An IoT-Based Electronic Health Protection Mechanism With AMBTC Compressed Images

Yijie Lin, Chia-Chen Lin, *Member, IEEE*, Ching-Chun Chang, and Chin-Chen Chang, *Fellow, IEEE*

***Abstract*—Since the COVID-19 outbreak, there has been a growing need for contactless healthcare to meet medical diagnosis demands. Electronic health systems using the Internet of Things (IoT) are rapidly advancing, transmitting significant amounts of private medical data online. In telemedicine, where patients are diagnosed remotely, sensitive information, such as patient records, may be embedded into medical images for security purposes. Due to the large file sizes of medical images produced by equipment, compression is essential for fast transmission. To safeguard medical images in telemedicine and address bandwidth limitations, we utilize data hiding techniques and absolute moment block truncation coding (AMBTC) compression to introduce an IoT-driven electronic health protection mechanism. Our mechanism employs diverse methods to compress and embed data across various image blocks. Additionally, it offers a flexible adaptation to meet different application requirements concerning embedding capacity, visual quality, and file size by adjusting thresholds and variants. Compared to alternative methods, our approach delivers superior payload capacity and efficiency while preserving visual fidelity.**

***Index Terms*—Absolute moment block truncation coding (AMBTC), data hiding, electronic health system, Internet of Things (IoT), medical images, security protection.**

I. Introduction

IN THIS rapidly developing information age, daily life is inseparable from the Internet. A large amount of either personal or commercial confidential information is transmitted over the Internet every day. Therefore, people have begun to pay more and more attention to information security and privacy issues in the last decade. Data hiding [1], [2], [3], [4], [5] is an effective technique to secure confidentiality, privacy, integrity, copyright protection, and ownership authentication of the transmitted data under diverse scenarios. In general, data hiding is mainly divided into reversible data hiding (RDH) and irreversible data hiding. Among two different categories, irreversible data hiding fails to rebuild the original image but offers relatively high performance on payload and imperception. In contrast, RDH [6], [7], [8], [9], [10] is particularly suitable for application scenarios, in which there is a high demand that the original image must be restored after the hidden secrets are extracted.

Distinguished by domain, data hiding methods can be mainly divided into plaintext domain [11], [12], encrypted domain [13], [14], and compressed domain [15], [16], [17]. In this work, we focus on compressed domains. The main purpose is to pursue greater embedding capacity for concealing ownership information or electronic medical records (EMRs) and acceptable visual quality required for medical diagnosis. Vector quantization (VQ) algorithm [18] and absolute moment block truncation coding (AMBTC) algorithm [19] are two commonly used image compression algorithms. There have been many data hiding methods based on VQ compressed images [20], [21], [22] and AMBTC compressed images [23], [24], [25], [26] in the last decade, and we have chosen the AMBTC algorithm as the basis for our proposed scheme. Ou and Sun [23] introduced an AMBTC-based data hiding technique, categorizing blocks into smooth and complex categories. Data embedding for smooth blocks involves utilizing bit planes, while for complex blocks, it entails swapping the order of quantization levels and synchronously adjusting bit planes. Building upon this, Kumar et al. [24] further refined their approach by initially employing a smoothing filter to preprocess the original image. They subsequently classified image blocks into three categories: 1) smooth; 2) less complex; and 3) highly complex. For smooth blocks, data concealment relied on a simple replacement strategy, for less complex blocks, it involved Hamming distance calculation, and for highly complex blocks, a pixel value differencing method was applied. Concurrently, in the same period, Chang et al. [25] followed a comparable block classification approach to Kumar et al., yet diverged in their choice of data hiding methodologies. They employed replacement, matrix encoding, and symmetric quantization values tailored to each block type, respectively. Similarly, Chen et al. [26] adopted the identical block classification scheme but introduced a gradient-based (GB) compression method compatible with AMBTC compression. This approach enabled the utilization of GB compression and data embedding for blocks exhibiting significant gradient effects, while employing AMBTC compression and data embedding for remaining blocks.

With the evolution of Internet technology, electronic health systems leveraging the Internet of Things (IoT) are increasingly prevalent, especially in applications like telemedicine.

Received 12 July 2024; revised 9 September 2024; accepted 16 September 2024. Date of publication 30 September 2024; date of current version 24 January 2025. This work was supported by the National Science and Technology Council under Grant NSTC 113-2634-F-005-001-MBK and Grant 111-2410-H167-005-MY2. *(Corresponding author: Chia-Chen Lin.)*

Yijie Lin and Chin-Chen Chang are with the Department of Information Engineering and Computer Science, Feng Chia University, Taichung 40724, Taiwan (e-mail: p1263670@o365.fcu.edu.tw; ccc@o365.fcu.edu.tw).

Chia-Chen Lin is with the Department of Computer Science and Information Engineering, National Chin-Yi University of Technology, Taichung 41170, Taiwan (e-mail: ally.cclin@ncut.edu.tw).

Ching-Chun Chang is with the Information and Communication Security Research Center, Feng Chia University, Taichung 40724, Taiwan (e-mail: ccc@fcu.edu.tw).

Digital Object Identifier 10.1109/JIOT.2024.3467152

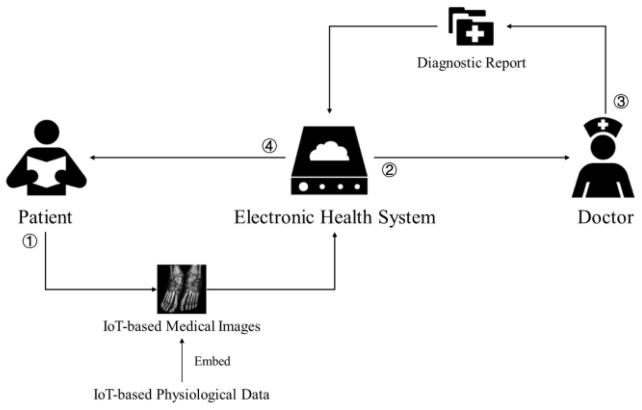


Fig. 1. Telemedicine application scenario of an IoT-based electronic health system.

These applications are particularly valuable during the COVID-19 era, given the surge in unexpected circumstances. For instance, due to home quarantine, both doctors and patients often resort to remote diagnosis via telemedicine. This process involves storing a vast number of medical images on servers, which are then transmitted over the Internet. Given the variability in the storage capacity of data servers and the bandwidth of telecommunication infrastructures, it becomes necessary for medical images to be compressed while preserving visual quality for accurate diagnoses, ensuring adaptability to diverse telemedicine setups. Moreover, the privacy of every patient is encapsulated within these images, which are under the ownership of the hospital. Therefore, safeguarding the integrity of visual data is imperative to thwart any attempts of hacking or unauthorized disclosure of medical images and patient information. Fig. 1 depicts a standard telemedicine configuration utilizing a cloud-based electronic health system. On the patient's end, IoT devices, such as wearable sensors, smart health monitors, and portable imaging tools, gather medical images and physiological data. This data is securely transmitted via encrypted channels to safeguard privacy and ensure data integrity. The doctor at the hospital then examines the images and data to provide a diagnostic report to the patient. However, if these IoT-based medical images and physiological data are transmitted sequentially, the transmission time doubles due to data duplication. This duplication escalates the risks of tampering, information leakage, or transaction disconnection. To address these concerns, we propose a data hiding scheme based on AMBTC compressed images. This scheme allows for the precompression of IoT-based patient medical images, with physiological data concealed within the compression results. This approach not only enhances transmission efficiency but also reduces transmission risks. Ultimately, the security of the IoT-based electronic health system is strengthened. The key contributions of our proposed scheme include the following.

1) We propose a threshold-based scheme that can flexibly adjust the range of block classification to adapt to different application scenarios.
2) We utilize different embedding strategies for various types of blocks, offering a higher payload than similar schemes, and achieving an efficiency improvement of over 20% in the best case scenario.
3) We propose two variants: one with indicators and one without. The former is suitable for application scenarios with loose requirements on compressed file size, while the latter is suitable for scenarios with strict requirements.
4) Due to the different processing details of block classification, our proposed scheme effectively preserves image features, resulting in a reconstructed image with visual quality discernible to the naked eye.

The remainder of this article is organized as follows. Section II provides a review of related work, Section III outlines our proposed data hiding scheme, Section IV presents the experimental results of our scheme, and finally, Section V concludes this article.

II. Related Work

This section covers the AMBTC compression algorithm, the side match VQ (SMVQ) compression algorithm, the turtle shell matrix data hiding algorithm, and the extended run-length encoding (ERLE) algorithm. The AMBTC algorithm compresses images by dividing them into blocks and using a combination of high and low values along with a bitmap. The SMVQ algorithm compresses VQ indices by employing a side match state codebook. The turtle shell matrix data hiding algorithm embeds data by modifying pixel pairs using a reference matrix. Finally, the ERLE compression algorithm reduces the size of sequences of continuous symbols, further optimizing the data compression. A detailed description of each technique mentioned above will be provided in the following section.

A. Absolute Moment Block Truncation Coding

The concept of AMBTC, introduced by Lema and Mitchell [19] in 1984, is a widely employed lossy compression technique in image processing. AMBTC aims to reduce the file size of images while preserving essential features as much as possible.

The encoding and decoding operations of AMBTC, depicted by (1)–(4), unfolds as follows. Initially, the mean value of a block, denoted as m, is computed using (1), where s denotes the block size, and i and j represent the coordinates of pixels within the current processing block. Subsequently, the pixels are categorized based on (2): if a pixel's value exceeds m, it is marked as "1" in the bitmap; otherwise, it is marked as "0." Consequently, a bitmap bm with dimensions $s \times s$ is generated. Using this bitmap, (3) computes the average value, labeled as the high value (hv), of all pixels marked as "1" in the bitmap, where n_1 signifies the total number of pixels in group "1." Similarly, (4) calculates the average value, denoted as the low value (lv), of pixels marked as "0" in the bitmap, with n_0 representing the total number of pixels in group "0." Ultimately, the image can be reconstructed utilizing (5) and p'_{ij} denoted as the reconstructed pixel, where i and j represent

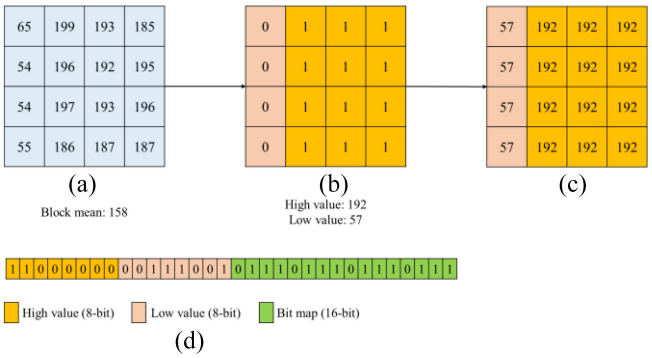


Fig. 2. Example of AMBTC encoding/decoding operations. (a) Original Block. (b) Bitmap. (c) Reconstructed Block.

the coordinates of pixels within the reconstructed block

$$m = \frac{1}{s \times s} \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} p_{ij} \tag{1}$$

$$bm_{ij} = \begin{cases} 1, \text{ if } p_{ij} > m \\ 0, \text{ if } p_{ij} \leq m \end{cases} \tag{2}$$

$$hv = \frac{1}{n_1} \sum_{\forall(p_{ij} > m)} p_{ij} \tag{3}$$

$$lv = \frac{1}{n_0} \sum_{\forall(p_{ij} \leq m)} p_{ij} \tag{4}$$

$$p'_{ij} = \begin{cases} hv, \text{ if } bm_{ij} = 1 \\ lv, \text{ if } bm_{ij} = 0. \end{cases} \tag{5}$$

Fig. 2 shows an example of an AMBTC compressed image. Fig. 2(a) depicts a block of the original image. According to (1), mean = 158 is calculated. Utilizing (2), the bitmap shown in Fig. 2(b) is obtained. Subsequently, $hv = 192$ and $lv = 57$ are calculated according to (3) and (4), respectively. Finally, the image is reconstructed as depicted in Fig. 2(c) using (5). Fig. 2(d) illustrates the data structure of a 32-bit AMBTC, achieving the compression effect significantly compared to the 128-bit block of the original image.

B. Side Match Vector Quantization

SMVQ, pioneered by Kim [27] in 1992, represents a lossy compression method within the VQ framework. In SMVQ, initial processing entails VQ encoding of blocks situated in the first column and row, utilizing a main codebook. Subsequently, given the significant correlation between adjacent blocks, residual block indices are generated based on their corresponding upper and left blocks, employing a state codebook.

Fig. 3 illustrates an instance of SMVQ encoding. Here, C denotes the block under current processing, while U and L denote its upper and left neighboring blocks, respectively. Both U and L have undergone VQ encoding, ensuring that all pixel values are known. In processing the current block C, begin by utilizing (6) to compute the value derived from the side match within C. Then, measure its Euclidean distance from all codeword side matches, selecting the nearest codeword from the state codebook to reconstruct the currently processed

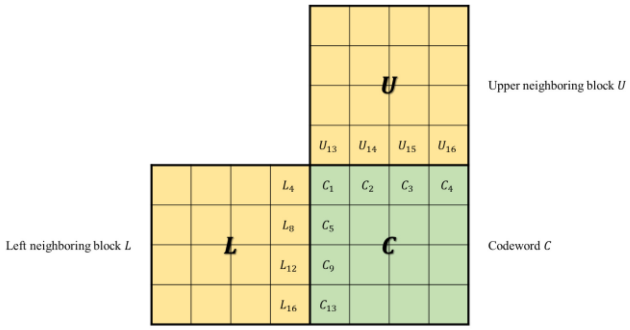


Fig. 3. Example of SMVQ encoding.

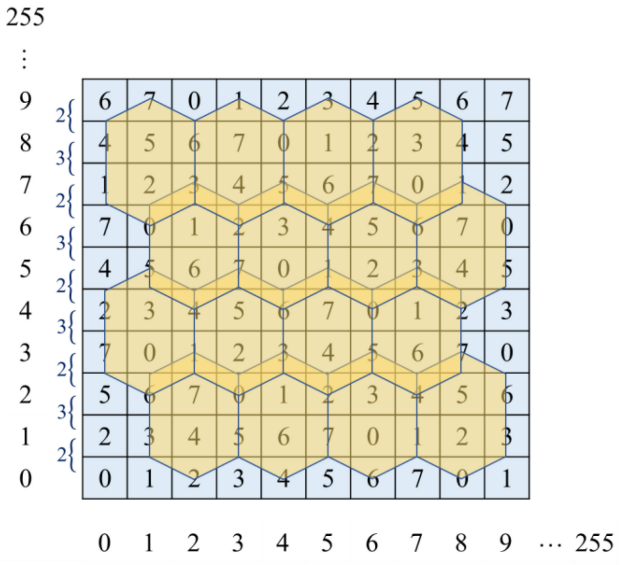


Fig. 4. Turtle shell matrix.

block C

$$\begin{cases} C_1 = \frac{L_4 + U_{13}}{2} \\ C_2 = U_{14} \\ C_3 = U_{15} \\ C_4 = U_{16} \\ C_5 = L_8 \\ C_9 = L_{12} \\ C_{13} = L_{16}. \end{cases} \tag{6}$$

C. Turtle Shell Matrix

Chang et al. [28] proposed a data hiding scheme based on a turtle shell matrix in 2014. As shown in Fig. 4, a 256×256 matrix is constructed as follows: the difference of elements in the same row is fixed to 1, while the difference of adjacent elements in the same column alternates between 2 and 3. In the reference matrix, there are many nonoverlapping hexagons, called turtle shells. It is observed that each turtle shell contains eight distinct elements, namely, the numbers 0–7. In other words, by using a pair of pixel values as matrix coordinates, an octal number on the turtle shell can always be determined.

In response to such characteristics, Chang et al. have designed specific embedding rules that allow carrying 3 bits in each pixel pair. Fig. 5 gives examples to illustrate, that the red circles represent the coordinates converted from the

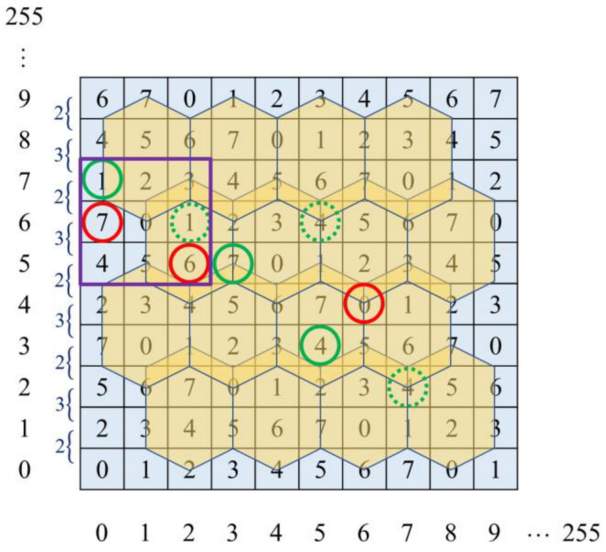


Fig. 5. Examples of turtle shell matrix.

original pixel pairs. In contrast, the green circles represent the coordinates that correspond to the secret message. Additionally, the green solid circles represent the final selected coordinates, whereas the green dotted circles represent the coordinates of the abandoned selection.

When considering coordinates within a single turtle shell, the pixel coordinate is directly adjusted to the position corresponding to the secret message. For instance, if a pixel pair, denoted as coordinate (2, 5) needs to carry a secret message of $(111)_2$, which translates to $(7)_8$. Then, the current coordinate (2, 5) is simply modified to (3, 5), indicating the element as 7. Thus, the stego pixel pair coordinate becomes (3, 5). When multiple shells are involved, the minimal Euclidean distance between the original and final coordinates is taken into account. For instance, consider the pixel pair coordinate (6, 4) carrying a secret message of $(100)_2$, corresponding to $(4)_8$. There are three candidate coordinates: (5, 3), (5, 6), and (7, 2), all mapping to the element denoted as 4. However, the final stego pixel pair coordinate is determined as (5, 3) because it is the closest to the original pixel pair coordinate (6, 4).

If the coordinates do not align with any turtle shell, a 3×3 matrix is constructed to determine the nearest location matching the secret message. For instance, consider the pixel pair coordinate (0, 6) tasked with carrying the secret message $(001)_2$ equivalent to $(1)_8$. Within the 3×3 matrix, two coordinates, (0, 7) and (2, 6), are found to map to the element 1. The coordinate (0, 7) is chosen as the stego pixel pair coordinate due to its proximity to (0, 6).

During extracting, the receiver only needs to convert the three stego pixel pairs, (3, 5), (5, 3), and (0, 7), into coordinates and obtain the corresponding values $(7)_8$, $(4)_8$, and $(1)_8$ from the turtle shell matrix, thus, the secret message is obtained $(111)_2$, $(100)_2$, and $(001)_2$, respectively.

D. Extended Run-Length Encoding

Chen and Chang [29] proposed the ERLE algorithm, where each run-length represents a continuous symbol, and compression is achieved by encoding the length.

ERLE operates in two modes: fixed- and variable-length codewords, distinguished by the fixed length parameter $L_{\text{fix}} = 4$. If the run-length parameter L_r is less than 4, fixed-length codewords are employed; otherwise, variable-length codewords are utilized. Illustrated in Fig. 6 are examples of ERLE. For instance, if the original sequence is "1100" and $L_r = 2 < 4$, fixed-length codewords are employed. The fixed length L_{fix} is directly scanned, and "0" serving as the prefix code, resulting in the output sequence "01100." During decoding, if the first bit is "0," it indicates the use of fixed-length codewords. Directly scanning L_{fix} bits yield the original sequence. If the original sequence is "111111111" and $L_r = 9 \geq 4$, variable-length codewords are employed Initially, calculate the prefix code length, which is 3 according to (7), starting with continuous "1"s and stopping at "0." Next, determine the length symbol $S_{\text{length}} = 9 - 2^3 = 1$ using (8), and represent it in binary with the same number of digits as L_{pre}, resulting in "001." The final bit is symbol "1," resulting in the output sequence "1100011." During decoding, if the first bit is "1," continue scanning until "0," thus "110" indicates $L_{\text{pre}} = 3$. Then, continue scanning 3 bits "001" and calculate $L_r = 2^3 + 1 = 9$ using (9). Finally, scan symbol "1" to reconstruct the original sequence "111111111"

$$L_{\text{pre}} = \lfloor \log_2(L_r) \rfloor \tag{7}$$

$$S_{\text{length}} = L_r - 2^{L_{\text{pre}}} \tag{8}$$

$$L_r = 2^{L_{\text{pre}}} + S_{\text{length}}. \tag{9}$$

III. Proposed IoT-Based Electronic Health Protection Mechanism

In this section, a novel IoT-based electronic health protection mechanism is introduced, utilizing the data hiding technique and AMBTC compression code. The proposed mechanism adopts adjustable thresholds to enhance flexibility. Additionally, it includes two variants of data hiding and extraction, catering to various application scenarios: one with indicators and one without. Section III-A introduces the flowchart of the proposed mechanism and briefly describes four phases: 1) block type classification; 2) data encryption; 3) data embedding; and 4) data extraction and image reconstruction. Sections III-B and III-C describe Variant-1 and Variant-2 of the data hiding and extraction method, respectively.

A. General Flowchart of Our Proposed Scheme

The flowchart of our proposed mechanism is depicted in Fig. 7. The sender initially compresses the original image using AMBTC, conducts block type classification based on a threshold, utilizes different methods to embed data according to diverse types, and eventually obtains the stego compressed file. The receiver conducts block type classification on the stego compressed file, employs the corresponding extracting methods to obtain secret data according to predetermined block types, and can subsequently reconstruct the compressed image. It is important to note that the data encryption key must be possessed by both the sender and the receiver, with the sender employing it to encrypt the secret message and the receiver using it to decrypt the secret message.

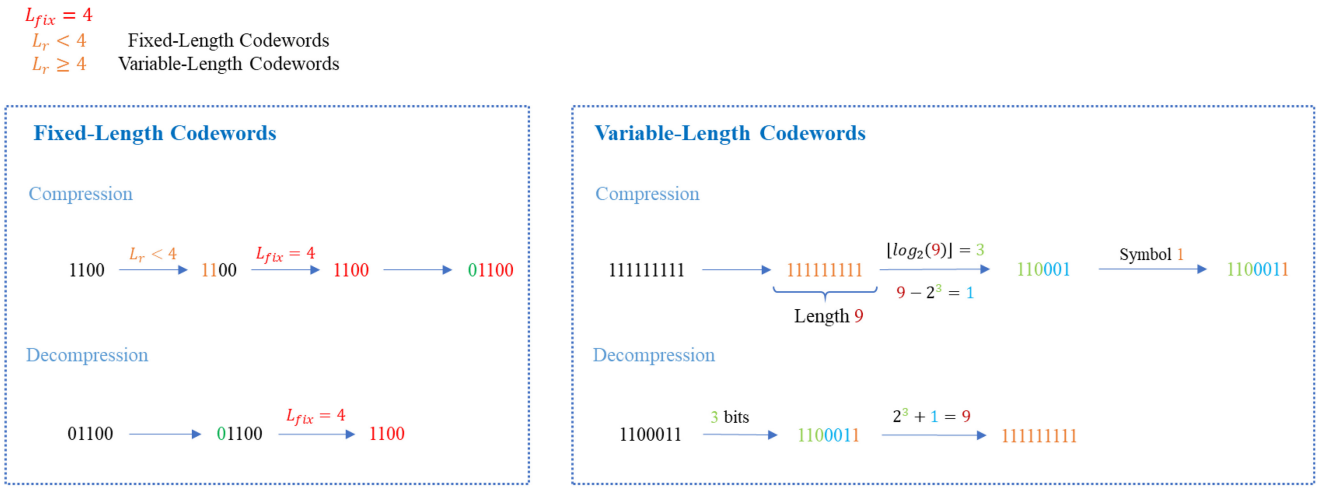


Fig. 6. Examples of ERLE.

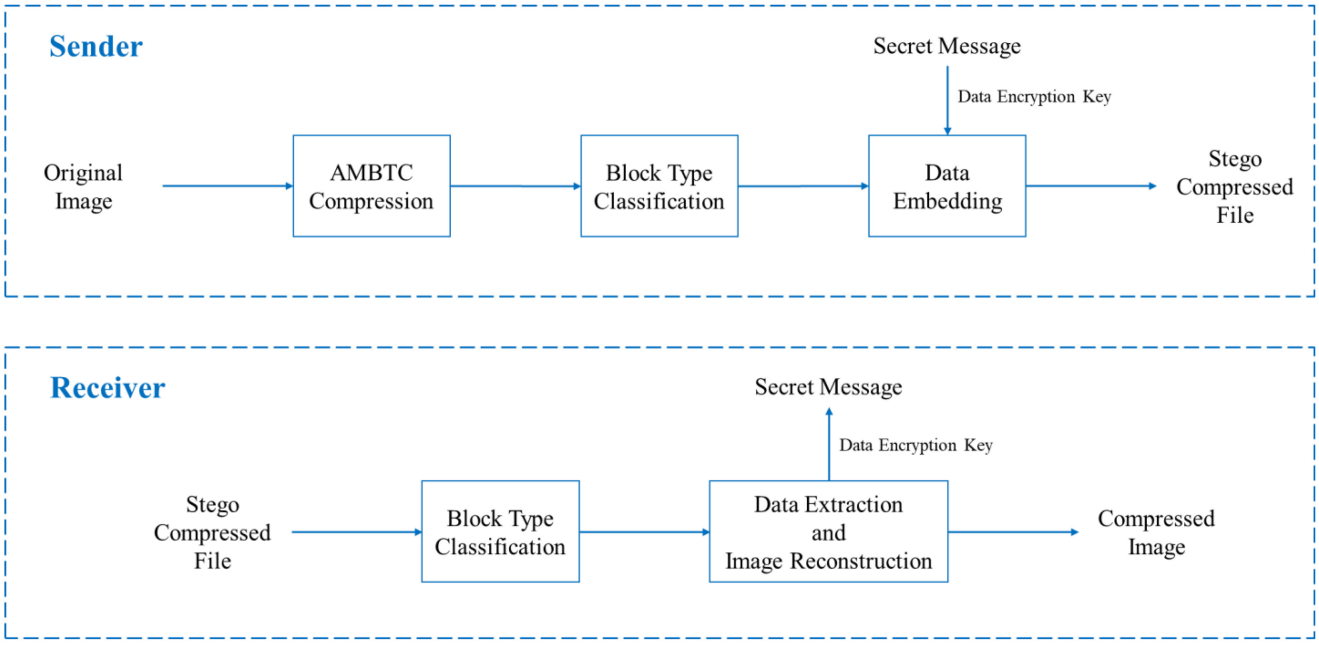


Fig. 7. Flowchart of our proposed mechanism.

1) Block Type Classification Phase: The original image is initially partitioned into 4 × 4 blocks, and AMBTC compression is then applied to extract high-value, low-value, and bitmap components. Following this, the difference value dv is calculated according to (10). An adjustable threshold, denoted as $T = (t1, t2)$, is employed in our proposed scheme for block type classification, as delineated in (11). If the threshold is appropriately set, *type* 1, *type* 2, and *type* 3 can, respectively, indicate whether the block is flat, smooth, or complex

$$dv = hv - lv \tag{10}$$

$$\text{Block type} = \begin{cases} \text{type 1, if } dv \leq t1 \\ \text{type 2, if } t1 < dv \leq t2 \\ \text{type 3, if } t2 < dv \end{cases} \tag{11}$$

2) Data Encryption Phase: Prior to embedding data, encryption using stream cipher encryption is required. As depicted in (12), the secret message s is a binary sequence, and a random sequence r of the same length is generated. The encrypted secret message s' is obtained by XORing the two. The random sequence r is designated as the data encryption key K_e

$$s' = s \oplus r. \tag{12}$$

3) Data Embedding Phase: Depending on the block type, different embedding methods are designed to achieve a relative balance between visual quality and embedding capacity.

4) Data Extraction and Image Reconstruction Phase: According to the data hiding methods, the corresponding extraction methods are adopted. Finally, the image can be reconstructed. Note that the secret data mentioned in the data extraction and reconstruction phase all refer to the encrypted secret message. Once all the encrypted secret messages are correctly extracted, they need to be decrypted by using the

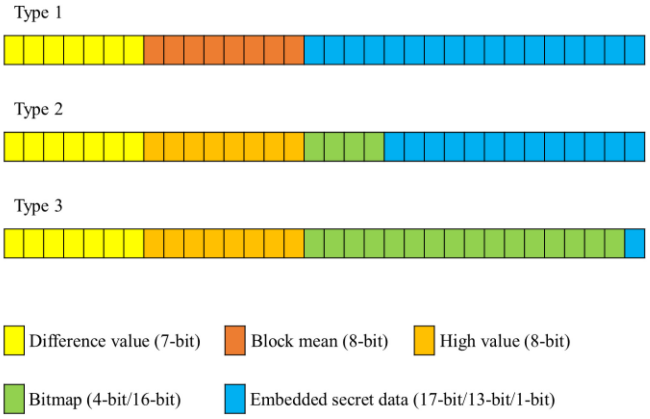


Fig. 8. Data structures for three block types in Variant-1.

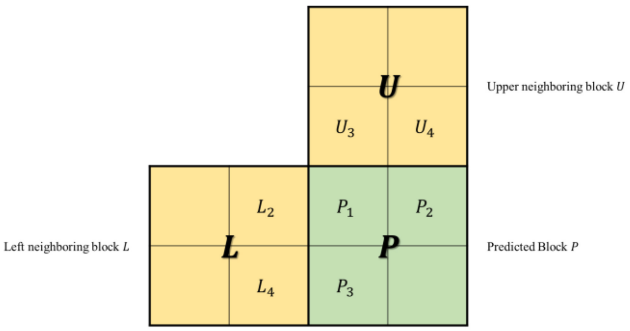


Fig. 9. Example of the side match method for type 2.

data encryption key to obtain the secret message. As shown in (13), the original secret message s can be obtained by XORing the encrypted secret message s' with the data encryption key r. As the data encryption key can be either stored in the patient's smartcard or preshared between the sender and receiver

$$s = s' \oplus r. \tag{13}$$

B. Variant-1: Data Hiding and Extraction Without Indicators

In this section, data hiding and extraction methods are presented for each block type without the use of indicators, and it is called Variant-1 for short in the following paragraphs. Variant-1 is particularly suitable for scenarios with limited bandwidth or storage, as it ensures that the file size remains constant. Essentially, the stego file size after data embedding aligns consistently with that of the AMBTC compressed file.

1) Data Hiding Methods for Three Block Types: Fig. 8 illustrates the data structure for each block type in Variant-1. Notably, the initial 7 bits of all three types store the difference value. Given the substantial intrablock correlation inherent in meaningful images, the difference between high and low values tends to be minimal, allowing for a concise 7-bit representation of the difference value. Block type classification can be achieved using (11).

As the difference value of *type* 1 is small, the block mean is directly stored for reconstruction, and the high and low values of AMBTC are disregarded, thus releasing the 17-bit embedding capacity. *type* 2 is a relatively smooth block, the original 16-bit bitmap is then shrunk into 4-bit, and the side

Fig. 10. All eight cases of the side match method.

match strategy is employed to predicate an approximately 16-bit bitmap. To achieve this objective, the upper block and the left block need to be decoded from the ABMTC compression codes. For the current block, its bitmap is divided into four subunits, each of which is a 2×2 block containing only 4 bits. As shown in Fig. 9, for the predicted bitmap of the 2×2 block P, P_4 is the reserved bitmap, so only the bitmap corresponding to the positions of P_1, P_2, and P_3 need to be predicted. Because P_1, P_2, and P_3 can each be either a high value or a low value, there are $2^3 = 8$ possible cases, as shown in Fig. 10. Calculate the values of P_1, P_2, and P_3 according to (14), and then compare these values with those of the eight cases to calculate the Euclidean distance. Finally, the case with the smallest distance is selected as the predicted bitmap. Therefore, *type* 2 only requires to record high value and a 4-bit bitmap, as each 2×2 block within the 4×4 block needs to record 1 bit, resulting in a total of 4 bits. Finally, 13 bits of embedded capacity can be freed up

$$\begin{cases} P_1 = \frac{L_2 + U_3}{2} \\ P_2 = U_4 \\ P_3 = L_4. \end{cases} \tag{14}$$

type 3 is a relatively complex block, requiring higher visual quality and minimal alteration to pixels. Therefore, to maintain the characteristics of AMBTC, it is still necessary to record high value and bitmap. By with the 7-bit difference value and recorded high value, the original high value and low value can be restored. Finally, it can free up 1-bit embedding capacity. It should be noted that we also use the turtle shell matrix, treating the high value and low value as pixel pairs, which can then embed 3-bit data. In the end, we store the stego high value and stego difference value, replacing the original ones. Therefore, the number of bits that can be embedded is $1 + 3 = 4$ in the *type* 3.

In Fig. 11, examples of the data embedding phase are demonstrated for Variant-1, employing a threshold $T = (16, 32)$. In Fig. 11(a), the block mean is calculated as 174. Following the AMBTC algorithm, a high value of 175 and a low value of 174 are obtained, resulting in a difference value of 1. Consequently, this block is classified as *type* 1, with the difference value and block mean recorded, while the remaining space is utilized for carrying secret data. Fig. 11(b) follows a similar process. The block mean, high value, low value, and difference value are computed. Given a difference value of 25, the block is classified as *type* 2. Here, only 4 bits of the bitmap need retention, with the remaining positions predicted using the side match algorithm. Hence, the difference value, high value, and bitmap are recorded, and the remaining space is utilized for embedding secret data. In Fig. 11(c), the calculations proceed in the same manner. With a difference value of 52, the block is classified as *type* 3. Subsequently,

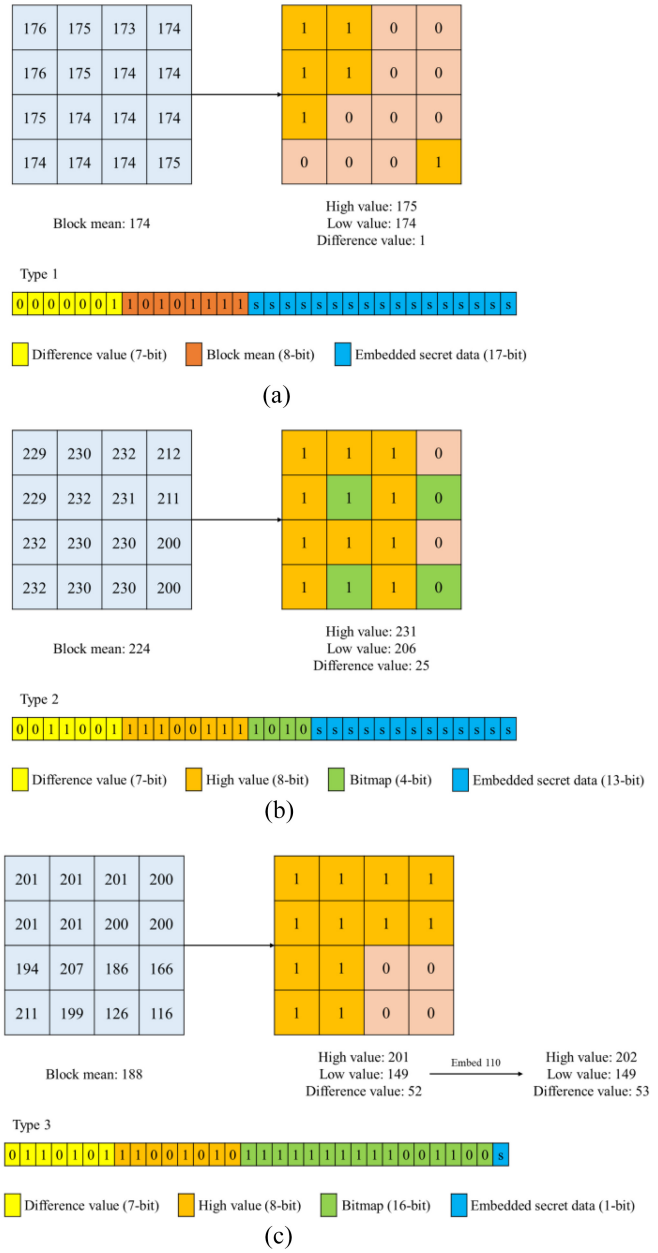


Fig. 11. Examples of data embedding phase for Variant-1. (a) Type 1. (b) Type 2. (c) Type 3.

the pair of high value and low value is mapped to the turtle shell matrix as coordinates (201, 149), concealed within secret data 110, resulting in stego coordinates of (202, 149). The new high value becomes 202, the low value is 149, and the difference value is 53. Finally, the difference value, high value, and bitmap are stored, with the additional space capable of embedding 1 bit of secret data.

2) Data Extraction and Image Reconstruction Methods for Three Block Types: When the receiver receives the stego compressed file, for each block, it first scans the first 7 bits, which represent the difference value for classifying the block by using (11). If it is *type* 1, the block mean is directly used as the value for all pixels in the block. If it is *type* 2, the bitmap is reconstructed through side match prediction, and the block is reconstructed using the AMBTC method. If it is *type* 3, the secret data is retrieved through the turtle shell matrix, and the image is reconstructed using the AMBTC method.

Fig. 12 shows examples of the data extraction and image reconstruction phase for Variant-1, with a threshold $T = (16, 32)$. The difference value employed in Fig. 12(a) is $(0000001)_2$, which equals $(1)_{10}$. Therefore, the block type is determined as *type* 1. Read the following 8 bits, the block mean $(10101111)_2$ is obtained, which is converted to $(174)_{10}$. The block mean is directly used as the value for all pixels to reconstruct the entire block. Read the following 17-bit to get the secret data. The difference value in Fig. 12(b) is $(0011001)_2$, which equals $(25)_{10}$ in decimal, classifying the block as *type* 2. Read the following 8 bits to get the high value $(11100111)_2$, which equals $(231)_{10}$. Further read the following 4 bits to acquire the 16-bit bitmap. Utilizing the side match algorithm, the 16-bit bitmap can be predicted and subsequently reconstructs the image block based on the predicated bitmap along with the high value and low value. Read the following 13-bit to retrieve the secret data. The difference value in Fig. 12(c) is $(0110101)_2$, which translates to $(53)_{10}$ in decimal, classifying it as *type* 3. Read the following 8 bits to obtain the high value $(11001010)_2$, equivalent to $(202)_{10}$. Further read the following 16 bits to form the bitmap. Reconstruct the image block based on the retrieved bitmap along with computed the high value and low value. Read the next 1-bit to get 1-bit secret data. Mapping the high value and low value as pairs to the turtle shell matrix, the value of the coordinates (202,149) is $(6)_8$, resulting in the remaining 3-bit secret data being $(110)_2$.

C. Variant-2: Data Hiding and Extraction With Indicators

We also designed Variant-2 with indicators, which is suitable for application scenarios that do not have strict requirements on compression size. Small auxiliary information is adopted to achieve a significant increase in embedding capacity. The difference between Variant-1 and Variant-2 is that additional indicators serve as auxiliary information in Variant-2, instead of recording 7-bit difference values as the basis for block type classification. Three prefix-free codes, wherein 0, 10, and 11 are used as indicators for *type* 1, *type* 2, and *type* 3, respectively. This is because meaningful image blocks tend to be relatively flat, resulting in a higher frequency of *type* 1 blocks, which are therefore represented by shorter codes. We counted the frequency of each type in the experimental section. The indicators of all blocks will be synthesized into a binary sequence. To minimize the size of the auxiliary information, the ERLE algorithm is employed to compress the auxiliary information sequence. The ERLE compressed sequence is then appended at the end of the AMBTC compressed bit sequence.

1) Data Hiding Methods for Three Block Types: Fig. 13 shows different types of data structures. *type* 1 only needs to store the 8-bit block mean, and the remaining 24 bits are used to conceal the secret data. *type* 2 simply stores the 8-bit high value and the $\lceil \log_2(t2) \rceil$-bit difference value. 4-bit bitmap is also required to be used to predict the 16-bit reconstructed bitmap by using side match method as mentioned in *type* 2 of

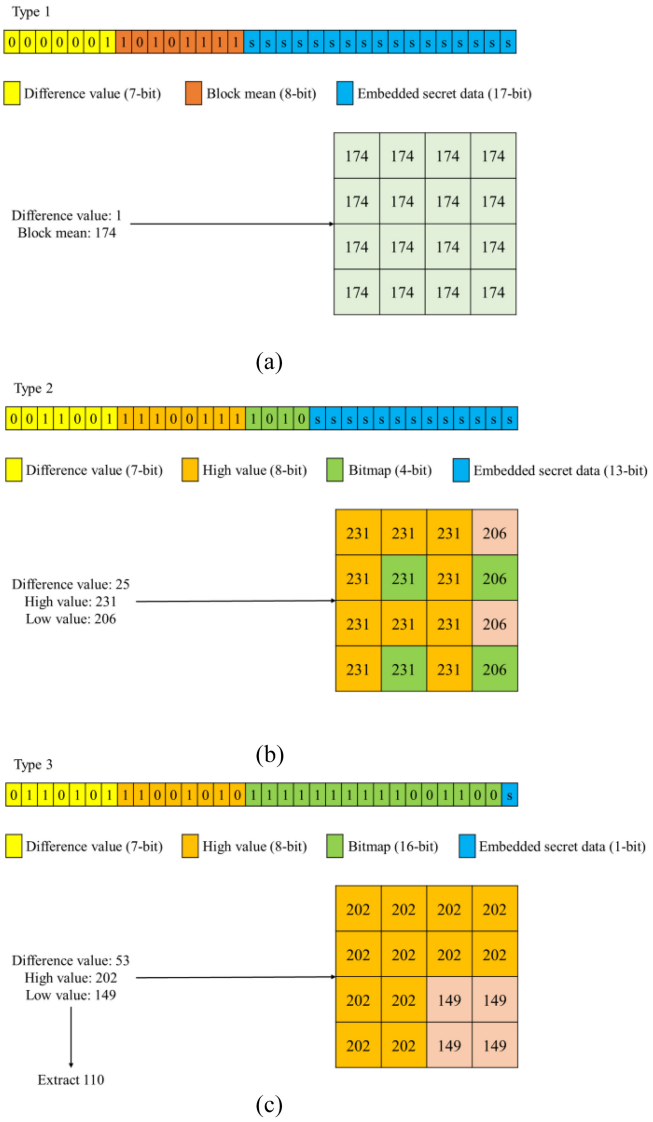


Fig. 12. Examples of data extraction and image reconstruction phase for Variant-1. (a) Type 1. (b) Type 2. (c) Type 3.

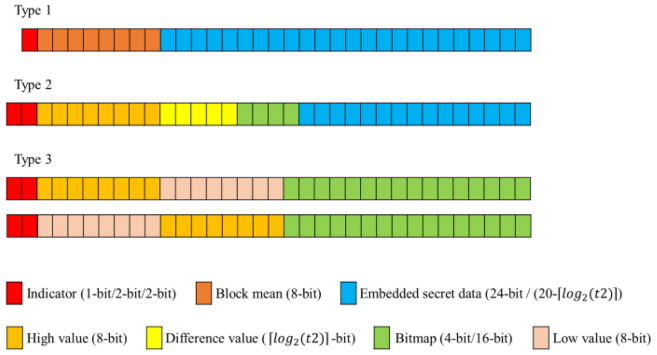


Fig. 13. Data structures for three block types in Variant-2.

Variant-1. The remaining space is used to carry secret data. *type* 3 is also similar to Section III-B. The high value and low value are mapped to the turtle shell matrix as a pair. The 3-bit secret data can be embedded, and the stego coordinates are obtained, which are the updated high value and low value. In other words, in *type* 3, the first 16 bits record the high value and low value of the AMBTC compression code. An additional 1-bit secret data can be embedded here. If the secret bit is 0, the high value is recorded before the low value, and vice versa. The last 16-bit records the bitmap, finally, the embedding capacity of *type* 3 is $3 + 1 = 4$ bits.

Fig. 14 illustrates examples of the data embedding phase for Variant-2, employing a threshold of $T = (16, 32)$. In Fig. 14(a), the block average is initially computed as 174. Following the AMBTC algorithm, the high value is determined as 175, and the low value as 174. Consequently, a difference value of 1 is calculated, categorizing the block as *type* 1, with 0 as the indicator. The block mean is recorded, and the remaining space is allocated for storing the secret data. Fig. 14(b) replicates the same process as before, computing the block average, high value, low value, and difference value. With a difference of 25, the block is classified as *type* 2, with 10 as the indicator. Only 4 bits of the bitmap need to be preserved, while the edge matching algorithm predicts the remaining positions. Consequently, the difference, high value, and bitmap are stored, and the secret data is embedded in the remaining space. Fig. 14(c) follows a similar procedure. With a difference of 52, the block is classified as *type* 3, with 11 as the indicator. Pairs of high value and low value are mapped to the turtle shell matrix as coordinates (201, 149), concealed within the secret data 110, resulting in stego coordinates (202, 149). As a result, the new high value becomes 202, the low value is 149, and the difference is 53. Finally, the high value, low value, and bitmap are stored, and additional secret data 1 can be embedded by swapping the positions of the high value and low value.

2) Data Extraction and Image Reconstruction Methods for Three Block Types: In Variant-2, when the receiver receives a stego compressed file, the indicator is read to determine the corresponding block type according to (11). For *type* 1 blocks, the block mean is directly used as the value for all pixels to reconstruct the block. For *type* 2 blocks, the bitmap is reconstructed through prediction, followed by image reconstruction using the AMBTC method. For *type* 3 blocks, the secret data is retrieved through the positions of the high value and low value, along with the turtle shell matrix, before image reconstruction using the AMBTC method.

Fig. 15 illustrates an example of the data extraction and image reconstruction stages for Variant-2 with a threshold $T = (16, 32)$. In Fig. 15(a), when the indicator is 0, the block is classified as *type* 1. Read the following 8 bits to obtain the block average value of $(10101111)_2$, converting to $(174)_{10}$. The block mean is directly applied as the value for all pixels in the entire block for its reconstruction. Continue to read the following 24-bit for the hidden secret data. When the indicator is 10 as shown in Fig. 15(b), belonging to *type* 2. Read the following bits to obtain the high-bit value of $(11100111)_2$, equal to $(231)_{10}$. Because $T = (16, 32)$, i.e., $t_2 = 32$, the following $\log_2(32) = 5$ bits are read to obtain the difference value. Read the following 4 bits to extract the 4-bit bitmap. Using the side matching algorithm, a 16-bit reconstructed bitmap can be predicated. Finally, a reconstructed image block can be derived with the predicated bitmap along with high

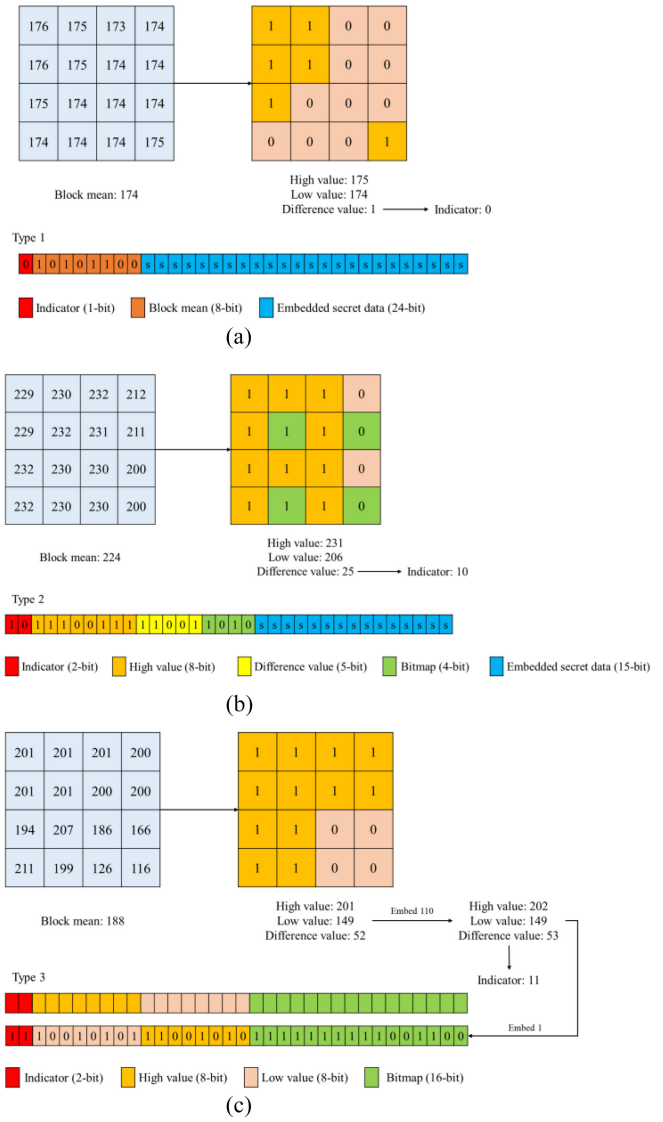


Fig. 14. Examples of data embedding phase for Variant-2. (a) Type 1. (b) Type 2. (c) Type 3.

value and low value. Read the following 15 bits of secret data. When the indicator shown in Fig. 15(c) is 11, indicating the current block is *type* 3. Read the following 16 bits to obtain $(10010101)_2$ and $(11001010)_2$. As $(11001010)_2$ is larger than $(10010101)_2$, it is concluded that the high value is transmitted before low value, and the embedded secret bit is determined as 1. Map the high value and low value to the turtle shell matrix as the pixel pair, the value of coordinate (202, 149) is $(6)_8$, then the remaining 3 bits of secret data are $(110)_2$. Finally, the following 16 bits are read to form the bitmap and reconstruct the image block from a bitmap with high value and low value.

IV. Experimental Results

In this section, we evaluate the performance of our proposed scheme. As shown in Fig. 16. Six 512 × 512 standard grayscale images, and six 400 × 400 medical images were selected from the public data set [30] to serve as test images. The experimental environment involved a Windows 11 computer equipped with an AMD Ryzen 7 5800H CPU with

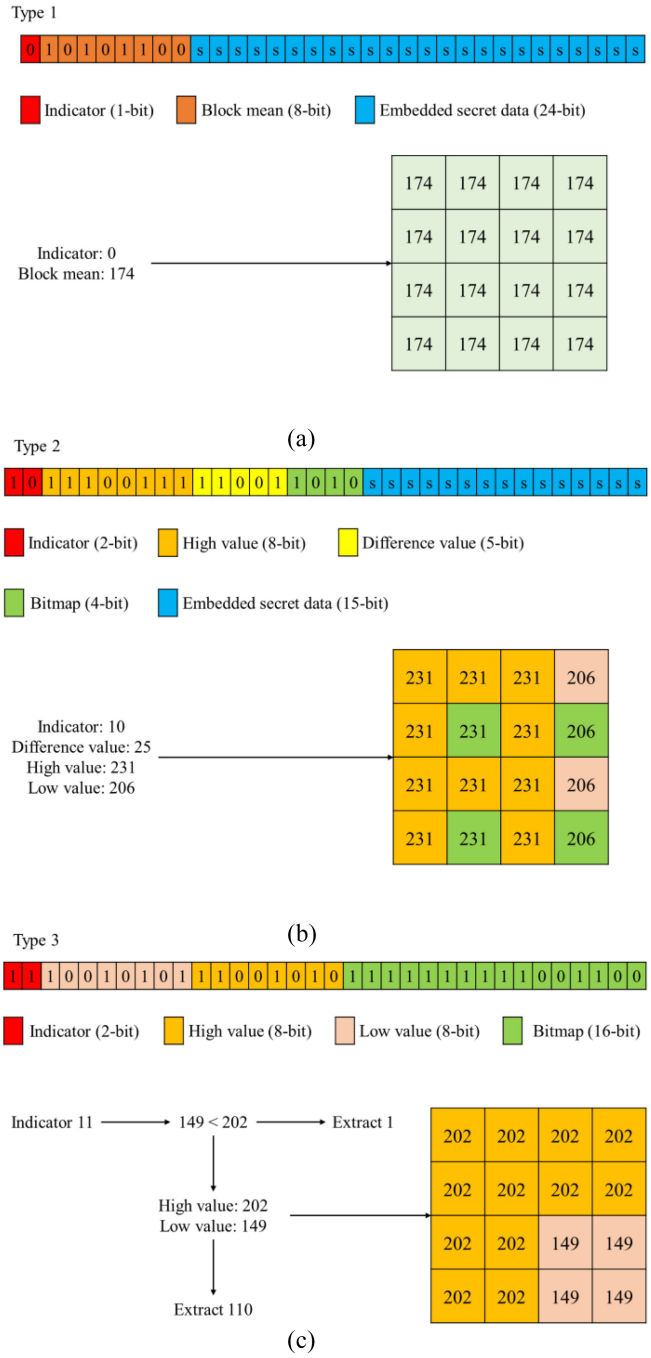


Fig. 15. Examples of data extraction and image reconstruction phase for Variant-2. (a) Type 1. (b) Type 2. (c) Type 3.

Radeon Graphics, 16 GB of memory, and MATLAB R2023b was used for experimentation.

Equations (15)–(20) present the metrics employed for assessing the performance of our two data hiding and extraction methods: Variant-1 and Variant-2, each defined and elucidated individually. Peak signal-to-noise ratio (PSNR) serves as a metric for gauging visual quality between two images, as depicted in (15), which is linked to the mean-square error (MSE). Equation (16) demonstrates the calculation method of MSE, where X and Y represent two images of size $m \times n$, and i and j represent pixel positions. A smaller MSE implies a larger PSNR, indicating higher visual quality.

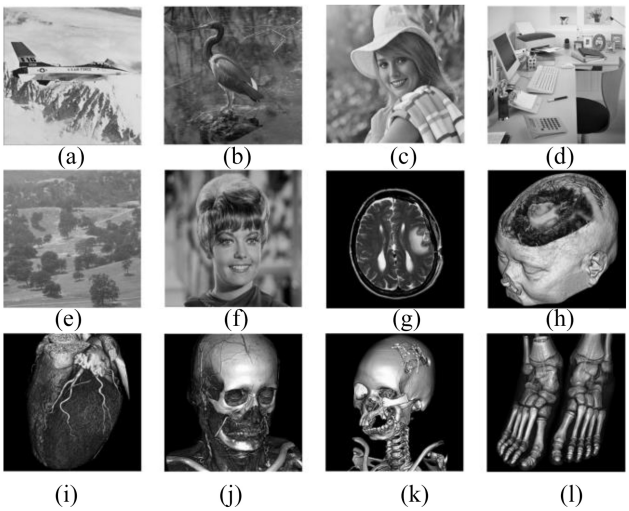


Fig. 16. Test images: (a) *Airplane*, (b) *Egretta*, (c) *Elaine*, (d) *Office*, (e) *Woodland*, (f) *Zelda*, (g) *Brainix*, (h) *Cerebrix*, (i) *Goudurix*, (j) *Manix*, (k) *Phenix*, and (l) *Vix*.

Structural similarity (SSIM), depicted in (17), is a metric used to quantify the similarity between two images. Given two images x and y, μ_x and μ_y are the average values of x and y, respectively, σ_x^2 and σ_y^2 are the variances of x and y, respectively, and σ_{xy} is the covariance of x and y. C_1 and C_2 are constants. A higher SSIM value indicates greater similarity between the two images, with a value of 1 denoting identical images. Multiscale SSIM index measure (MS-SSIM) evaluates image details across various resolutions and scales. In contrast to SSIM, MS-SSIM's assessment results may align more closely with subjective quality evaluations of an image. Equation (18) demonstrates the calculation of contrast measurement, denoted as $c(x, y)$, and structure measurement, $c(x, y)$, at M scales for two images, x and y.

The lightness measurement, denoted as $l(x, y)$, is computed solely at the final scale, with weights α_M, β_j, and γ_j employed to adjust the relative significance of different measurements. Similar to SSIM, a higher MS-SSIM value indicates greater similarity between the two images. Mean absolute error (MAE) also serves as a metric for assessing visual quality, as expressed in (19), where X and Y represent two images of size $m \times n$, and i and j denote the pixel positions. A smaller MAE value suggests a closer resemblance between the two images. Efficiency, depicted in (20), represents embedding performance, calculated as the ratio of payload to file size. Enhanced embedding performance is indicated by a higher efficiency value

$$\mathrm{PSNR} = 10 \times \log_{10} \frac{255^2}{\mathrm{MSE}} \tag{15}$$

$$\mathrm{MSE} = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} \left[X(i,j) - Y(i,j)\right]^2 \tag{16}$$

$$\mathrm{SSIM}(x, y) = \frac{\left(2\mu_x\mu_y + C_1\right)\left(2\sigma_{xy} + C_2\right)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)} \tag{17}$$

$$\mathrm{MS-SSIM}(x, y) = \left[l(x, y)\right]^{\alpha_M} \times \prod_{j=1}^{M} \left[c(x, y)\right]^{\beta_j} \times \left[s(x, y)\right]^{\gamma_j} \tag{18}$$

$$\mathrm{MAE} = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} |X(i,j) - Y(i,j)|. \tag{19}$$

$$\mathrm{Efficiency} = \frac{\mathrm{Payload}}{\mathrm{File\ Size}}. \tag{20}$$

Figs. 17 and 18 depict comparisons at various thresholds for Variant-1 and Variant-2, respectively. Each image comprises five points corresponding to different thresholds: $T = (1, 2)$, $T = (2, 4)$, $T = (4, 8)$, $T = (8, 16)$, and $T = (16, 32)$. Both Figs. 17 and 18 demonstrate a consistent trend: as the payload increases, the PSNR and SSIM decrease. This illustrates that our proposed mechanism can effectively choose the most suitable threshold to balance visual quality and embedding capacity based on varying application scenarios. Additionally, we introduce two data hiding and extraction methods, Variant-1 and Variant-2, about the flexibility of altering file size. The advantage of Variant-1 lies in its preservation of the original AMBTC compressed file size, rendering it suitable for scenarios with strict compression requirements. Conversely, Variant-2 is more suitable for scenarios where a slight increase in compression size is acceptable in exchange for a higher payload capacity.

Tables I and II provide comparisons across various variants and thresholds. Table I focuses on six standard grayscale images, while Table II examines six medical images. The comparison involves two variants, Variant-1 and Variant-2, and two thresholds, $T = (4, 8)$ and $T = (16, 32)$, evaluated across four scenarios. As previously discussed, Variant-1 maintains a fixed file size after data hiding but supports a smaller payload. In contrast, Variant-2 slightly increases the file size but offers a larger embedding capacity, thus achieving higher efficiency. Specifically, Variant-2 includes indicators to record block types, which results in a modest file size increase but allows for significantly more embedding space, enhancing overall efficiency. When comparing thresholds, $T = (4, 8)$ yields better visual quality, as indicated by PSNR and MAE, but offers a smaller payload. On the other hand, $T = (16, 32)$ provides a larger embedding capacity, though it slightly reduces visual quality. This is because the threshold determines block type intervals, with $T = (16, 32)$ resulting in more *type 1* and *type 2* blocks compared to $T = (4, 8)$. This increases embedding capacity but decreases the proportion of *type 3* blocks, which affects visual quality. For Variant-2 at $T = (16, 32)$, the payload for the test image *Zelda* reaches 364 833 bits, with an efficiency of 68.12%. Although the payload (328 162 bits) for the test image *Office* is relatively lower, the efficiency still stands at 60.79%, indicating that Variant-2 is suitable for applications requiring larger payloads. The same trend is observed with medical images; for Variant-2 at $T = (16, 32)$, the test medical image *Brainix* achieves a payload of 205 399 bits with an efficiency of 62.67%. This demonstrates that our scheme performs well with medical images as well.

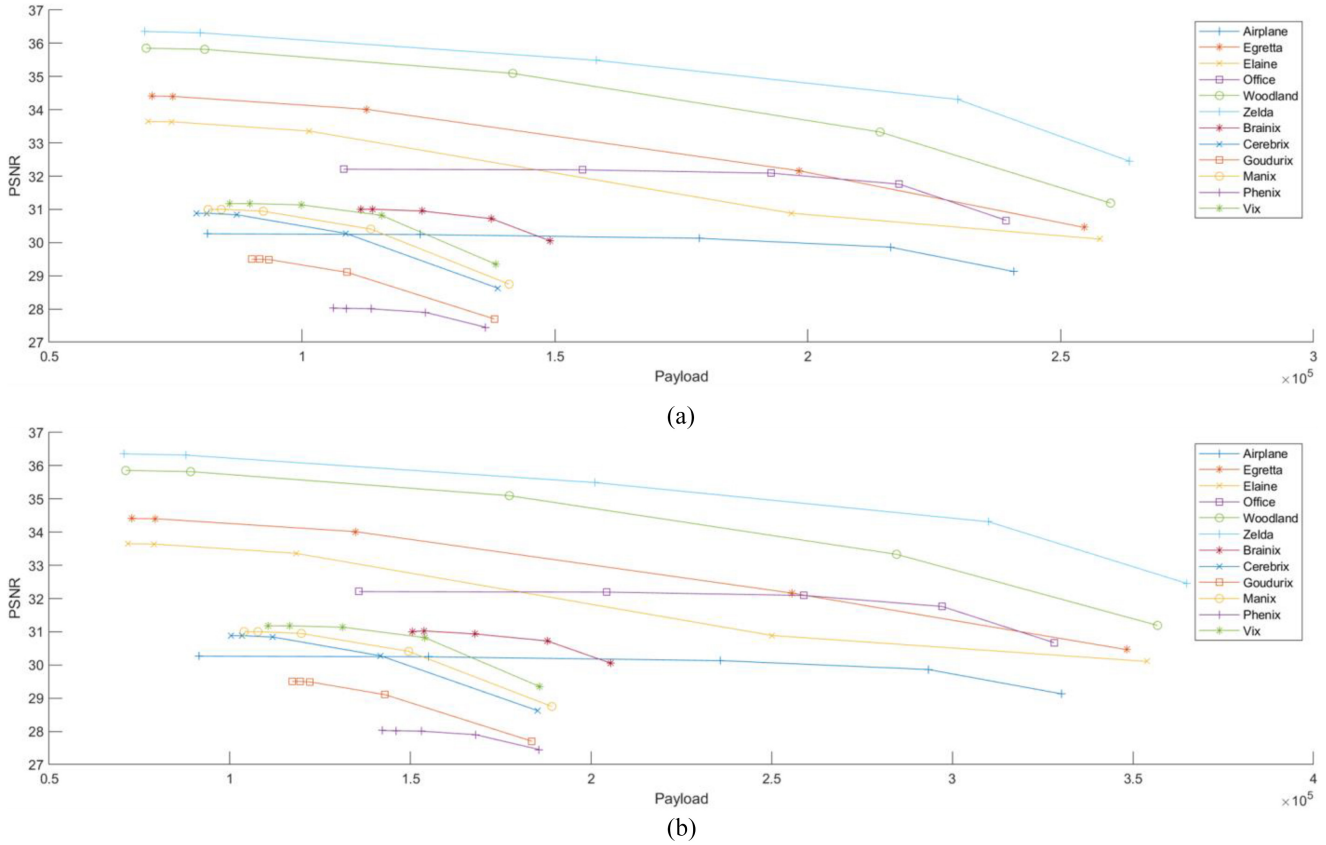
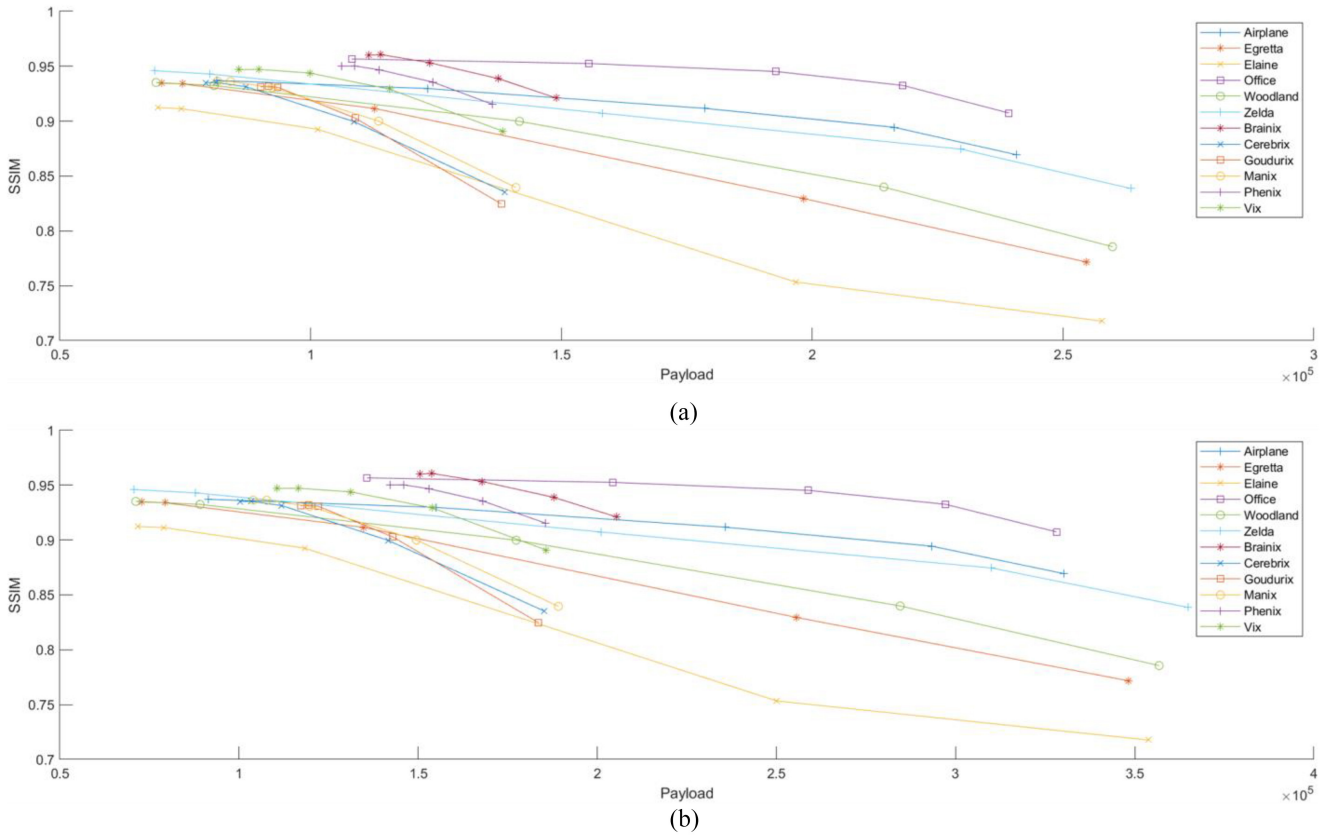


Fig. 17. Comparison of PSNR and payload at different thresholds: $T = (1, 2)$, $T = (2, 4)$, $T = (4, 8)$, $T = (8, 16)$, and $T = (16, 32)$. (a) Variant-1. (b) Variant-2.

Moreover, PSNR, SSIM, MS-SSIM, and MAE values for Variant-1 and Variant-2 are identical at the same threshold because the same method is used for image construction. The primary difference is in the storage method, which results in varying file sizes between the variants. Medical images, often having a monochrome background, tend to have smaller differences in background blocks, making them more likely to be classified as *type 1*. This allows for more effective compression and higher efficiency, especially with smaller thresholds. Conversely, general images with more texture and color information improve side match prediction, resulting in higher PSNR values compared to medical images. In summary, for the same threshold, both Variant-1 and Variant-2 produce identical visual quality metrics. The key difference is that Variant-1 keeps the file size unchanged, while Variant-2 increases the file size slightly but significantly enhances payload and efficiency. Different thresholds impact the distribution of block types and the resulting performance. A smaller threshold [e.g., $T = (4, 8)$] typically results in a higher number of *type 3* blocks, which improves visual quality but reduces the payload. Conversely, a larger threshold [e.g., $T = (16, 32)$] increases the number of *type 1* blocks, allowing for a greater payload but at the expense of visual quality. This flexibility enables the selection of appropriate variants and thresholds tailored to specific application needs, highlighting the adaptability of our proposed scheme.

Table III compares our proposed scheme with other AMBTC compression schemes [23], [24], [25], [26], focusing on three key metrics: 1) PSNR; 2) payload; and 3) efficiency. All schemes, including ours, classify blocks based on pixel value variations within each block type. Our scheme uses Variant-2 with $T = (16, 32)$. Although the PSNR of our scheme shows only slight differences compared to others, we have achieved significant improvements in both payload and efficiency. Notably, our scheme's efficiency surpasses that of others by over 20%. For example, in the standard grayscale test image *Airplane*, our scheme achieves a payload of 330 200 bits and an efficiency of 61.42%. Similarly, in the medical image *Brainix*, the payload reaches 205 399 bits, with an efficiency of 62.67%. These results demonstrate that our scheme excels in embedding capacity and efficiency while maintaining high visual quality. This advantage is due to our method of classifying blocks based on the difference between the original pixel values and the block average, using thresholds to determine block types. For type 1 blocks, with small differences, we embed a larger capacity; for *type 2* blocks, with moderate differences, we employ the side match algorithm to predict the bitmap and embed a moderate capacity; and for *type 3* blocks, with large differences, we use the turtle shell matrix to embed a smaller capacity, thus preserving visual quality.

Fig. 19 displays the reconstructed image of our proposed mechanism with a variant featuring indicators (Variant-2) and



(a)

(b)

Fig. 18. Comparison of SSIM and payload at different thresholds: $T = (1, 2)$, $T = (2, 4)$, $T = (4, 8)$, $T = (8, 16)$, and $T = (16, 32)$. (a) Variant-1. (b) Variant-2.

TABLE I
COMPARISON OF SIX STANDARD GRAYSCALE IMAGES AT DIFFERENT VARIANTS AND THRESHOLDS

Images	Airplane	Egretta	Elaine	Office	Woodland	Zelda	Airplane	Egretta	Elaine	Office	Woodland	Zelda
Metric	Variant-1 with $T = (4,8)$						Variant-1 with $T = (16,32)$					
PSNR (dB)	30.13	34.01	33.36	32.09	35.10	35.49	29.13	30.46	30.10	30.67	31.19	32.46
SSIM	0.912	0.912	0.893	0.945	0.900	0.907	0.869	0.772	0.718	0.907	0.786	0.839
MS-SSIM	0.986	0.988	0.987	0.993	0.985	0.988	0.978	0.956	0.961	0.985	0.956	0.975
MAE	1.79	1.28	1.50	1.39	1.31	1.30	2.29	2.61	2.81	1.83	2.38	2.04
File Size (bits)	524,288	524,288	524,288	524,288	524,288	524,288	524,288	524,288	524,288	524,288	524,288	524,288
Payload (bits)	178,601	112,781	101,369	192,675	141,634	158,170	240,720	254,604	257,804	239,214	259,851	263,496
Efficiency (%)	34.07	21.51	19.33	36.75	27.01	30.17	45.91	48.56	49.17	45.63	49.56	50.26
Metric	Variant-2 with $T = (4,8)$						Variant-2 with $T = (16,32)$					
PSNR (dB)	30.13	34.01	33.36	32.09	35.10	35.49	29.13	30.46	30.10	30.67	31.19	32.46
SSIM	0.912	0.912	0.893	0.945	0.900	0.907	0.869	0.772	0.718	0.907	0.786	0.839
MS-SSIM	0.986	0.988	0.987	0.993	0.985	0.988	0.978	0.956	0.961	0.985	0.956	0.975
MAE	1.79	1.28	1.50	1.39	1.31	1.30	2.29	2.61	2.81	1.83	2.38	2.04
File Size (bits)	546,074	549,414	542,473	543,153	553,666	555,859	537,570	541,340	540,967	539,835	539,813	535,579
Payload (bits)	235,809	134,781	118,297	258,937	177,362	201,152	330,200	348,341	353,839	328,162	356,789	364,833
Efficiency (%)	43.18	24.53	21.81	47.67	32.03	36.19	61.42	64.35	65.41	60.79	66.09	68.12

$T = (16, 32)$, exhibiting noticeable visual quality discernible to the naked eye. Notably, $T = (16, 32)$ entails a larger payload, leading to a slight decrease in visual quality, albeit remaining within perceptible limits. For elevated visual standards, adjusting the threshold to reduce the payload and achieve a higher PSNR is a viable option.

TABLE II
COMPARISON OF SIX MEDICAL IMAGES AT DIFFERENT VARIANTS AND THRESHOLDS

Images	Brainix	Cerebrix	Goudurix	Manix	Phenix	Vix	Brainix	Cerebrix	Goudurix	Manix	Phenix	Vix
Metric	Variant-1 with $T = (4,8)$						Variant-1 with $T = (16,32)$					
PSNR (dB)	30.95	30.84	29.49	30.95	28.00	31.14	30.05	28.63	27.70	28.75	27.45	29.35
SSIM	0.953	0.931	0.931	0.931	0.947	0.944	0.921	0.835	0.825	0.840	0.915	0.891
MS-SSIM	0.995	0.993	0.992	0.992	0.994	0.995	0.989	0.972	0.965	0.973	0.989	0.986
MAE	1.28	1.74	1.90	1.70	2.07	1.68	1.67	2.73	2.91	2.66	2.48	2.39
File Size (bits)	320,000	320,000	320,000	320,000	320,000	320,000	320,000	320,000	320,000	320,000	320,000	320,000
Payload (bits)	123,684	87,087	93,440	92,321	113,723	99,809	149,006	138,735	137,991	140,927	136,276	138,337
Efficiency (%)	38.65	27.21	29.20	28.85	35.54	31.19	46.56	43.35	43.12	44.04	42.59	43.23
Metric	Variant-2 with $T = (4,8)$						Variant-2 with $T = (16,32)$					
PSNR (dB)	30.95	30.84	29.49	30.95	28.00	31.14	30.05	28.63	27.70	28.75	27.45	29.35
SSIM	0.953	0.931	0.931	0.931	0.947	0.944	0.921	0.835	0.825	0.840	0.915	0.891
MS-SSIM	0.995	0.993	0.992	0.992	0.994	0.995	0.989	0.972	0.965	0.973	0.989	0.986
MAE	1.28	1.74	1.90	1.70	2.07	1.68	1.67	2.73	2.91	2.66	2.48	2.39
File Size (bits)	328,309	326,971	323,966	327,495	325,786	328,619	327,756	332,961	332,301	332,104	328,019	332,046
Payload (bits)	167,890	111,933	122,069	119,789	153,013	131,188	205,399	185,092	183,640	189,136	185,581	185,806
Efficiency (%)	51.14	34.23	37.68	36.58	46.97	39.92	62.67	55.59	55.26	56.95	56.58	55.96

TABLE III
COMPARISON WITH OTHER SCHEMES ON PSNR, PAYLOAD, AND EFFICIENCY WHEN USING VARIANT-2 WITH $T = (16, 32)$

Schemes	Images	Airplane	Brainix	Cerebrix	Goudurix	Manix	Phenix	Vix
Ou et al. [23]	PSNR (dB)	31.71	30.95	30.83	29.75	30.97	28.01	31.17
	Payload (bits)	174,754	117,580	74,950	75,400	82,000	101,635	88,390
	Efficiency (%)	33.33	36.74	23.42	23.56	25.63	31.76	27.62
Kumar et al. [24]	PSNR (dB)	31.33	30.68	29.98	29.02	30.19	27.82	30.48
	Payload (bits)	200,912	130,250	101,608	100,808	106,328	117,118	109,234
	Efficiency (%)	38.32	40.70	31.75	31.50	33.23	36.60	34.14
Chang et al. [25]	PSNR (dB)	29.94	29.59	28.73	27.84	29.02	26.92	29.40
	Payload (bits)	210,142	134,698	110,668	110,122	114,513	123,444	116,316
	Efficiency (%)	40.08	42.09	34.58	34.41	35.79	38.58	36.35
Chen et al. [26]	PSNR (dB)	31.85	31.05	30.96	29.73	31.06	28.04	31.42
	Payload (bits)	209,086	136,810	107,323	99,463	109,795	118,348	115,369
	Efficiency (%)	38.67	41.46	32.52	30.14	33.27	35.86	34.96
Ours	PSNR (dB)	29.13	30.05	28.63	27.70	28.75	27.45	29.35
	Payload (bits)	330,200	205,399	185,092	183,640	189,136	185,581	185,806
	Efficiency (%)	61.42	62.67	55.59	55.26	56.95	56.58	55.96

In the proposed Variant-2, we mentioned using shorter codes as indicators for types with higher frequencies. As shown in Fig. 20, for meaningful images, adjacent pixels are relatively smooth, resulting in a higher frequency of *type* 1, which is therefore represented by a shorter code.

V. CONCLUSION

This article presents an IoT-based medical protection mechanism that integrates data hiding techniques with AMBTC compression. By analyzing the difference values within image blocks, the blocks are categorized into three types, with an adjustable threshold to adapt to various application scenarios. Different embedding strategies are employed for each block type to balance embedding capacity and visual quality. Two data hiding and extraction methods, Variant-1 and Variant-2, are proposed, differing by file size: Variant-1 maintains file size consistency with AMBTC compressed files and does not use indicators, while Variant-2, which includes indicators, slightly increases the file size to offer a larger payload. Variant-1 is suitable for applications with limited bandwidth, whereas Variant-2 supports applications requiring

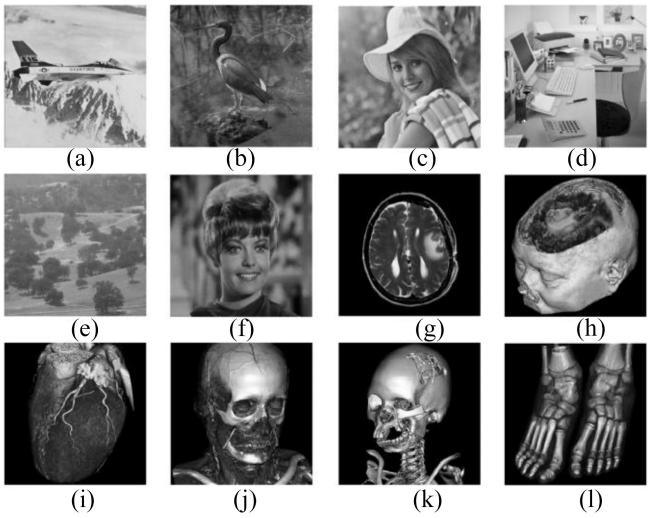


Fig. 19. Reconstructed images at a variant with indicators and $T = (16, 32)$: (a) *Airplane* (Payload = 330 200 bits, PSNR = 29.13 dB), (b) *Egretta* (Payload = 348 341 bits, PSNR = 30.46 dB), (c) *Elaine* (Payload = 353 839 bits, PSNR = 30.10 dB), (d) *Office* (Payload = 328 162 bits, PSNR = 30.67 dB), (e) *Woodland* (Payload = 356 789 bits, PSNR = 31.19 dB), (f) *Zelda* (Payload = 364 833 bits, PSNR = 32.46 dB), (g) *Brainix* (Payload = 205 399 bits, PSNR = 30.05 dB), (h) *Cerebrix* (Payload = 185 092 bits, PSNR = 28.63 dB), (i) *Goudurix* (Payload = 183 640 bits, PSNR = 27.70 dB), (j) *Manix* (Payload = 189 136 bits, PSNR = 28.75 dB), (k) *Phenix* (Payload = 185 581 bits, PSNR = 27.45 dB), and (l) *Vix* (Payload = 185 806 bits, PSNR = 29.35 dB).

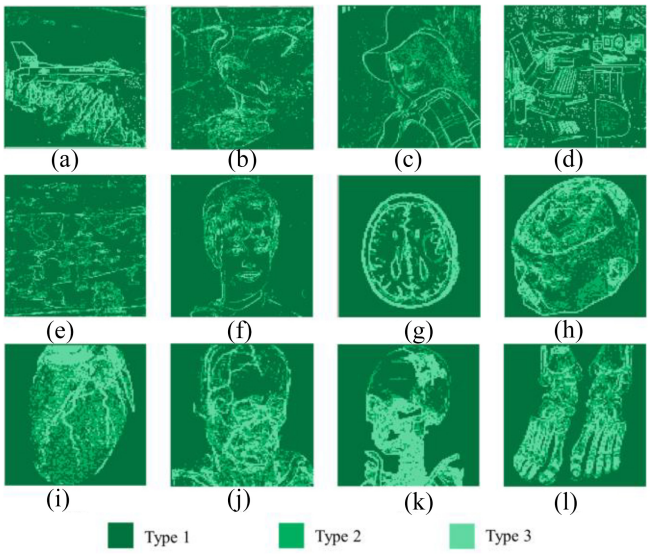


Fig. 20. Block type visualization at $T = (16, 32)$: (a) *Airplane*, (b) *Egretta*, (c) *Elaine*, (d) *Office*, (e) *Woodland*, (f) *Zelda*, (g) *Brainix*, (h) *Cerebrix*, (i) *Goudurix*, (j) *Manix*, (k) *Phenix*, and (l) *Vix*.

higher payloads. Experimental results demonstrate that the scheme maintains visually acceptable quality in both medical and standard grayscale images, validating its applicability across different image types. Overall, our scheme provides significant flexibility, allowing adjustments to threshold and variant to meet specific needs for embedding capacity, visual quality, and file size. Future research will focus on improving PSNR and payload for various scenarios, as well as ensuring data integrity.

REFERENCES

[1] M. D. Swanson, B. Zhu, and A. H. Tewfik, "Robust data hiding for images," in *Proc. IEEE Digital Signal Process. Workshop*, 1996, pp. 37–40.

[2] K.-H. Jung and K.-Y. Yoo, "Data hiding method using image interpolation," *Comput. Stand. Interfaces*, vol. 31, no. 2, pp. 465–470, 2009.

[3] Y. Lin, C.-C. Lin, J.-C. Liu, and C.-C. Chang, "Verifiable (t, n) secret image sharing scheme based on slim turtle shell matrix," *J. Inf. Secur. Appl.*, vol. 80, Feb. 2024, Art. no. 103679.

[4] Z. Fu, X. Chai, Z. Tang, X. He, Z. Gan, and G. Cao, "Adaptive embedding combining LBE and IBBE for high-capacity reversible data hiding in encrypted images," *Signal Process.*, vol. 216, Mar. 2024, Art. no. 109299.

[5] P. Ping, P. Wei, D. Fu, B. Guo, O. T. Bloh, and F. Xu, "IMIH: Imperceptible medical image hiding for secure healthcare," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 5, pp. 4652–4667, Sep./Oct. 2024.

[6] Z. Ni, Y.-Q. Shi, N. Ansari, and W. Su, "Reversible data hiding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 3, pp. 354–362, Mar. 2006.

[7] X. Zhang, "Reversible data hiding in encrypted image," *IEEE Signal Process. Lett.*, vol. 18, no. 4, pp. 255–258, Apr. 2011.

[8] X. Zhang, "Separable reversible data hiding in encrypted image," *IEEE Trans. Inf. Forensics Security*, vol. 7, pp. 826–832, 2012.

[9] W. He, L. Wang, Y. Wang, Z. Cai, and G. Xiong, "Reversible data hiding using morphology based pixel classification," *Expert Syst. Appl.*, vol. 250, Sep. 2024, Art. no. 123844.

[10] Y. Yang, H. He, F. Chen, Y. Yuan, and N. Mao, "Secure reversible data hiding in encrypted image based on 2D labeling and block classification coding," *J. Inf. Secur. Appl.*, vol. 83, Jun. 2024, Art. no. 103771.

[11] H. Shi, Z. Zhou, J. Qin, H. Sun, and Y. Ren, "A separable privacy-preserving technique based on reversible medical data hiding in plaintext encrypted images using neural network," *Multimedia Tools Appl.*, pp. 1–26, Mar. 2024.

[12] J.-C. Liu, C.-C. Chang, Y. Lin, C.-C. Chang, and J.-H. Horng, "A matrix coding-oriented reversible data hiding scheme using dual digital images," *Mathematics*, vol. 12, no. 1, p. 86, 2024.

[13] Y. Wang, G. Xiong, and W. He, "High-capacity reversible data hiding in encrypted images based on pixel-value-ordering and histogram shifting," *Expert Syst. Appl.*, vol. 211, Jan. 2023, Art. no. 118600.

[14] L. Xiong, X. Han, C.-N. Yang, and Z. Xia, "RDH-DES: Reversible data hiding over distributed encrypted-image servers based on secret sharing," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 19, no. 1, pp. 1–19, 2023.

[15] X. Ye, Y. Zhang, X. Xiao, S. Yi, and R. Lan, "Usability enhanced thumbnail-preserving encryption based on data hiding for JPEG images," *IEEE Signal Process. Lett.*, vol. 30, pp. 793–797, Jun. 2023.

[16] W. Tang, H. Yao, Y. Le, and C. Qin, "Reversible data hiding for JPEG images based on block difference model and Laplacian distribution estimation," *Signal Process.*, vol. 212, Nov. 2023, Art. no. 109130.

[17] X. Zhang, Z. Pan, Q. Zhou, G. Fan, and J. Dong, "A reversible data hiding method based on bitmap prediction for AMBTC compressed hyperspectral images," *J. Inf. Secur. Appl.*, vol. 81, Mar. 2024, Art. no. 103697.

[18] R. Gray, "Vector quantization," *IEEE ASSP Mag.*, vol. 1, no. 2, pp. 4–29, Apr. 1984.

[19] M. Lema and O. Mitchell, "Absolute moment block truncation coding and its application to color images," *IEEE Trans. Commun.*, vol. 32, no. 10, pp. 1148–1157, Oct. 1984.

[20] C.-C. Lin, X.-L. Liu, and S.-M. Yuan, "Reversible data hiding for VQ-compressed images based on search-order coding and state-codebook mapping," *Inf. Sci.*, vol. 293, pp. 314–326, Feb. 2015.

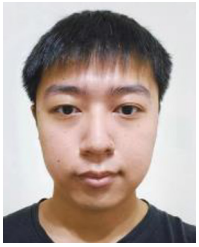
[21] C. Qin and Y.-C. Hu, "Reversible data hiding in VQ index table with lossless coding and adaptive switching mechanism," *Signal Process.*, vol. 129, pp. 48–55, Dec. 2016.

[22] Y. Lin, J.-C. Liu, C.-C. Chang, and C.-C. Chang, "Embedding secret data in a vector quantization codebook using a novel thresholding scheme," *Mathematics*, vol. 12, no. 9, p. 1332, 2024.

[23] D. Ou and W. Sun, "High payload image steganography with minimum distortion based on absolute moment block truncation coding," *Multimedia Tools Appl.*, vol. 74, pp. 9117–9139, Nov. 2015.

[24] R. Kumar, D.-S. Kim, and K.-H. Jung, "Enhanced AMBTC based data hiding method using hamming distance and pixel value differencing," *J. Inf. Secur. Appl.*, vol. 47, pp. 94–103, Aug. 2019.

[25] C.-C. Chang, X. Wang, and J.-H. Horng, "A hybrid data hiding method for strict AMBTC format images with high-fidelity," *Symmetry*, vol. 11, no. 10, p. 1314, 2019.
[26] Y.-Y. Chen, Y.-C. Hu, H.-Y. Kao, and Y.-H. Lin, "Security for eHealth system: Data hiding in AMBTC compressed images via gradient-based coding," *Complex Intell. Syst.*, vol. 9, no. 3, pp. 2699–2711, 2023.
[27] T. Kim, "Side match and overlap match vector quantizers for images," *IEEE Trans. Image Process.*, vol. 1, no. 2, pp. 170–185, Apr. 1992.
[28] C. C. Chang, Y. Liu, and T. S. Nguyen, "A novel turtle shell based scheme for data hiding," in *Proc. 10th Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, Aug. 2014, pp. 89–93.
[29] K. Chen and C.-C. Chang, "High-capacity reversible data hiding in encrypted images based on extended run-length coding and block-based MSB plane rearrangement," *J. Vis. Commun. Image Represent.*, vol. 58, pp. 334–344, Jan. 2019.
[30] "OsiriX database. "Accessed: Mar. 14, 2023. [Online]. Available: https://www.osirix-viewer.com/support/knowledge-base/



Yijie Lin received the B.S. degree in computer science and information engineering from National Pingtung University, Pingtung, Taiwan, in 2022. He is currently pursuing the Ph.D. degree with the Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan.

His current research interests include steganography, watermarking, information security, image processing, computer vision, and artificial intelligence.

Chia-Chen Lin (Member, IEEE) received the Ph.D. degree in information management from National Chiao Tung University, Hsinchu, Taiwan, in 1998.

She is currently a Professor with the Department of Computer Science and Information Engineering, National Chin-Yi University of Technology, Taichung, Taiwan. Her research interests include image and signal processing, steganography, watermarking, cybersecurity, mobile agent, electronic commerce, machine learning, and artificial intelligence.

Dr. Lin also serves an associate editor and an editor for several representative EI and SCIE journals. From 2009 to 2012, she served as the Vice Chairman of Tainan Chapter IEEE Signal Processing Society. Since 2018, she has been a Fellow of IET and the School Counselor for Providence University.

Ching-Chun Chang received the Ph.D. degree in computer science from the University of Warwick, Coventry, U.K., in 2019.

He engaged in a short-term scientific mission supported by European Cooperation in Science and Technology Actions at the Faculty of Computer Science, Otto von Guericke University Magdeburg, Magdeburg, Germany, in 2016. He was granted the Marie-Curie Fellowship and participated in a research and innovation staff exchange scheme supported by Marie Skłodowska-Curie Actions at the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA, in 2017. He was a Visiting Scholar with the School of Computer and Mathematics, Charles Sturt University, Bathurst, NSW, Australia, in 2018, and the School of Information Technology, Deakin University, Geelong, VIC, Australia, in 2019. He was a Research Fellow with the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2020. He has been a Postdoctoral Fellow with the National Institute of Informatics, Tokyo, Japan, since 2021. His research interests include steganography, watermarking, forensics, biometrics, cybersecurity, applied cryptography, image processing, computer vision, natural language processing, computational linguistics, machine learning, and artificial intelligence.

Chin-Chen Chang (Fellow, IEEE) received the Ph.D. degree in computer science from National Chiao-Tung University, Hsinchu, Taiwan, in 1982.

He has worked on many different topics in information security, cryptography, and multimedia image processing and published several hundreds of papers in international conferences and journals and over 30 books. He was cited over 48 553 times and has an H-factor of 100 according to Google Scholar. Several well-known concepts and algorithms were adopted in textbooks. He also worked with the National Science Council, Ministry of Technology, Ministry of Education, Ministry of Transportation, Ministry of Economic Affairs, and other government agencies on more than 100 projects and holds 34 patents. He had given over a 1000 invited talks at institutions, including the Chinese Academy of Sciences, Academia Sinica, Tokyo University, Kyoto University, National University of Singapore, Nanyang Technological University, The University of Hong Kong, National Taiwan University, and Peking University. He has mentored seven postdoctoral, 68 Ph.D. students, and 208 master students. He is a leader in the field of information security of Taiwan.

Prof. Chang is also the recipient of several awards, including the Top Citation Award from Pattern Recognition Letters, the Outstanding Scholar Award from Journal of Systems and Software, and the Ten Outstanding Young Men Award of Taiwan. He has been the Editor-in-Chief of *Information Education*, a magazine that aims at providing educational materials for middle-school teachers in computer science. Also, he has been an Associate Editor of *Information Sciences*, USA. He founded the Chinese Cryptography and Information Security Association, accelerating information security the application and development and consulting on the government policy. He was elected as a Fellow of IET in 2000, a Fellow of CS in 2020, an AAIA Fellow in 2021, and a member of the Academy of Europe in 2022 and the European Academy of Sciences and Arts in 2022.