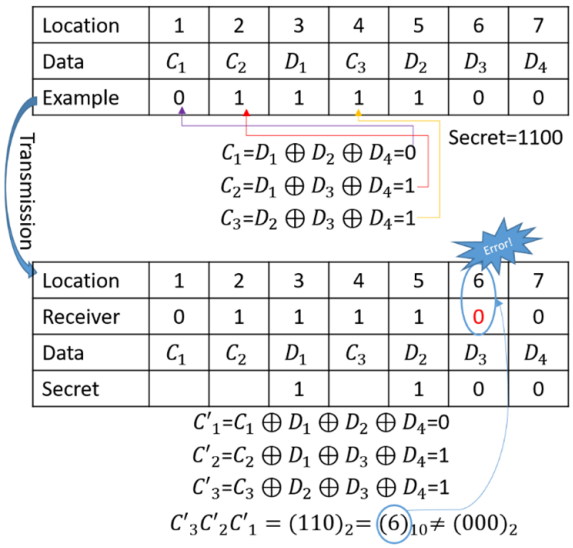


Figure 1. Example of coding and error correction process on (7,4) Hamming code.

According to the encoding rules of the (7,4) Hamming code, the first, second, and fourth positions are designated for parity bits in a group of seven bits. The four bits of data to be encoded are placed sequentially in the remaining positions. Parity bit C_1 at the first position is responsible for checking data bits in positions 1, 3, 5, and 7. Parity bit C_2 at the second position checks data bits in positions 2, 3, 6, and 7. Parity bit C_3 at the fourth position checks data bits in positions 4, 5, 6, and 7. Parity bits are generated based on the count of '1's in the data bits they oversee. For a parity bit, if there is an odd number of '1's in the data being checked, the parity bit is set to 1; otherwise, it is set to 0. The specific formulas for each parity bit are shown in Equations (1)–(3). Where D_i denotes the i-th bit of secret data in the group.

$$C_1 = D_1 \oplus D_2 \oplus D_4, \tag{1}$$

$$C_2 = D_1 \oplus D_3 \oplus D_4, \tag{2}$$

$$C_3 = D_2 \oplus D_3 \oplus D_4. \tag{3}$$

When the receiver receives a group of seven bits of data, it recalculates the parity bits based on the '1' counts in the data positions they oversee. The new parity bits are then used to determine whether the data changed during transmission. If the newly generated parity bits are not all zeroes, it indicates there is an error in the data at least. If there is only one error, the value of $(C'_3C'_2C'_1)$ is converted from the binary format to decimal as the error location. Once the exact location is figured out, the correction can be performed at the location. If there are multiple errors occurring in various positions, the presence of the errors can be detected from the parity bits but not their exact locations. Equations (4)–(6) show how the receiver's parity bits are calculated:

$$C'_1 = C_1 \oplus D_1 \oplus D_2 \oplus D_4, \tag{4}$$

$$C'_2 = C_2 \oplus D_1 \oplus D_3 \oplus D_4, \tag{5}$$

$$C'_3 = C_3 \oplus D_2 \oplus D_3 \oplus D_4. \tag{6}$$

2.2. Kosuru et al.'s Method [16]

In order to make the data hiding technique capable of error detection and correction while extracting information, Kosuru et al. combined the Hamming code technique with the QVD data embedding method. Firstly, Kosuru et al. divided the cover image into non-overlapping 2×2 pixel blocks, and then divided the pixels in each block into two parts: one part was the LSB values L and the other part was the quotient values Q. As shown in Figure 2, each pixel value was divided by two to obtain one part with seven MSBs and one part with one LSB. Then the eight-bit data, $D_8D_7D_6D_5D_4D_3D_2D_1$, was read from the secret data stream, and the coding rule of (12,8) Hamming code was used to calculate the four-bit check digit $c_4c_3c_2c_1$. Similar to the encoding rules of (7,4) Hamming code, the specific calculation is as follows:

$$\begin{cases} c_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7 \\ c_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7 \\ c_3 = D_2 \oplus D_3 \oplus D_4 \oplus D_8 \\ c_4 = D_5 \oplus D_6 \oplus D_7 \oplus D_8 \end{cases}. \tag{7}$$

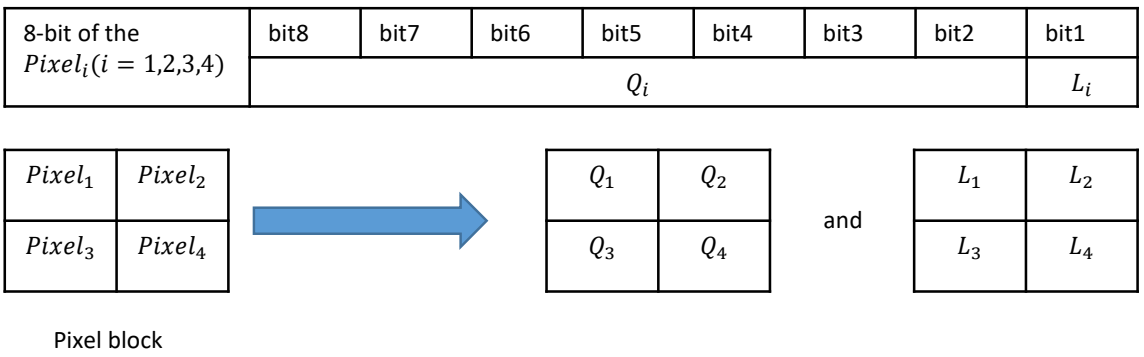


Figure 2. The way to divide the pixels of a pixel block into two parts.

The secret data and the checksum are concatenated together to obtain $D_8D_7D_6D_5D_4D_3D_2D_1c_4c_3c_2c_1$. Then, $D_8D_7D_6$ is transformed into a decimal number b_1, $D_5D_4D_3$ is transformed into a decimal number b_2, and $D_2D_1c_4$ is transformed into a decimal number b_3. Then, $Q_1 - Q_2 = d_1$, $Q_1 - Q_3 = d_2$ and $Q_1 - Q_4 = d_3$ are calculated, respectively. Then, each of the 16 quantization ranges (QR) of Table 1 are observed to see which of them the absolute values of d_1, d_2 and d_3 belong to. Each QR has its upper bound (UB) and lower

bound (LB). Using the LB differences, b_1, b_2 and b_3 are embedded into the interpolated d_1, d_2 and d_3. The specific calculation is as follows:

$$d'_j = \begin{cases} LB_j + b_j, & if\ d_j \geq 0 \\ -LB_j - b_j, & if\ d_j < 0 \end{cases}, (\mathrm{j} = 1, 2, 3) \tag{8}$$

where LB_j is the LB of the QR which d_j belongs. The values $|d'_1 - d_1| = m_1$, $|d'_2 - d_2| = m_2$ and $|d'_3 - d_3| = m_3$ are further computed. The change in the difference after embedding the data is mapped to the corresponding quotient pairs through Equations (9)–(11).

$$\left(Q_1{}^1, Q'_2\right) = \begin{cases} \left(Q_1 - \left\lfloor \frac{m_1}{2} \right\rfloor, Q_2 + \left\lceil \frac{m_1}{2} \right\rceil\right), & if\ mod(d_1, 2) = 0 \\ \left(Q_1 - \left\lceil \frac{m_1}{2} \right\rceil, Q_2 + \left\lfloor \frac{m_1}{2} \right\rfloor\right), & if\ mod(d_1, 2) = 1 \end{cases}, \tag{9}$$

$$\left(Q_1{}^2, Q'_3\right) = \begin{cases} \left(Q_1 - \left\lfloor \frac{m_2}{2} \right\rfloor, Q_3 + \left\lceil \frac{m_2}{2} \right\rceil\right), & if\ mod(d_2, 2) = 0 \\ \left(Q_1 - \left\lceil \frac{m_2}{2} \right\rceil, Q_3 + \left\lfloor \frac{m_2}{2} \right\rfloor\right), & if\ mod(d_2, 2) = 1 \end{cases}, \tag{10}$$

$$\left(Q_1{}^3, Q'_4\right) = \begin{cases} \left(Q_1 - \left\lfloor \frac{m_3}{2} \right\rfloor, Q_4 + \left\lceil \frac{m_3}{2} \right\rceil\right), & if\ mod(d_3, 2) = 0 \\ \left(Q_1 - \left\lceil \frac{m_3}{2} \right\rceil, Q_4 + \left\lfloor \frac{m_3}{2} \right\rfloor\right), & if\ mod(d_3, 2) = 1 \end{cases}, \tag{11}$$

where $\lfloor * \rfloor$ implies rounding down and $\lceil * \rceil$ implies rounding up. Then $Q_1{}^4 = \left\lceil \frac{Q_1{}^1 + Q_1{}^2 + Q_1{}^3}{3} \right\rceil$ is calculated. This gives us four embedded Q_1 values $Q_1{}^1$, $Q_1{}^2$, $Q_1{}^3$ and $Q_1{}^4$. By using the differences d'_1, d'_2 and d'_3, we can easily derive four sets of Q'_2, Q'_3 and Q'_4 corresponding to the four Q_1 values. From these four cases, we pick the one with the smallest mean square error from the original quotient value as the final stego quotient value Q'_1, Q'_2, Q'_3 and Q'_4 to be used.

Table 1. The quantization range table of the quotient difference.

QR	QR1	QR2	QR3	QR4	QR5	QR6	QR7	QR8	QR9	QR10	QR11	QR12	QR13	QR14	QR15	QR16
LB	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120
UB	7	15	23	31	39	47	55	63	71	79	87	95	103	111	119	127

The stego quotient values Q'_1, Q'_2, Q'_3 and Q'_4 are then checked for any overflow. If there is no overflow, then L_1 = flag = 1 is recorded, and L_2, L_3 and L_4 are replaced with the remaining c_1, c_2 and c_3. If there is overflow, we need to abandon the above operation and directly embed D_8D_7 into the two-bit LSBs of the original Q_1, embed D_6D_5 into the two-bit LSBs of the original Q_2, embed D_4D_3 into the two-bit LSBs of the original Q_3, and embed $D_2D_1c_4$ into the three-bit LSBs of the original Q_4. Then, L_1 = flag = 0 is marked and L_2, L_3 and L_4 are replaced with the remaining c_1, c_2 and c_3. The final stego pixel is $Pixel'_i = Q'_i \times 2 + L_i$ $(i = 1, 2, 3, 4)$.

The operation of extracting the secret data is the inverse operation of embedding. In this way, Kosuru et al. can extract 12 bits of information from a pixel block and compute the check digit again for error detection and correction by the rule of (12,8) Hamming code. The method is similar to the (7,4) Hamming code introduced in Section 2.1.

3. Proposed Scheme

The (7,4) Hamming coding, although more complex than parity checks, remains relatively simple and finds applicability in numerous scenarios. However, it can only correct a single bit error for every seven bits of data, making its error tolerance limited in cases of multiple or complex errors. In this section, we will introduce a straightforward yet efficient data hiding method capable of performing data verification and correction. Section 3.1 will cover the encoding of secret data, Section 3.2 will explain the encoding of

indicator data, Section 3.3 will describe the data embedding process, and Section 3.4 will detail the data extraction along with the verification and correction processes.

3.1. The Copy Encoding

We use an efficient yet simple coding method, which we call copy encoding, to encode the secret data stream. Initially, we group successive identical bits in the secret data stream into groups. The bit values in a group can only be all 0s or all 1s. The number of bits in a group is recorded as the indicator of the group. Thus, IN_j is the indicator for the j-th group and j is the index of the group. If there are more than seven successive bits with the same value in the secret stream, we need to group them into multiple groups to guarantee that the value of IN_j is in the range of [1, 7]. The indicators recorded are used to guide the extraction of secret data. Then we encode each group of secret data stream according to the parity of IN. The principle of secret data encoding at this stage aims to ensure that the number of encoded bit counts in a group is an odd number and is greater than one, so that the principle of 'majority voting rule' can be applied to verify and correct the data in the group during the data extraction process. In case IN_j is an even number, we copy one bit of data in the group and append it to the end of the bits in the group to ensure that the count of the encoded data in the group is odd. If IN_j is an odd number and is greater than one, the encoded secret bit stream of the group is the same as the original. When a group contains only one bit, i.e., the group's indicator $IN_j = 1$, we duplicate the data twice to make the encoded group comprise three bits.

Figure 3 provides an example of encoding secret data using copy encoding. When there is a secret data stream 000000000001000 as shown in the figure, we want to group the successive identical bits into groups first. There are 11 consecutive bits with the same bit value of 0; since we want to keep the value of an indicator in the range of [1, 7], these 11 '0' bits will be split into 2 groups: the first 7 bits in 1 group and $IN_1 = 7$; the remaining four bits into 1 group and $IN_2 = 4$. After the serial '0's, there is a single bit one which forms a group on its own and $IN_3 = 1$. Finally, the last three successive bits of zero can be gathered into the fourth group and $IN_4 = 3$. With this process, we finish grouping the secret data stream and IN denotes the indicators of the groups created. Following the grouping, we encode each group. Since IN_1 and IN_4 are odd numbers and not equal to one, bits in the first group and the fourth group remain unchanged. IN_2 is an even number; therefore, we need to copy one bit and append it to the end of the group to gain five bits of zero. When $IN_3 = 1$, we need to copy the single bit twice and append them to make it a group of three bits of one. Lastly, concatenate the encoded bits in groups to form the encoded secret data stream 000000000000111000. The algorithm for copy encoding is shown in Algorithm 1. The time complexity of the encoding process of secret data is O(N).

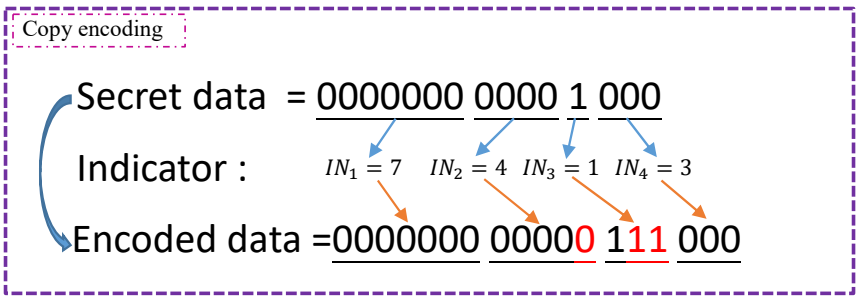


Figure 3. Examples of generating indicators and using copy encoding.

Algorithm 1. Copy encoding algorithm.

Input: Secret data $\mathcal{S} = s_1 s_2 s_3 \cdots s_n$
Output: Indicator IN, encoded data $\mathcal{S}'$

add s_1 to $group_j$ /* $j = 1$ */
$IN_j = 1$ /* $j = 1$ */
for $i = 2$: n **do**
  if $IN_j \neq 7$
    if $s_i == s_{i-1}$
      add s_i to $group_j$
      $IN_j = IN_j + 1$
    else
      $j = j + 1$
      add s_i to $group_j$
      $IN_j = 1$
    end if
  else if $IN_j == 7$
    $j = j + 1$
    add s_i to $group_j$
    $IN_j = 1$
  end if
end for
if $\mathrm{mod}(IN_j, 2) == 0$
  copy one secret bit of $group_j$, appending it to the end of $group_j$
else if $\mathrm{mod}(IN_j, 2) == 1$ && $IN_j == 1$
  copy the secret bit in the $group_j$ twice
else if $\mathrm{mod}(IN_j, 2) == 1$ && $IN_j \neq 1$
  the secret bits in $group_j$ are unchanged
end if
Concatenate the data in all $group_j$ to obtain the encoded data $\mathcal{S}'$.
return IN, and $\mathcal{S}'$

3.2. Coding of Indicators IN

To prevent indicators from being tampered with during transmission, it is necessary to encode the indicators IN as well. Encoding the indicators enables us to extract the correct indicators to use in guiding data extraction with data validation and correction. The copy encoding method will inevitably produce extra indicators, so it is not suitable to encode indicators IN. Therefore, we chose the traditional (7,4) Hamming coding method to encode indicators instead. When encoding the secret data, we control the indicators IN to fall in the range of [1, 7], which can then be converted from decimal numbers to three-bit binary numbers. We need to first convert all the indicators IN to binary representations and form an indicator stream. Subsequently, we encode the indicator stream using the (7,4) Hamming coding method. A three-bit parity code is generated for every four bits of data. The details of the encoding process were described in the Related Work section earlier.

Figure 4 shows how the indicators are converted to an indicator stream and how indicators are encoded. As shown in Figure 4, the indicators are first converted from decimals to three-bit binary streams and form the indicator stream 111100001011. Subsequently, we divide the indicator stream into four-bit groups and encode each group by using (7,4) Hamming coding method to obtain the encoded indicator stream. The encoded indicator stream we obtain from the example is 1111111 0000000 0110011.

Since the range of IN is [1, 7], $IN = 0$ is used as a flag to terminate the indicator data stream. In other words, encoding $IN = 0$ means that our indicator data stream has been terminated. When we decode $IN = 0$, indicator data stream has been fully decoded and the secret data stream follows.

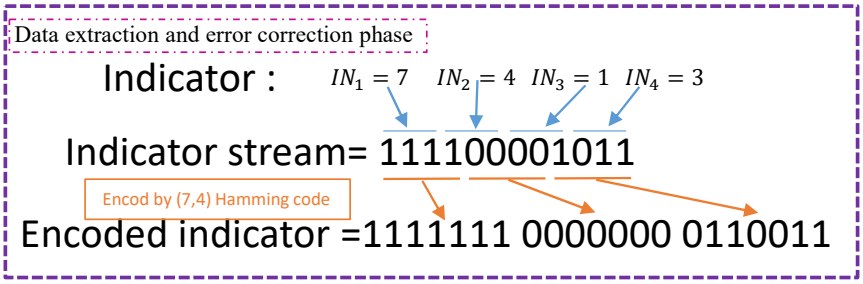


Figure 4. Examples of generating encoded indicators.

3.3. The Data Embedding Phase

Once the secret data and indicators are encoded, we move on to the phase of data embedding. We use the simple but efficient data embedding method, least significant bit (LSB) substitution, to hide the secret data. Initially, we concatenate the encoded indicators stream with the encoded secret data stream to obtain the resulting data stream that we want to embed in the cover image. For each pixel of the cover image, we sequentially replace its least significant bits (LSBs) with a bit from the encoded data stream. If the size of the result data stream is large, we replace multiple LSBs of each pixel with the multiple bits from the result data stream sequentially. After hiding the result data stream, we receive the desired stego image for uploading.

3.4. Data Extraction and Error Correction Phase

When the receiver downloads the stego image from the cloud, the receiver can read the encoded indicator stream and the encoded secret data stream directly from the LSBs of the stego image pixels. The receiver should extract the indicators IN first to subsequently guide the verification, correction, and extraction of secret data. The receiver needs to group the encoded indicator stream into groups and each group consists of seven bits. Subsequently, the receiver recalculates the three-bit check code in each group using Equations (4)–(6) and performs the process of data verification and error correction using (7,4) Hamming codes which were detailed in the Related Work section.

Next, the receiver removes the three-bit parity code from the encoded indicator stream and acquires the pure indicator stream, and then regroups the indicator stream into three-bit groups. After converting the three-bit indicator stream in each group into decimal form, the indicators IN are subsequently used to guide the verification, correction, and extraction of secret data in the group.

Then, the receiver can deduce the length of the j-th group of the encoded secret data streams based on the indicator IN_j. Since the purpose of the copy encoding method is to apply the "majority principle" to perform data verification and error correction on the data stream, the number of data bits corresponding to the indicators in the encoded group should be odd numbers and greater than one. According to the parity of the indicator IN_j, we can easily deduce the length of the corresponding encoded secret data group. If the IN_j is an even number, it means that we copied one bit from the j-th group of secret data. Therefore, the $IN_j + 1$ bits of data should be in the j-th group when extracting. If the IN_j is an odd number and not one, it indicates that the j-th group was unchanged after encoding and the IN_j bits of data can be read as the encoded data for the j-th group. If the indicator $IN_j = 1$, it means that we copied the secret data of the j-th group twice. Therefore, when extracting, we should read $IN_j + 2$ bits of data as the encoded data for the j-th group. After the grouping is completed, the receiver needs to verify and correct the encoded data in each group. According to the proposed copy encoding rules, the elements within a group should remain the same for correct transmission. Therefore, if there is a mixture of bit value zero and bit value one in a group, it means that an error occurred during the transmission. If the amount of the encoded data in a group is an odd number and greater than one, it provides a basis for error correction using voting strategy. It is like voting in an election:

the value of each element in the group represents what it supports initially. In the situation of correction, the value of the majority votes in the group is 'the leader' of the group, and all elements in the group should follow 'the leader' and correct their values to be consistent with 'the leader'. Through the voting strategy, the values of the elements in a group should be consistent with the value of the majority votes. After the error correction is completed, the receiver deletes the extra bits added during the encoding based on the indicators IN to complete the extraction of the secret data. The algorithm for data extraction and decoding is shown in Algorithm 2. The time complexity of the decoding process of secret data is O(N).

Algorithm 2. Data extraction and decoding algorithm.

Input: Indicator IN, encoded data $\mathcal{S}' = s'_1 s'_2 s'_3 \cdots s'_n$
Output: Secret data $\mathcal{S}$

for $j = 1$: *end* **do**
  if $\mathrm{mod}(IN_j, 2) == 0$
    add $(IN_j + 1)$ bits encoded data to $group_j$
  else if $\mathrm{mod}(IN_j, 2) == 1$ && $IN_j == 1$
    add $(IN_j + 2)$ bits encoded data to $group_j$
  else if $\mathrm{mod}(IN_j, 2) == 1$ && $IN_j \neq 1$
    add IN_j bits encoded data to $group_j$
  end if
  if the data of $group_j$ is all 0′s or all 1′s,
    no error occurred in $group_j$.
    add IN_j bits data in $group_j$ to $\mathcal{S}$
  else if the data of $group_j$ is not all 0′s or all 1′s,
    error occurred in $group_j$.
    correct the error according to the majority principle
    add IN_j bits data in $group_j$ to $\mathcal{S}$
  end if
end for
return $\mathcal{S}$

Figure 5 provides an example of data extraction and error correction. First, the receiver groups the encoded indicator stream into seven-bit groups. In the example, we can divide the encoded indicator stream into three groups. According to the rules of (7,4) Hamming code, the three-bit parity codes are calculated respectively. Then, the data bits in the group are error corrected based on the three-bit parity codes. Removing the parity codes in groups to obtain the pure indicator stream 111100001011, the receiver divides the pure indicator stream into three-bit groups. In the example, we can divide the pure indicator stream into four groups and convert these groups into decimals to obtain IN_j (j = 1,2,3,4). Since $IN_1 = 7$ is an odd number, the first seven bits of data form the first group. $IN_2 = 4$ and four is an even number, so the next $IN_2 + 1 = 5$ bits of data become the second group. When the indicator of the third group $IN_3 = 1$, the $IN_3 + 2 = 3$ bits of data read convert to the third group. Finally, $IN_4 = 3$ is an odd number, and the next three bits of data belong to the fourth group. Following the above steps, we have grouped the encoded secret data stream. The receiver can calculate the proportion of elements in the group, figure out where the errors occurred and use a voting strategy to correct the errors based on the "majority principle". As shown in Figure 5, there are six bits with a value of zero and one bit with a value of one in the first group. The voting operation is based on the "majority principle" to determine whether an element is right or wrong. It meets the principle of majority voting that there are six bits out of seven with a bit value of zero; therefore, the bit with a value of one can be replaced by a bit with value zero. After the correction, according to $IN_1 = 7$, seven-bit information is read from the corrected data stream as the extracted secret data. The other two groups with errors were corrected and extracted using the same method. The receiver can obtain the final secret data stream 000000000001000 after all four groups are extracted and corrected.

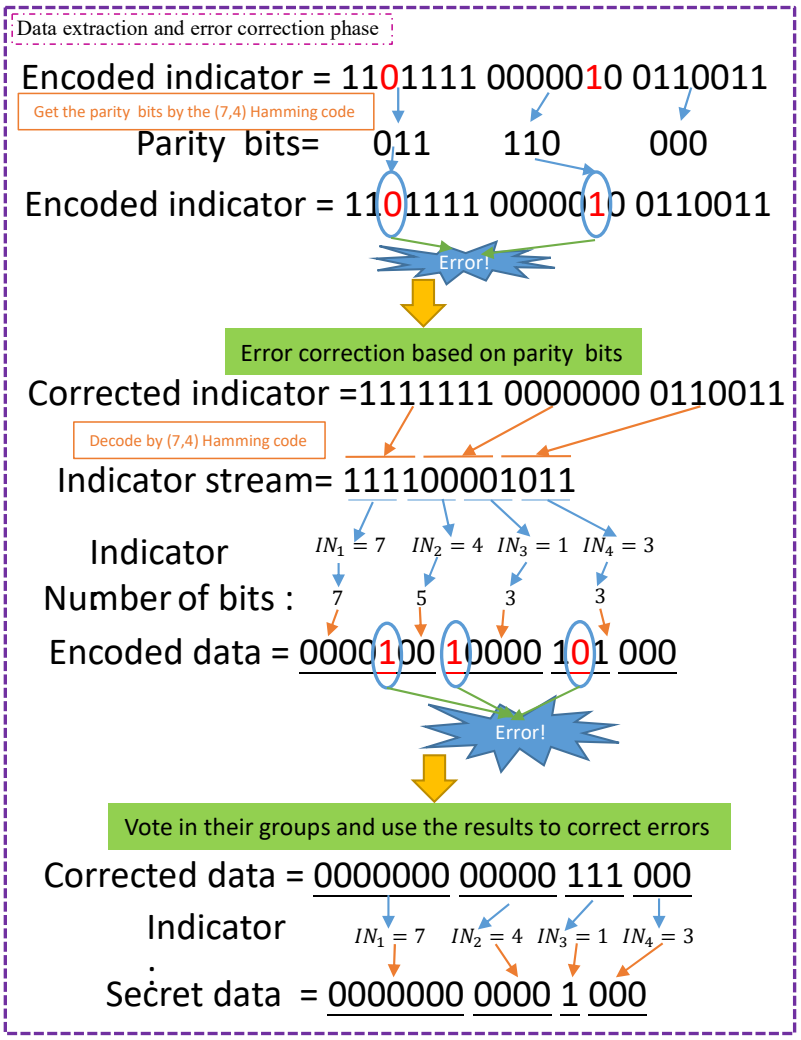


Figure 5. Example of extracting data and correcting errors based on indicators.

4. Experimental Results and Analysis

In this section, we present the results of the experimental evaluations conducted on the proposed data hiding scheme. Our tests encompassed several color images. Each color image comprises of 3 channels: red, green, and blue, and each channel has a size of 512×512 pixels. During the data embedding process, we embedded data into the least significant bits (LSBs) of the pixels in the red channel, the green channel, and the blue channel sequentially. The experiments were conducted on a Windows PC utilizing MATLAB software, version 2017a. Our test set includes colored images "Airplane", "Baboon", "Lake", "Lena", and "Peppers".

A stream of binary data generated by a random number generator as the secret data will be encoded and embedded into the cover image. The effectiveness of the proposed scheme is assessed based on embedding capacity (EC), peak signal-to-noise ratio (PSNR), Structural Similarity (SSIM), Quality Index (QI), and entropy (EN). EC represents the quantity of secret data that a method can embed, where a larger EC indicates that the method can transmit more information in a carrier, thereby possessing higher transmission efficiency. Assuming the carrier is a cover image, EC can be the number of embedded secret data bits or calculated bits per pixel (bpp) as the capacity measurement. PSNR and SSIM, expressed in decibels (dB), are the two primary metrics used to evaluate the image

distortion between the original image and stego one. The calculations for PSNR and SSIM are as follows:

$$\text{MSE} = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(C_{ij} - S_{ij} \right)^2, \tag{12}$$

$$\text{PSNR} = 10 log_{10} \frac{(255)^2}{\text{MSE}} (\text{dB}), \tag{13}$$

$$\text{SSIM} = \frac{(2\mu_C \mu_S + c_1)(2\sigma_{CS} + c_2)}{\left[(\mu_C)^2 + (\mu_S)^2 + c_1 \right] \left[(\sigma_C)^2 + (\sigma_S)^2 + c_2 \right]}. \tag{14}$$

In Equation (12), $M \times N$ is the size of the cover image, C_{ij} represents the pixel values of the cover image, and S_{ij} represents the pixel values of the stego image. A higher PSNR value indicates smaller distortion caused by the hidden data, and the stego image closely resembles the cover image. Generally, a PSNR value greater than 30 dB suggests that the information embedding has caused imperceptible distortion to human eyes, making it an acceptable result.

In Equation (14), μ represents the mean and serves as an estimate of brightness; σ is the standard deviation and serves as an estimate of contrast; σ_{CS} represents the covariance between the cover image C and the stego image S and serves as a measure of structural similarity. Additionally, c_1 and c_2 are two constants. SSIM combines factors related to brightness, contrast, and structure to comprehensively assess the similarity between two images. SSIM values can range from -1 to 1; with values close to 1 indicating that the stego image is highly similar to the cover image.

QI is another metric used for evaluating images' similarity. we adopted the QI settings as described in [16]. The calculation formula for QI is as follows:

$$\text{QI} = \frac{4 \times \overline{C} \times \overline{S} \times \left\{ \sum_{i=1}^{M} \sum_{j=1}^{N} \left(C_{ij} - \overline{C} \right) \times \left(S_{ij} - \overline{S} \right) \right\}}{\left\{ \sum_{i=1}^{M} \sum_{j=1}^{N} \left(C_{ij} - \overline{C} \right)^2 + \sum_{i=1}^{M} \sum_{j=1}^{N} \left(S_{ij} - \overline{S} \right)^2 \right\} \times \left(\overline{C}^2 + \overline{S}^2 \right)}, \tag{15}$$

where $\overline{C}$ means the average value of the cover image pixels and $\overline{S}$ is the average value of the stego image pixels. The higher the value of QI, the smaller the difference between the cover image and the stego image.

The entropy (EN) can represent the characteristics of an image's gray scale distribution. The more similar the entropy of two images, the more similar they are. The entropy (EN) is calculated as follows:

$$\text{entropy} = \sum_{pi=0}^{255} \theta(pi) \log \frac{1}{\theta(pi)}, \tag{16}$$

where pi represents the image pixel value between 0 and 255, and $\theta(pi)$ is the probability of the image pixel value pi.

Tables 2–4 recorded the performance of the proposed scheme across various metrics for different embedding capacities on different test images. Upon reviewing the data in these three tables, it became evident that the proposed scheme performs quite well when embedding a small amount of secret data. For instance, when embedding 100,000 bits of information, the average PSNR across the 5 test images reached an impressive 56 dB, with an SSIM of 0.9996, and a QI value approaching 1. Additionally, the entropies of the stego images were also close to those of the original cover images. This indicates that the proposed scheme produces stego images highly similar to the original cover images when embedding a low amount of data.

Table 2. Performance of the proposed scheme with an embedding capacity of 100,000 bits.

Image Name	PSNR	SSIM	QI	EN_{in}	EN_{out}
Airplane	56.0251	0.9987	0.9999	6.6639	6.6634
Baboon	56.0270	0.9999	0.9999	7.7624	7.7621
Peppers	56.0138	0.9999	0.9999	7.6698	7.6694
Lake	56.0319	0.9998	0.9999	7.7622	7.7619
Lena	56.0504	0.9999	0.9999	7.7494	7.7502
Average	56.0296	0.9996	0.9999	-	-

Table 3. Performance of the proposed scheme with an embedding capacity of 500,000 bits.

Image Name	PSNR	SSIM	QI	EN_{in}	EN_{out}
Airplane	47.2959	0.9916	0.9994	6.6247	6.6634
Baboon	47.2913	0.9993	0.9996	7.7177	7.7621
Peppers	47.3644	0.9995	0.9997	7.6465	7.6694
Lake	47.2969	0.9985	0.9998	7.7169	7.7619
Lena	47.2799	0.9995	0.9997	7.7001	7.7502
Average	47.3057	0.9977	0.9996	-	-

Table 4. Performance of the proposed scheme with an embedding capacity of 1,000,000 bits.

Image Name	PSNR	SSIM	QI	EN_{in}	EN_{out}
Airplane	35.7220	0.8984	0.9904	6.4677	6.6634
Baboon	35.7160	0.9899	0.9942	7.4948	7.7621
Peppers	35.7467	0.9931	0.9957	7.4445	7.6694
Lake	35.7260	0.9794	0.9960	7.4828	7.7619
Lena	35.7370	0.9932	0.9946	7.4321	7.7502
Average	35.7295	0.9708	0.9942	-	-

As the size of the embedded data stream increases, the performance metrics of the stego images generated by the proposed scheme exhibit a decreasing trend. Even when embedding 1,000,000 bits of secret data, the scheme still produced stego images with an average PSNR of 35 dB, an SSIM of 0.97, and a QI maintained at 0.9942. The entropies of the stego images also did not differ significantly from those of the original images. It implies that when embedding a substantial amount of secret data, the proposed scheme introduces limited distortion and exhibits good performance in terms of structural similarity against the original images. The overall performance is commendable.

Table 5 shows the time taken for encoding and decoding using different binary images as secret data. The binary images used as secret data are of size 256×256. From Table 4, it can be noticed that the proposed copy encoding method takes a longer time to encode and decode for complex secret data. However, the longest encoding time is only 0.81 s and the longest decoding time is only 0.58 s. Since there are more indicators required for the complex secret data, the indicators encoded and decoded with (7,4) Hamming code take more time as well. For the smooth secret data, the proposed copy encoding method takes about 0.2 s to encode all the secret data and 0.3 s to decode them. Overall, the proposed copy encoding method is simple and efficient in encoding and decoding.

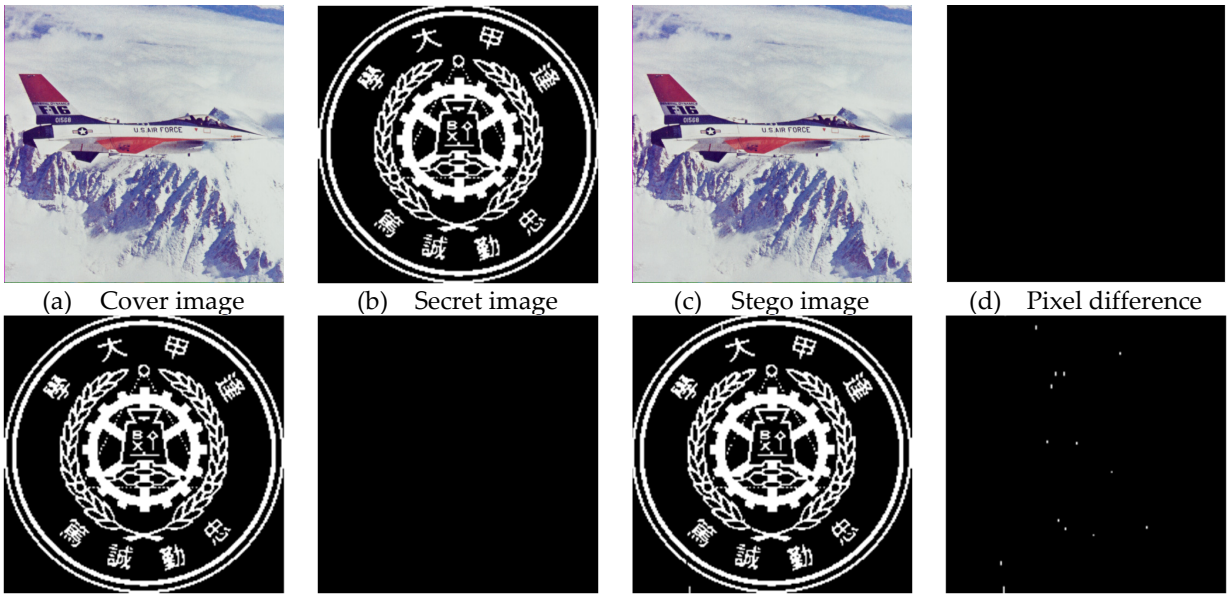
The focus of the proposed scheme is on error detection and correction. The scheme utilizes a copy encoding method along with a voting-based error-correction approach. When fewer than half of the data bits in a group are erroneous, the scheme can accurately detect them and correct them. As illustrated in Figure 5, the error correction strategy of the proposed method effortlessly rectifies the data if the number of errors does not exceed half of the data bits in the group.

Table 5. Encoding and decoding time for indicators and secret data. (Unit: second.)

Image Name	Indicators Encoding	Secret Data Encoding	Indicators Decoding	Secret Data Decoding
Airplane	0.74	0.26	0.43	0.35
Baboon	1.59	0.81	0.75	0.58
Barbara	1.01	0.45	0.54	0.42
Boat	0.74	0.26	0.43	0.33
Couple	0.87	0.31	0.47	0.35
Lena	0.74	0.23	0.42	0.31
Peppers	0.72	0.23	0.42	0.31

Figure 6 shows the visualization results of the proposed scheme before and after embedding the encoded secret image into the cover image. As can be seen from (a) and (c), it is difficult for a malicious user to find out that (c) is an image with the embedded secret image (b). (d) is the pixel difference between the stego image and the original cover image. From (d), we can find it is almost impossible for people to find out the difference between the original cover image and the stego image at the pixel level. In general, people will treat the stego image as a cover image and not pay much attention to it. If a malicious user still chooses to attack the stego image by adding salt and pepper noise to the image in order to destroy the information of the image, the secret data image encoded using the proposed method can be mostly correctly extracted. Salt and pepper noise, randomly appearing as white or black dots, is a common type of noise in images. These noises may be black pixels in bright areas, white pixels in dark areas, or both. Table 6 shows the correctness percentages of the secret data extracted from the stego image after the stego image suffered from different densities of salt and pepper noise attacks. When the stego image is attacked by the salt and pepper noise with a density of 0.01, the extracted data can be corrected by the proposed scheme to obtain a secret image that is almost the same as the original secret image, and the difference between them is insignificant. Figure 6e,f demonstrate the observation. When the stego image is subjected to a salt and pepper noise attack with a density of 0.01, the indicator data stream encoded using the (7,4) Hamming code possesses 99.56% correctness after correction; additionally, the secret data stream encoded using the copying encoding method possesses 100% correctness after correction, as shown in Table 6. While the density of the salt and pepper noise attack is increased, the correctness of the indicator data and the secret data is decreased. However, we can still extract the indicator data from the stego image with a 94.83% correctness rate and the secret data with a 99.85% correctness rate, even after the stego image has suffered a salt and pepper noise attack with a density of 0.1. From Figure 6g,h, when the stego image is subjected to the salt and pepper noise attack, the data image extracted from a stego image is very similar to the original one. It means that the attack does not affect our ability to obtain the approximate information on the stego image. In summary, the proposed scheme has an acceptable error correction efficiency, and the secret data can have a correctness rate of about 99%.

Salt and pepper noise attacks on stego images do not necessarily make the embedded data erroneous. In Table 7, we model the case where the error occurs in the extracted data stream. Table 7 demonstrates the accuracy of the information extracted by the proposed scheme under different proportions of error rates. The error rate is the percentage of error bits in the extracted data. γ_{ind} and γ_{data} denote the accuracy rates for the indicator extraction and the secret data extraction, respectively. From Table 7, it can be found that the indicators encoded by using (7,4) Hamming code can be extracted 88.7% correctly by the error correction mechanism even when 10% of the bits are erroneous. On the other hand, the proposed method can extract 99.7% of the correct information through the error correction mechanism in cases where there are 10% of error bits. That is to say, the error correction efficiency of the proposed scheme is so high that the information can be extracted almost error-free, even in cases where 10% of the bits are in error.



(a) Cover image (b) Secret image (c) Stego image (d) Pixel difference

(e) Noise density = 0.01 (f) Data difference (g) Noise density = 0.1 (h) Data difference

Figure 6. Visual presentation of stego image and extracted secret images. Part (**a**) is the cover image; (**b**) is the secret image; (**c**) is the stego image; (**d**) is the difference between the cover image and stego image; (**e**,**g**) are the extracted secret images after error correction when the noise density is 0.01 and 0.1; (**f**,**h**) are the differences between the extracted image and the original secret image.

Table 6. Correctness of extracted data under different densities of salt and pepper noise attack.

Noise Density	γ_{ind}	γ_{data}
0.01	99.56%	100.00%
0.02	99.07%	100.00%
0.03	98.62%	99.99%
0.04	97.94%	99.98%
0.05	97.49%	99.94%
0.06	96.75%	99.97%
0.07	96.26%	99.90%
0.08	95.89%	99.94%
0.09	95.08%	99.92%
0.1	94.83%	99.85%

Table 7. Error correction rates of the proposed scheme under different error rates.

Error Rate	γ_{ind}	γ_{data}
1%	99.08%	99.99%
3%	96.79%	99.97%
5%	94.80%	99.86%
10%	88.87%	99.34%

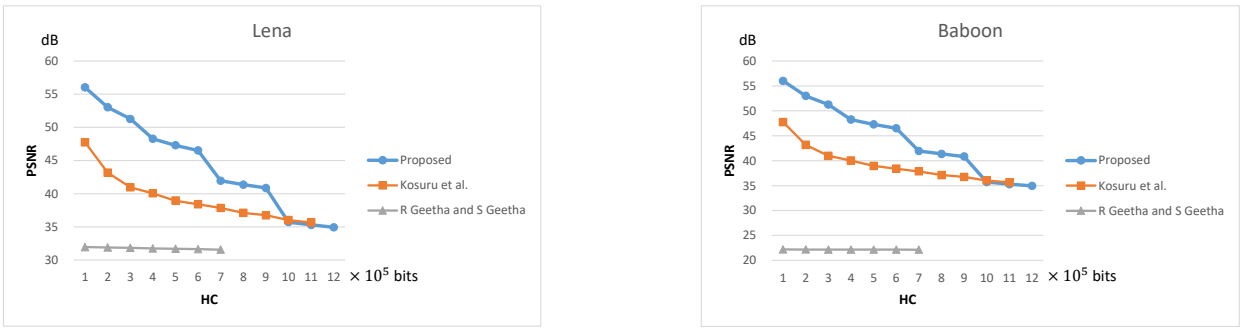
Table 8 presents the performance of the proposed scheme in comparison to several existing IRDH methods. A "×" denotes the absence of the feature, while a "✓" signifies its implementation. The Hiding Capacity (HC) represents the total amount of data a scheme can hide in an image without deducting redundant data bits used for error detection and correction. Pradhan et al. [11] proposed a PVD technique using 3 × 3 size blocks. The concept of QVD was proposed by Jung [12]. Although this method obtained high HC values, it suffered from overflow and extraction errors. Pradhan et al. [13] combined additive–subtraction-based Quotient Value Difference (ASQVD) with neighbor matching to improve the HC values. Sonar and Swain [14] combined pixel value correlation with

QVD to further improve HC. However, these methods focus on the data hiding aspect and do not consider error detection, let alone error correction. Swain and Pradhan [15] proposed a hybrid approach using QVD and quotient value correlation that allows data integrity verification at the receiver. From the table, there are methods that do not consider error detection and correction, such as [11–14], have advantages in HC, PSNR, and QI because they do not require extra bits for information detection. Unfortunately, no one can determine whether the data transmitted using these schemes is reliable or not. On the other hand, methods that take error detection and correction into account inevitably sacrifice some hiding capacity to record detection codes. Despite having similar PSNR values, these methods including the proposed one have smaller HCs compared to conventional IRDH methods. Although the method in [15] exhibits high PSNR and QI values, it can only detect data errors but not correct them. As for [16], while it slightly outperforms the proposed scheme in terms of PSNR and QI, it can only correct 1 error for every 12 bits of data and results in lower error correction efficiency. The proposed scheme improves error correction efficiency substantially as it can accurately detect and correct errors when fewer than half of the data bits in a group are erroneous. If there are three bits in a group, it can correct at least one error; with five bits, it can correct two errors; with seven bits, it can correct three errors. Compared to [16], the proposed scheme exhibits a significant enhancement to error correction efficiency.

Table 8. Performance comparison of the proposed scheme with other methods.

Method	PSNR	QI	HC	Error Detection	Error Correction
Pradhan et al. [11]	39.78	0.9985	1,879,572	×	×
Jung [12]	35.27	0.9967	2,757,637	×	×
Pradhan et al. [13]	33.02	0.9947	2,794,301	×	×
Sonar and Swain [14]	35.15	0.9966	3,086,396	×	×
Swain and Pradhan [15]	39.74	0.9988	1,738,014	✓	×
Kosuru et al. [16]	36.78	0.9977	2,359,296	✓	✓
Proposed	35.73	0.9942	2,551,826	✓	✓

To further analyze the performance, we compare the proposed scheme with [7] which supports error detection, and [16] which supports both error detection and error correction. As depicted in Figure 7, the figure illustrates the PSNR performance of several methods when embedding varying amounts of secret data in two test images. It is evident that the proposed method imposes less damage to the images and achieves higher PSNR values when embedding fewer secret data. The proposed scheme uses LSB replacement for secret data embedding. There can be one or two LSBs needed to complete the data embedding and check information depending on the data amount. When the embedded data are less, the required checking information is less. The embedding method used in [7] is Intermediate Significant Bit substitution (ISB), which embeds the secret data in the intermediate significant position and preserves the LSB. Therefore, the ISB method causes more distortion. The embedding method used in [16] requires a three-bit LSB change to embed the data in the image and thus, the PSNR of the stego image will be even lower. However, as the volume of secret data exceeds 10×10^5 bits, the proposed scheme requires more redundant bits for error detection and correction. The increment of the bits requires more LSBs to embed and results in greater image distortion. In summary, for scenarios involving a relatively small volume of transmitted secret data, the proposed scheme offers better PSNR and higher error-correction efficiency.



(**a**) (**b**)

Figure 7. Comparison of the proposed scheme with other schemes capable of error detection and correction. (**a**) Lena; (**b**) Baboon.

5. Conclusions

The majority of existing data hiding techniques primarily focus on enhancing the embedding capacity of methods and the visual quality of the stego images. However, there is limited research emphasizing on the ability of detecting and correcting errors during the extraction process at the receiver's end. The proposed approach introduces a simple and efficient data hiding method capable of detecting and correcting errors while maintaining comparable embedding capacity and PSNR. The proposed scheme first groups adjacent and identical data in the secret data and records the count of data in each group as indicators. Subsequently, it encodes the secret data using a copy encoding method. The indicators are then encoded using the (7,4) Hamming code. Finally, the encoded secret data and the encoded indicators are embedded one after the other in the cover image. The receiver can read the indicators first to group the data and then employs a voting-based error detection and correction method to verify the correctness of the extracted data. Experimental results show that when the secret data to be encoded is smooth, encoding takes about 0.2 s and decoding takes about 0.3 s. When the secret data is complex, the time spent on encoding and decoding is also less than 1 s. The data extracted and corrected by the proposed copy encoding method can reach a 99% correctness rate even with a noise attack on the stego image. It can be concluded that the proposed copy encoding method is a simple and efficient coding method with error detection and correction capability. Meanwhile, the proposed method outperforms other schemes in terms of PSNR when embedding secret data is less than 10×10^5 bits.

A limitation of this work is the generation of significant redundancy during the encoding process due to the need for indicators to guide subsequent data extraction. In the future, we will explore various strategies to reduce redundancy without compromising error correction efficiency.

Author Contributions: Conceptualization and methodology, H.C., J.-C.L., C.-C.C. and J.-H.H.; software, H.C.; validation, H.C., J.-C.L., C.-C.C. and J.-H.H.; data curation, H.C.; writing—original draft preparation, H.C.; writing—review and editing, J.-C.L. and H.C.; supervision, C.-C.C. and J.-H.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

Abbreviated terms used in the paper and their full names:

LSB	Least Significant Bit
VR	Virtual Reality
AR	Augmented Reality
5G	the Fifth Generation
DH	Data Hiding
RDH	Reversible Data Hiding
IRDH	Irreversible Data Hiding
CRC	Cyclic Redundancy Check
PSNR	Peak Signal-to-noise Ratio
QVD	Quotient Value Differencing
EC	Embedding Capacity
SSIM	Structural Similarity
QI	Quality Index
EN	Entropy
ISB	Intermediate Significant Bit

References

1. Yu, C.; Zhang, X.; Li, G.; Zhan, S.; Tang, Z. Reversible data hiding with adaptive difference recovery for encrypted images. *Inf. Sci.* **2022**, *584*, 89–110. [CrossRef]
2. Li, L.; Yao, Y.; Yu, N. High-fidelity video reversible data hiding using joint spatial and temporal prediction. *Signal Process.* **2023**, *208*, 108970. [CrossRef]
3. Kim, C. Dual Reversible Data Hiding Based on AMBTC Using Hamming Code and LSB Replacement. *Electronics* **2022**, *11*, 3210. [CrossRef]
4. Wu, X.; Yang, C.-N.; Liu, Y.-W. A general framework for partial reversible data hiding using hamming code. *Signal Process.* **2020**, *175*, 107657. [CrossRef]
5. Celik, M.U.; Sharma, G.; Tekalp, A.M.; Saber, E. Lossless generalized-LSB data embedding. *IEEE Trans. Image Process.* **2005**, *14*, 253–266. [CrossRef] [PubMed]
6. Kim, C.; Yang, C.-N.; Zhou, Z.; Jung, K.-H. Dual efficient reversible data hiding using Hamming code and OPAP. *J. Inf. Secur. Appl.* **2023**, *76*, 103544. [CrossRef]
7. Geetha, R.; Geetha, S. Embedding electronic patient information in clinical images: An improved and efficient reversible data hiding technique. *Multimed. Tools Appl.* **2020**, *79*, 12869–12890. [CrossRef]
8. Faheem, Z.B.; Ali, M.; Raza, M.A.; Arslan, F.; Ali, J.; Masud, M.; Shorfuzzaman, M. Image Watermarking Scheme Using LSB and Image Gradient. *Appl. Sci.* **2022**, *12*, 4202. [CrossRef]
9. Almazaydeh, L. Secure RGB image steganography based on modified LSB substitution. *Int. J. Embed. Syst.* **2020**, *12*, 453–457. [CrossRef]
10. Mahmoud, M.M.; Elshoush, H.T. Enhancing LSB Using Binary Message Size Encoding for High Capacity, Transparent and Secure Audio Steganography–An Innovative Approach. *IEEE Access* **2022**, *10*, 29954–29971. [CrossRef]
11. Pradhan, A.; Raja Sekhar, K.; Swain, G. Digital Image Steganography based on Seven Way Pixel Value Differencing. *Indian J. Sci. Technol.* **2016**, *9*, 1–11. [CrossRef]
12. Jung, K.-H. Data hiding scheme improving embedding capacity using mixed PVD and LSB on bit plane. *J. Real-Time Image Process.* **2018**, *14*, 127–136. [CrossRef]
13. Pradhan, A.; Sekhar, K.R.; Swain, G. Image steganography using add-sub based QVD and side match. In *Digital Media Steganography*; Hassaballah, M., Ed.; Academic Press: Cambridge, MA, USA, 2020; pp. 81–97. [CrossRef]
14. Sonar, R.; Swain, G. Steganography based on quotient value differencing and pixel value correlation. *CAAI Trans. Intell. Technol.* **2021**, *6*, 504–519. [CrossRef]
15. Swain, G.; Pradhan, A. Image Steganography Using Remainder Replacement Adaptive QVD and QVC. *Wirel. Pers. Commun.* **2022**, *123*, 273–293. [CrossRef]
16. Kosuru, S.N.V.J.D.; Pradhan, A.; Basith, K.A.; Sonar, R.; Swain, G. Digital Image Steganography with Error Correction on Extracted Data. *IEEE Access* **2023**, *11*, 80945–80957. [CrossRef]
17. Darabkh, K.A.; Al-Dhamari, A.K.; Jafar, I.F. A New Steganographic Algorithm Based on Multi Directional PVD and Modified LSB. *Inf. Technol. Control* **2017**, *46*, 1. [CrossRef]
18. Cintas-Canto, A.; Kermani, M.M.; Azarderakhsh, R. Error Detection Constructions for ITA Finite Field Inversions Over GF(2^m) on FPGA Using CRC and Hamming Codes. *IEEE Trans. Reliab.* **2023**, *72*, 651–661. [CrossRef]
19. Chan, C.-S.; Chang, C.-C. An efficient image authentication method based on Hamming code. *Pattern Recognit.* **2007**, *40*, 681–690. [CrossRef]

20. Kumar, U.K.; Umashankar, B.S. Improved Hamming Code for Error Detection and Correction. In Proceedings of the 2007 2nd International Symposium on Wireless Pervasive Computing, San Juan, PR, USA, 5–7 February 2007. [CrossRef]
21. Sobolewski, J.S. Cyclic redundancy check. In *Encyclopedia of Computer Science*; GBR: John Wiley and Sons Ltd.: Hoboken, NJ, USA, 2003; pp. 476–479.
22. Castagnoli, G.; Brauer, S.; Herrmann, M. Optimization of cyclic redundancy-check codes with 24 and 32 parity bits. *IEEE Trans. Commun.* **1993**, *41*, 883–892. [CrossRef]
23. Li, B.; Shen, H.; Tse, D. An Adaptive Successive Cancellation List Decoder for Polar Codes with Cyclic Redundancy Check. *IEEE Commun. Lett.* **2012**, *16*, 2044–2047. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.