Multimed Tools Appl (2018) 77:11727–11750
DOI 10.1007/s11042-017-4814-7

An SMVQ-based reversible data hiding technique exploiting side match distortion

Kris Manohar[1] · **The Duc Kieu**[1]

Received: 8 October 2016 / Revised: 27 February 2017 / Accepted: 3 May 2017 /
Published online: 20 May 2017
© Springer Science+Business Media New York 2017

Abstract Secure online communication is a necessity in today's digital world. This paper proposes a novel reversible data hiding technique based on side match vector quantization (SMVQ). The proposed scheme classifies SMVQ indices as Case 1 or 2 based on the value of the first state codeword's side match distortion (*SMD*) and a predefined threshold t. The proposed scheme uses this classification to switch between compression codes designed for Cases 1 and 2 SMVQ indices. The length of these compression codes is controlled by the parameter ℓ. Thus, with the selection of appropriate ℓ and t values, the proposed scheme achieves good compression, creating spaces to embed secret information. The embedding algorithm can embed n secret bits into each SMVQ index, where n = 1, 2, 3, or 4. The experimental results show that the proposed scheme obtains the embedding rates of 1, 2, 3, or 4 bit per index (bpi) at the average bit rates of 0.340, 0.403, 0.465, or 0.528 bit per pixel (bpp) for the codebook size 256. This improves the performance of recent VQ and SMVQ-based data hiding schemes.

Keywords Reversible data hiding · Steganography · Watermarking · Vector quantization · Side match vector quantization · Side match distortion

1 Introduction

The increasing use of digital communication raises concerns about information security and privacy. In response to this, researchers designed cryptographic algorithms such as DES [8] and RSA [21], which convert digital objects (i.e., texts, images, videos, and audios) into

✉ The Duc Kieu
ktduc0323@yahoo.com.au; duc.kieu@sta.uwi.edu

Kris Manohar
justkrismanohar@gmail.com

[1] Faculty of Science and Technology, Department of Computing and Information Technology, University of the West Indies, St. Augustine, Trinidad and Tobago

ciphertexts for secure transmission over public channels. Although ciphertexts may be secure, their unrecognizable form alerts malicious users to the transmission of secure information. To resolve this issue, information hiding (also called data hiding) embeds secret information within cover objects, making it difficult for malicious users to distinguish between original cover objects and stego objects (i.e., cover objects embedded with secret information). Because of the minimal storage space and bandwidth consumption of compressed images, researchers have proposed reversible information hiding schemes that use compressed images as cover objects. Reversible data hiding schemes ensure that both the cover object and the secret message can be successfully restored from a stego object. In particular, many researchers used vector quantization (VQ) [9] and side match vector quantization (SMVQ) [13] compressed images as cover objects to embed secret data [2–7, 12, 14–16, 18–20, 22–26].

In 2011, Yang et al. [26] used the differences between a VQ index and its four adjoining neighbors to regulate their embedding capacity. The frequencies of the two smallest differences determine the number of secret bits embedded into a VQ index. To reduce the size of the output code stream, Huffman codes [10] were used as indicators. While this achieves an average bit rate of 0.494 bits per pixel (bpp), their embedding rate is 1.298 bits per index (bpi) on average. In 2013, Wang et al. [22] used the values of a VQ index's neighbors to build two state codebooks. Because state codebooks are smaller than main codebooks, fewer bits are required to represent state codewords. Wang et al.'s adjoining state codebook mapping (ASCM) technique [22] obtains an average embedding rate of 2.441 bpi at an average bit rate of 0.496 bpp. Pan et al. [19] proposed an improvement to Chang et al.'s information hiding scheme [2] that was based on the search order coding (SOC) algorithm [11]. They extended Chang et al.'s compression codes [2] to include all SOC codes by adding an indicator bit. While this strategy gains an average bit rate of 0.457 bpp, their embedding capacity is fixed at 1 bpi. Lee et al. [14] increased the correlation among the VQ indices within the smooth regions of an image by sorting the main codebook in order of codewords' mean values. Their algorithm utilizes coarse and fine subcodebooks to compress these highly correlated VQ indices, creating spaces to embed secret bits. Although their scheme does not perform well with complex images, it achieves a good average embedding rate of 3.046 bpi at an average bit rate of 0.540 bpp, improving the works of Chang et al. [4], Wang and Lu [23], and Chang et al. [7]. Chang et al. [6] embedded secret bits into transformed index values (TIVs). The TIVs refer to the position of a VQ index's codeword within the state codebook generated by the SMVQ algorithm [13]. As a result, the TIVs tend to be smaller and more correlated than the VQ indices. Their scheme compresses the TIVs that are either less than eight or have a 2-bit search order code, achieving an average bit rate of 0.407 bpp. The limitations of their scheme are a fixed embedding rate of 1 bpi and ignoring SOC code 11.

In 2014, Wang et al. [24] modified the locally adaptive coding (LAC) algorithm [1] to increase the frequency of LAC stack positions in the interval [0, 7]. This created an SMVQ frequency distribution that is ideal for compression with variable bit encoding. In order to control the size of the output code stream, Wang et al.'s scheme [24] only embeds secret bits into the first four LAC stack positions. While this approach achieves an average embedding rate of 1.861 bpi at an average bit rate of 0.403 bpp, their embedding capacity decreases as image complexity and main codebook size increase.

In 2015, Lin et al. [16] combined SOC and state codebook mapping (SCM) to improve Wang et al.'s ASCM technique [22]. If a search order code does not exist for a VQ index, Lin et al.'s scheme [16] uses the SCM to build a state codebook for each search point in the SOC algorithm. This generates more state codebooks than ASCM, increasing the probability of

compressing a VQ index. Lin et al.'s approach [16] embeds secret bits if the compression code is less than ψ bits. While this generates good average embedding rates of 3.035 bpi and 3.879 bpi with $\psi = 8$ and 9, the respective average bit rates of 0.519 bpp and 0.572 bpp are worse than the traditional VQ compression.

In 2015, Chang et al. [5] proposed the use of SMVQ and search order codes to represent the secret bits 1 and 0, respectively. Their scheme uses three of the four 2-bit search order codes (00, 01, and 10) to compress SMVQ indices. The fourth search order code, 11, is used to inform the decoder that no search order code is found. Additionally, their embedding strategy ignores the shorter search order codes when the secret bit is 1. Despite a fixed embedding capacity of 1 bpi, Chang et al.'s scheme [5] achieves excellent average bit rates of 0.368 and 0.417 bpp with state codebooks of sizes 32 and 64.

In order to improve the embedding capacity and bit rate of SMVQ-based reversible information hiding schemes, this paper proposes to classify SMVQ indices as either Case 1 or 2 based on the value of the minimum side match distortion (*SMD*) and a threshold *t*. This classification increases the probability of Case 1 SMVQ indices belonging to the interval [0, *a*], and Case 2 SMVQ indices belonging to the interval $[a + 1, M - 1]$, where $a = 2^w - 1$, $w = 0$, 1, 2, 3, ..., $\lceil \log_2 M \rceil$, and *M* is the state codebook size. The proposed scheme controls the length of compression codes used for these subintervals through the value of a parameter ℓ. Therefore, when a Case 1 or 2 SMVQ index is detected, the proposed scheme can switch between compression codes optimized for the subintervals [0, *a*] and $[a + 1, M - 1]$. After an SMVQ index is compressed, *n* secret bits are appended to its compression code, where $n = 1$, 2, 3, or 4. Performance comparisons with the recent works of Chang et al. [5], Chang et al. [6], Lin et al. [16], and Wang et al. [24] confirm that our proposed scheme improves the embedding capacity and bit rate of recent reversible VQ and SMVQ-based data hiding schemes.

The remainder of this paper is organized as follows: Section 2 reviews related works. Section 3 describes the proposed scheme. Section 4 presents the experimental results and comparisons to the recent works [5, 6, 16, 24]. Section 5 summarizes the contributions of this paper.

2 The related works

2.1 Vector quantization

Vector quantization (VQ) is a lossy data compression technique proposed by Gray in 1984 [9]. VQ has three components, namely, codebook generator, VQ encoder, and VQ decoder. The codebook *D* containing *N* *k*-dimensional codewords CW_y's is generated by using the LBG clustering algorithm [17], where $CW_y = (cw_{y,0}, cw_{y,1}, ..., cw_{y,k-1})$. A grayscale image *G* sized $H \times W$ to be compressed is then divided into nonoverlapping pixel blocks *X*s, each of size $p \times q$, where $p \times q = k$. These pixel blocks are processed in raster scan order (i.e., left to right and top to bottom). The VQ encoder compresses each pixel block *X* by searching the codebook *D* and selecting the codeword CW_r with the smallest Euclidean distance from *X*. The Euclidean distance between *X* and CW_y is defined as

$$ED(X, CW_y) = \sqrt{\sum_{i=0}^{p-1} \sum_{j=0}^{q-1} \left(X_{i,j} - cw_{y,p\times i+j}\right)^2},$$

where $X_{i,j}$ is the jth component in the ith row of the pixel block X, CW_y is the yth codeword in D, and $cw_{y,z}$ ($z = p \times i + j$) is the zth component of CW_y. The index value r ($0 \le r \le N - 1$) of the chosen codeword CW_r (i.e., the best-matched or optimal codeword) is then added into the VQ index table *VIT* sized $(H/p) \times (W/q)$ in raster scan order. Each VQ index in *VIT* is represented by using $\lceil \log_2 N \rceil$ bits. The VQ decoder reconstructs the image G from the received VQ index table *VIT* and the codebook D. Each pixel block X of the image G is reconstructed by replacing each VQ index in *VIT* with the corresponding k-dimensional codeword in D.

2.2 Side match vector quantization

Side match vector quantization (SMVQ) was designed by Kim in 1992 [13] to improve the bit rate of vector quantization [9]. Kim proposed the use of a smaller state codebook (also called subcodebook) Y of size M instead of the codebook D sized N used in VQ (D is also known as super or main codebook). Since $M < N$, the number of bits used for an SMVQ index (i.e., $\lceil \log_2 M \rceil$ bits) is less than that used for a VQ index (i.e., $\lceil \log_2 N \rceil$ bits). SMVQ divides a grayscale image G sized $H \times W$ into nonoverlapping pixel blocks Xs, each of size $p \times q$. These pixel blocks are processed in raster scan order. Pixel blocks in the first row and column are encoded using the VQ encoder with the main codebook, and residual blocks (i.e., blocks are not in the first row or column) are encoded using the SMVQ encoder with the state codebook.

Given the upper and left adjacent pixel blocks, denoted as U and L, of the current pixel block X, as shown in Fig. 1, the boundary pixel vector (BV) of the current pixel block X (also called the state vector X') is defined as $BV(X) = (X_{0,0}, X_{0,1}, ..., X_{0,q-1}, X_{1,0}, X_{2,0}, ..., X_{p-1,0})$, where $X_{0,0} = (U_{p-1,0} + L_{0,q-1}) / 2$, $X_{0,1} = U_{p-1,1}$, ..., $X_{0,q-1} = U_{p-1,q-1}$, $X_{1,0} = L_{1,q-1}$, ..., $X_{p-1,0} = L_{p-1,q-1}$. The *SMD* between the $BV(X)$ and a codeword $CW_y = (cw_{y,0}, cw_{y,1}, ..., cw_{y,q-1}, ..., cw_{y,p\times q-1})$ in the main codebook D is defined as

$$SMD\left(BV(X), CW_y\right) = \sqrt{\sum_{j=0}^{q-1}\left(X_{0,j} - cw_{y,j}\right)^2} + \sqrt{\sum_{i=1}^{p-1}\left(X_{i,0} - cw_{y,i\times q}\right)^2}.$$

The codewords of the main codebook D are sorted in ascending order based on the *SMD*s with the $BV(X)$. The first M codewords are then selected to be the state codewords SCW_y's for the state codebook Y, where $0 \le y \le M - 1$. Similar to VQ, the input pixel block X is replaced with the $\lceil \log_2 M \rceil$-bit representation of an SMVQ index x in the state codebook Y. The SMVQ index x refers to the state codeword SCW_x with the smallest Euclidean distance to X—that is,

				$U_{0,0}$	$U_{0,1}$	...	$U_{0,q-1}$
				$U_{1,0}$	$U_{1,1}$	...	$U_{1,q-1}$
				...	U		
				$U_{p-1,0}$	$U_{p-1,1}$	...	$U_{p-1,q-1}$
$L_{0,0}$	$L_{0,1}$	...	$L_{0,q-1}$	$X_{0,0}$	$X_{0,1}$	...	$X_{0,q-1}$
$L_{1,0}$	$L_{1,1}$	...	$L_{1,q-1}$	$X_{1,0}$	$X_{1,1}$	...	$X_{1,q-1}$
...	L			...	X		
$L_{p-1,0}$	$L_{p-1,1}$	...	$L_{p-1,q-1}$	$X_{p-1,0}$	$X_{p-1,1}$	...	$X_{p-1,q-1}$

Fig. 1 Adjacent pixel blocks U, L, and X used in SMVQ

$ED(X, SCW_x) \le ED(X, SCW_y)$ for all SCW_y in Y, where SCW_y is the yth state codeword in the state codebook Y.

2.3 Chang et al.'s method

In 2013, Chang et al. [6] proposed a hybrid information hiding scheme combining SMVQ and SOC [11]. Their scheme introduced the use of TIVs for the VQ-based reversible information hiding scheme. A TIV d refers to a state codeword SCW_d that is equal to the best matched (or optimal) codeword CW_r selected by the VQ algorithm for an input pixel block X. That is, $ED(CW_r, SCW_d) \le ED(CW_r, SCW_y)$ for all SCW_y in Y, where SCW_y is the yth state codeword in the state codebook Y. Because $M = N$, there exists a unique d such that $ED(CW_r, SCW_d) = 0$ and $SCW_d = CW_r$. As a result, the SMVQ decoder can be used to reconstruct the VQ compressed image from the *TIT*. The following summary of their encoding and embedding algorithm is cited from Chang et al. [6]. The extraction algorithm is simply the inverse process of the encoding and embedding algorithm.

2.3.1 Encoding and embedding algorithm

Input: Grayscale cover image G, super codebook D, confidential message S.
Output: An output binary code stream CS.

Step 1: Encode cover image G using VQ encoding to obtain index table *VIT*.
Step 2: Transform index table *VIT* using SMVQ to create transformed index table *TIT*.
Step 3: Read transformed index d from the transformed index table *TIT*.
Step 4: Search for the SOC code of transformed index d, and read the confidential bit b from confidential message S.
Step 5: If $b = 1$ and $d \le 7$ then d is encoded by the indicator bits 10 followed by the 3-bit binary representation of d. However, if $b = 1$ and $d > 7$ then d is encoded by the indicator bits 11 followed by the $\lceil \log_2 N \rceil$-bit binary representation of d.
Step 6: Otherwise, (i.e., $b = 0$), one of the following steps is employed:
Step 6.1: If d has a SOC code in the range 0 to 2^2–2 then d is encoded by the indicator bit 0 followed by the 2-bit SOC code of d.
Step 6.2: Otherwise (i.e., d does not have an SOC code), if $b = 0$ and $d \le 7$ then d is encoded by the indicator bits 011, followed by the 3-bit binary representation of d. However, if $b = 0$ and $d > 7$ then d is encoded by the indicator bits 011 followed by the $\lceil \log_2 N \rceil$-bit binary representation of d.
Step 7: Finally, the compression code for d is concatenated to the code stream CS.
Step 8: Steps 3–7 are repeated until all indices are processed.

2.4 Lin et al.'s method

Lin et al. [16] proposed a hybrid data hiding scheme for VQ compressed images based on SOC [11] and SCM [22]. Their scheme locates the current VQ index r through a process of elimination. If r is not found in the SOC path, Lin et al.'s method [16] uses a state codebook mapping (SCM) procedure to generate a state codebook of size M for each search point within the SOC path. Specifically, given the VQ index value of the current

search point y, a list of unseen codewords from the main codebook is created. That is, all the codewords contained in any of the state codebook(s) built so far, and all the codewords referenced by the VQ index values of any search point within the SOC path are ignored. The following summary of their encoding and embedding algorithm is cited from Lin et al. [16]. Their decoding and extraction algorithm is simply the inverse process of the encoding and embedding algorithm.

2.4.1 Encoding and embedding algorithm

Input: VQ index table, per block output length ψ, SOC bit length v, secret data.
Output: Compression codes

Step 1: Input an index from the VQ index table, which is processed in raster scan order.
Step 2: Find a search point with the same VQ index r in the predefined SOC path of the index table until the current processed index cannot be encoded with any of the 2^v SOC codes, or a matched search point is found.
Step 3: If a matched search point is found, follow the SOC embedding procedure and encode r with the indicator bit 0, followed by the corresponding SOC code and $(\psi - v - 1)$ secret bits; otherwise, the SCM procedure is performed (see below).
Step 4: Go to Step 1 and encode the next VQ index until all indices have been processed.

2.4.2 State codebook mapping procedure (SCM)

Input: Current processed index r, main codebook D length N, SOC bit length v, secret data, state codebook length M, search points (i.e., indices in the SOC path), and per block output length ψ.
Output: Compression code of in process index.

Step 1: Generate a state codebook with M unseen VQ indices for each search point.
Step 2: Find an element in all state codebooks with the same VQ index as that of r.
Step 3: If an element is found at position p in the state codebook of the corresponding search point, then r is encoded with the indicator bits 10, followed by the corresponding v-bit SOC code, followed by the $\lceil \log_2 M \rceil$-bit binary representation of p, followed by $(\psi - v - \lceil \log_2 M \rceil - 2)$ secret bits. Otherwise, r is preserved, and is encoded with the indicator bits 11, followed by the $\lceil \log_2 N \rceil$-bit binary representation of r.

2.5 Chang et al.'s method

Chang et al.'s scheme [5] encodes residual SMVQ indices using the SOC algorithm with a state codebook size $M = 32$ or 64. Because M is less than the super codebook size N (i.e., $M < N$), the visual quality of the reconstructed image is reduced and their scheme is only reversible for SMVQ compressed images. Despite this, Chang et al. [5] can embed 1 bit per residual SMVQ index at excellent bit rates. The following summary of Chang et al.'s encoding and embedding algorithm [5] is cited from their paper. The decoding and extraction algorithm is simply the inverse process of the encoding and embedding algorithm.

2.5.1 Encoding and embedding algorithm

Input: Each input pixel block X of cover image G in raster scan order and secret message S.
Output: Compression code C of image G, including secret message S.

Step 1: Encode image G by using SMVQ to obtain index table *SIT*.
Step 2: Read SMVQ index x in the residual area of index table *SIT*.
Step 3: Determine the SOC code for index x, and read a secret bit b from secret message S.
Step 4: If the SOC code is determined, and the SOC code $\leq 2^2-2$. Perform one of the following steps:
Step 4.1: If $b = 1$, index x is denoted by indicator bit 1, followed by the $\lceil \log_2 M \rceil$-bit binary representation of x.
Step 4.2: If $b = 0$, index x is denoted by indicator bit 0, followed by the two-bit SOC code of index x.
Step 5: If the SOC code is not determined for index x, perform one of the following steps:
Step 5.1: If $b = 1$, index x is denoted by indicator bit 1, followed by the $\lceil \log_2 M \rceil$-bit binary representation of x.
Step 5.2: If $b = 0$, index x is denoted by indicator bit 0, followed by the two-bit binary information of 2^2-1, followed by the $\lceil \log_2 M \rceil$-bit binary representation of x.
Step 6: The compression code of index x is concatenated to the C.
Step 7: Repeat Steps 2 through 6 until all indices in the residual area of index table *SIT* have been processed.

3 The proposed method

The proposed scheme exploits the frequency distributions of SMVQ indices in the intervals [0, a] and [a + 1, M - 1] where $a = 2^w - 1$ and $w = 1, 2, 3, ..., \lceil \log_2 M \rceil$. As illustrated in Fig. 2, it can be seen that the frequency distribution of indices in the interval [0, 3] (e.g., $a = 3$) is much higher than the frequency distribution of indices in the [4, 255]. This implies that compression codes that are optimal for indices in the smaller interval [0, 3] are not optimal for indices in the larger interval [4, 255] and vice versa. Thus, if the proposed scheme can classify an SMVQ index x into either the interval [0, a] or [a + 1, M - 1], then it can use the optimal the compression codes for each interval. To achieve this goal, the proposed scheme compares a predefined threshold value t to the *SMD* value of the first state codeword, $SMD(X', scw_0)$, where the state vector X' is the boundary pixel vector $BV(X)$ as described in Section 2.2. Therefore, when the $SMD(X', scw_0) \leq t$, we assume that the SMVQ index x is in the interval [0, a] (i.e., Case 1). Otherwise, (i.e., $SMD(X', scw_0) > t$) we assume that x is in the interval [a + 1, M - 1] (i.e., Case 2). During the development of the proposed scheme, we experimented with more than two intervals such as 3, 4, and up to 10 intervals, but we found that the two intervals [0, a] and [a + 1, M - 1] produced the optimal result in terms of bit rate. Therefore, we decided to simplify the proposed scheme by using only two intervals. The proposed scheme exploits a feature of the SMVQ frequency distributions for the intervals [0, a] and [a + 1, M - 1], as shown in Fig. 2, to achieve a better bit rate and embedding rate.

We observed that Case 1 SMVQ indices have a higher probability of belonging to the smaller interval [0, a] (e.g., [0, 3]) than the larger interval [a + 1, $M - 1$] (e.g., [4, 255]), where $a = 2^w - 1$ and $w = 1, 2, 3, ..., \lceil \log_2 M \rceil$. Specifically, when $a = 3$, $t = 8$, and $M = N = 256$, the

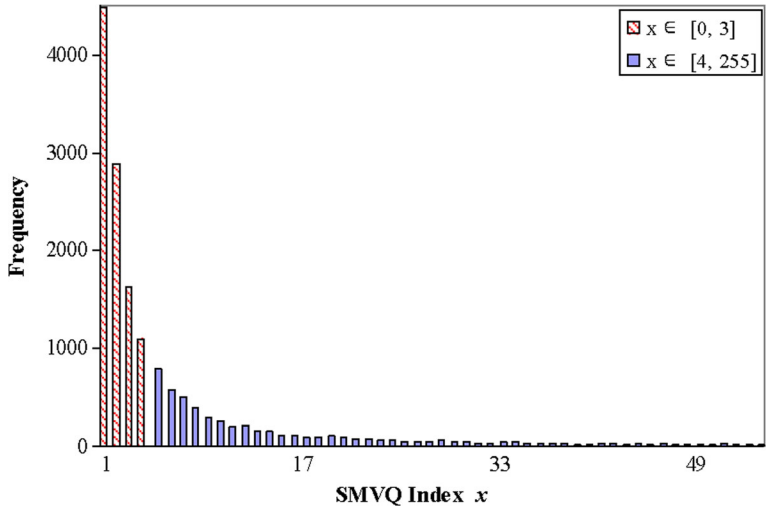


Fig. 2 The SMVQ frequency distribution of the test image Lena with $M = N = 256$

probability that a Case 1 SMVQ index belongs to the intervals [0, 3] and [4, 255] is 0.833 and 0.167, respectively; see Fig. 3. To exploit this characteristic, we designed the Case 1 compression codes as shown in Table 1a, where $\alpha = 2^{\ell}$, $\ell = 0, 1, 2, 3, ..., \lfloor((\log_2 M) - 2)/15\rfloor$. For example, when $\ell = 1$ and $M = N = 256$, only Case 1 SMVQ indices that are less than 15 can be compressed (i.e., these SMVQ indexes can be represented with less than $\lceil \log_2 M \rceil$ bits). Additionally, when $a = 3$ and $t = 8$, the probability that a Case 1 SMVQ index belongs to the smaller interval [0, 3] is 0.833. Therefore, with $\ell = 1$ and $t = 8$, the proposed scheme can compress at least 83.3% of all Case 1 SMVQ indices.

In contrast, we observed that the probability of Case 2 SMVQ indices belonging to the smaller interval [0, a] (e.g., [0, 3]) and the larger interval [$a + 1$, $M - 1$] (e.g., [4, 255]) is similar. Specifically, when $a = 3$, $t = 8$, and $M = N = 256$, the probability that a Case 2 SMVQ

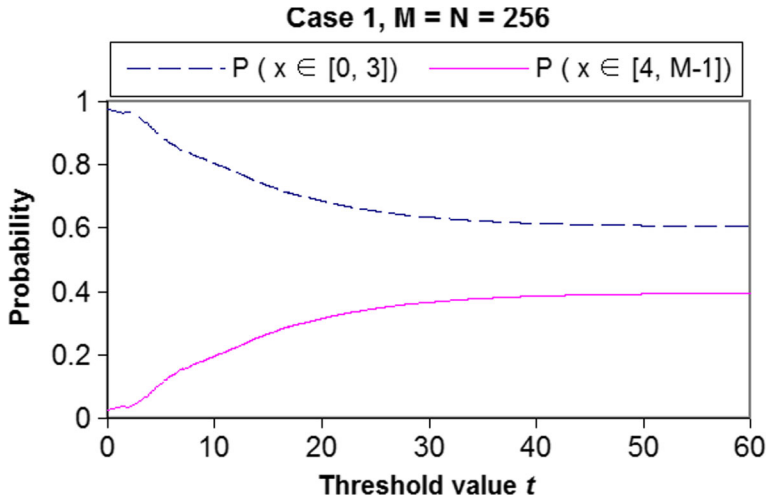


Fig. 3 The probability of a Case 1 SMVQ index from the SMVQ index tables of the test images in Fig. 7, belonging to the intervals [0, 3] and [4, M - 1] with various values of t and $M = N = 256$

Table 1 The proposed compression codes

	Intervals	Compression code z of x
a) For Case 1 SMVQ indices where $\alpha = 2^{\ell}$ (e.g., $\ell = 0$, 1, 2, or 3)		
Case 1.1	$x = 0$	$z = 0$
Case 1.2	$1 \le x \le \alpha$	$z = 10 \| \| \ell$-bit representation of $(x - 1)$
Case 1.3	$\alpha + 1 \le x \le 3\alpha$	$z = 110 \| \| (\ell + 1)$-bit representation of $(x - \alpha - 1)$
Case 1.4	$3\alpha + 1 \le x \le 7\alpha$	$z = 1110 \| \| (\ell + 2)$-bit representation of $(x - 3\alpha - 1)$
Case 1.5	$7\alpha + 1 \le x \le 15\alpha$	$z = 11110 \| \| (\ell + 3)$-bit representation of $(x - 7\alpha - 1)$
Case 1.6	$15\alpha + 1 \le x \le M - 1$	$z = 11111 \| \| \lceil \log_2 M \rceil$-bit representation x
b) For Case 2 SMVQ indices where $\beta = 2^{\ell + 1}$ (e.g., $\ell = 0$, 1, 2, or 3)		
Case 2.1	$0 \le x \le \beta - 1$	$z = 00 \| \| (\ell + 1)$-bit representation of $(x - 1)$
Case 2.2	$\beta \le x \le 3\beta - 1$	$z = 01 \| \| (\ell + 2)$-bit representation of $(x - \beta - 1)$
Case 2.3	$3\beta \le x \le 7\beta - 1$	$z = 10 \| \| (\ell + 3)$-bit representation of $(x - 3\beta - 1)$
Case 2.4	$7\beta \le x \le M - 1$	$z = 11 \| \| \lceil \log_2 M \rceil$-bit representation x

index belongs to the intervals [0, 3] and [4, 255] are 0.589 and 0.411, respectively; see Fig. 4. Because of this characteristic, the proposed scheme cannot appropriately determine whether a Case 2 SMVQ index will belong to the smaller or larger interval. Therefore, we designed the length of the Case 2 compression codes as shown in Table 1b to optimize the compression performance of the proposed method. The details of the Case 2 compression codes are presented in Table 1b, where $\beta = 2\alpha = 2^{\ell+1}$. It is clear from Table 1 that the selection of appropriate values for ℓ and t is necessary to achieve a good compression. The details of how to select these values for a given codebook size are presented in Section 4.1.

The works of Chang et al. [6], Wang et al. [24], and Chang et al. [5] all build an SMVQ index table from the input cover image G and then compress each SMVQ index in raster scan order, creating spaces to embed secret information. In contrast, the proposed scheme compresses each SMVQ index immediately after it is generated by the SMVQ algorithm (i.e.,

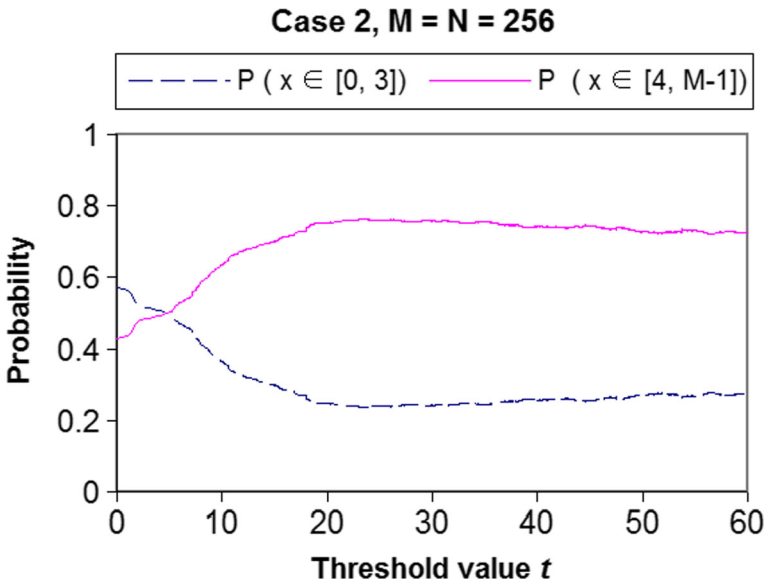


Fig. 4 The probability of a Case 2 SMVQ index from the SMVQ index tables of the test images in Fig. 7, belonging to the intervals [0, 3] and [4, M - 1] with various values of t and $M = N = 256$

SMVQ indices are not stored in an SMVQ index table). This ensures that the *SMD* values are not recomputed when the proposed scheme classifies an SMVQ index as Case 1 or 2 by comparing the *SMD* value of the first state codeword with the threshold value t. The details of the encoding and embedding algorithms are presented in Section 3.1, and the details of the decoding and extraction algorithms are presented in Section 3.2.

3.1 The encoding and embedding phase

The grayscale cover image G sized $H \times W$ is divided into nonoverlapping pixel blocks X, each of size $k = p \times q$. The proposed scheme processes these pixel blocks in raster scan order. If the current pixel block X is located in the first row or column, then the proposed scheme computes the VQ index r and outputs the $\lceil \log_2 N \rceil$-bit binary representation of r, followed by the next n secret bits $b_1 b_2 ... b_n$ to the binary code stream CS (i.e., $CS = CS \| b_1 b_2 ... b_n$), where the symbol '||' denotes the concatenation operation. Otherwise (i.e., X is not in the first row or column), the proposed scheme constructs the state vector X' (i.e., the boundary pixel vector of the current pixel block X), builds a state codebook of size M, and computes the SMVQ index x. If the *SMD* between the first state codeword scw_0 and the state vector X' is less than or equal to the predefined threshold value t (i.e., $SMD(X', scw_0) \le t$), the SMVQ index x is classified as Case 1, and the compression code z is generated by following the rules defined in Table 1a. Otherwise (i.e., $SMD(X', scw_0) > t$), the SMVQ index x is classified as Case 2, and the compression code z is generated by using the rules defined in Table 1b. Then, the next n secret bits $b_1 b_2 ... b_n$ from the secret message S are embedded into z (i.e., $z = z \| b_1 b_2 ... b_n$), and z is appended to the binary output code stream (i.e., $CS = CS \| z$). The flowchart of the proposed encoding and embedding scheme is shown in Fig. 5. The proposed encoding and embedding algorithm is summarized as follows:

3.1.1 The encoding and embedding algorithm

Input: A grayscale cover image G sized $H \times W$, codebook D sized N, secret message S, and predefined values ℓ, t, M, and n, where n = 1, 2, 3, or 4.

Output: A binary code stream CS.

Step 1: Divide G into nonoverlapping pixel blocks X, each of size $p \times q$.
Step 2: Read n secret bits $b_1 b_2 ... b_n$ from S.
Step 3: Read a pixel block X from G in raster scan order.
Step 4: If X is in the first row or column, compute VQ index r and output $\lceil \log_2 N \rceil$-bit representation of r, followed by n secret bits $b_1 b_2 ... b_n$ to the code stream CS.
Step 5: Else (i.e., X is not in the first row or column),
Step 5.1: Build the state vector X' and the M-sized state codebook of X.
Step 5.2: Compute the SMVQ index x.
Step 5.3: If $SMD(X', scw_0) \le t$, then follow the rules in Table 1a to generate the compression code z of x, where scw_0 is the first state codeword in the state codebook.
Step 5.4: Else (i.e., $SMD(X', scw_0) > t$), follow the rules in Table 1b to generate the compression code z of x.
Step 5.5: Append n secret bits $b_1 b_2 ... b_n$ to z (i.e., $z = z \| b_1 b_2 ... b_n$).
Step 5.6: Update the output code stream as $CS = CS \| z$.
Step 6: Repeat *steps 3* to 5 *until all pixel blocks X are processed.*

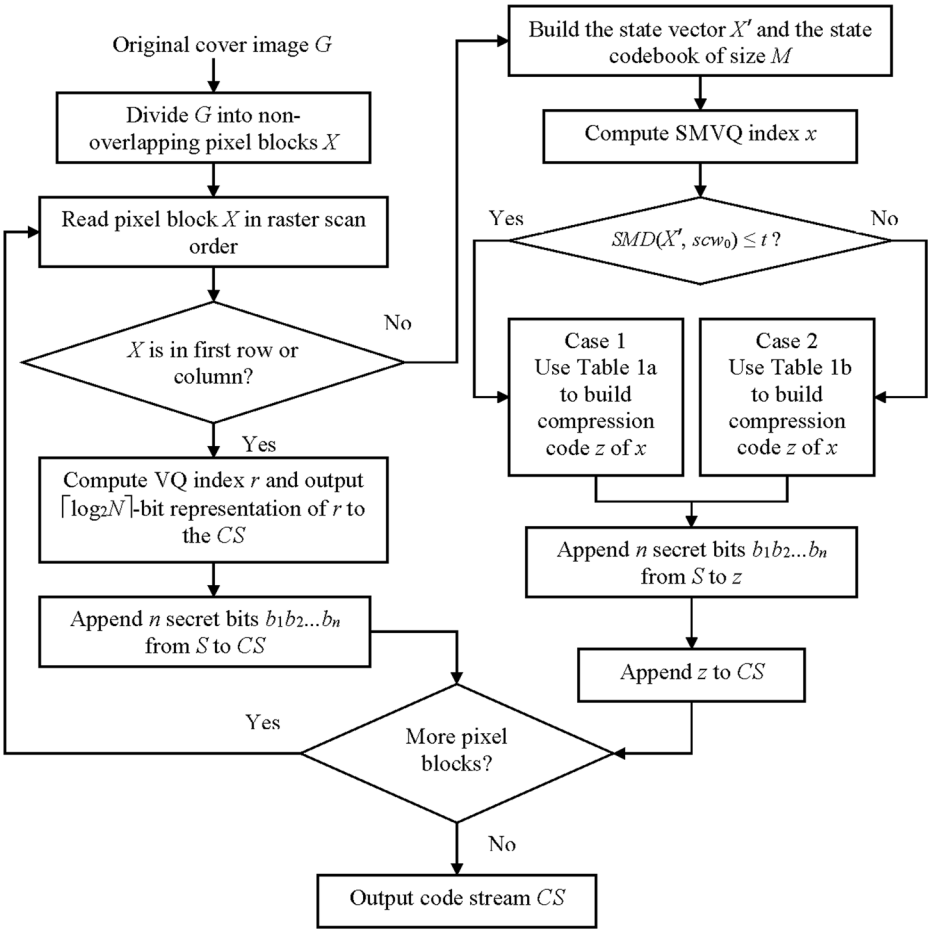


Fig. 5 The flowchart of the proposed encoding and embedding scheme

Step 7: *O*utput *t*he bi*n*ar*y* $c_o$de $s_t$ream *CS*.

3.2 The decoding and extracting phase

The decoding and extracting phase recovers the reconstructed cover image *G*' sized $H \times W$ and secret message *S*. The expected inputs are the received binary code stream *CS*, the main codebook *D* sized *N*, and the preset values ℓ, *t*, *M*, and *n*, where $n = 1, 2, 3$, or 4. First, *S* and *G*' are initialized to be empty. Then *G*' is divided into nonoverlapping pixel blocks *X*, each of size $k = p \times q$. These pixel blocks are recovered in raster scan order. Let *X* be the currently recovered pixel block. If *X* is located in the first row or column, then the VQ index *r* is recovered as the decimal value *e* of the next $\lceil \log_2 N \rceil$ bits from the code stream *CS* (i.e., $r = e$). The values of the *r*th codeword, cw_r, from *D* are used to recover *X*. Finally, the recovered pixel block *X* is inserted into the reconstructed cover image *G*' in raster scan order. The next *n* consecutive bits $c_i c_{i+1} \ldots c_{i+n-1}$ are extracted from *CS* as *n* secret bits $b_1 b_2 \ldots b_n$ and appended to the secret message *S* (i.e., $S = S \| b_1 b_2 \ldots b_n$).

If X is not located in the first row or column, the state vector X' and a state codebook of size M are constructed. If the SMD of the first state codeword scw_0 with the state vector X' is less than or equal to the predefined threshold value t (i.e., $SMD(X', scw_0) \leq t$), the SMVQ index x is recovered by following the rules defined in Table 2a. Otherwise (i.e., $SMD(X', scw_0) > t$), x is recovered by following the rules defined in Table 2b. The current pixel block X is reconstructed using the values of the xth state codeword, scw_x. The reconstructed pixel block X is then inserted into G' in raster scan order. The next n consecutive bits $c_i c_{i+1} \ldots c_{i+n-1}$ are extracted from CS as n secret bits $b_1 b_2 \ldots b_n$ and added to the secret message S (i.e., $S = S \| b_1 b_2 \ldots b_n$). The above steps are repeated until the cover image G' is reconstructed and the secret message S is recovered. The flowchart of the proposed decoding and extracting scheme is shown in Fig. 6. The decoding and extracting algorithm of the proposed scheme is summarized as follows:

3.2.1 The decoding and extracting algorithm

Input: The received binary code stream CS, codebook D sized N, and preset values ℓ, t, M, and n, where $n = 1, 2, 3$, or 4.

Output: The extracted secret message S and the reconstructed cover image G' sized $H \times W$.

Step 1: Initialize S and G' to be empty.
Step 2: Divide G' into nonoverlapping pixel blocks X, each of size $p \times q$.
Step 3: Let X be the pixel block that is currently recovered in raster scan order.
Step 4: If X is in the first row or column, then recover r as the decimal value e of the next $\lceil \log_2 N \rceil$ bits from CS (i.e., $r = e$).

Table 2 The proposed decoding rules

	Bits extracted from CS	Recovery rules
a) For Case 1 SMVQ indices where $\alpha = 2^{\ell}$ (e.g., $\ell = 0, 1, 2$, or 3)		
Case 1.1	$c_i = 0$	Recover x as 0, $x = 0$
Case 1.2	$c_i = 1, c_{i+1} = 0$	Recover x as the decimal value of the next ℓ bits from CS plus 1, $x = (c_{i+2} \| \ldots \| c_{i+\ell+1})_{10} + 1$
Case 1.3	$c_i = 1, c_{i+1} = 1, c_{i+2} = 0$	Recover x as the decimal value of the next $\ell + 1$ bits from CS plus $\alpha + 1$, $x = (c_{i+3} \| \ldots \| c_{i+\ell+3})_{10} + \alpha + 1$
Case 1.4	$c_i = 1, c_{i+1} = 1, c_{i+2} = 1, c_{i+3} = 0$	Recover x as the decimal value of the next $\ell + 2$ bits from CS plus $3\alpha + 1$, $x = (c_{i+4} \| \ldots \| c_{i+\ell+5})_{10} + 3\alpha + 1$
Case 1.5	$c_i = 1, c_{i+1} = 1, c_{i+2} = 1, c_{i+3} = 1, c_{i+4} = 0$	Recover x as the decimal value of the next $\ell + 3$ bits from CS plus $7\alpha + 1$, $x = (c_{i+5} \| \ldots \| c_{i+\ell+7})_{10} + 7\alpha + 1$
Case 1.6	$c_i = 1, c_{i+1} = 1, c_{i+2} = 1, c_{i+3} = 1, c_{i+4} = 1$	Recover x as the decimal value of the next $\lceil \log_2 M \rceil$ bits from CS, $x = (c_{i+5} \| \ldots \| c_{i+\gamma})_{10}$, where $\gamma = \lceil \log_2 M \rceil + 4$
b) For Case 2 SMVQ indices where $\beta = 2\alpha = 2^{\ell+1}$ (e.g., $\ell = 0, 1, 2$, or 3)		
Case 2.1	$c_i = 0, c_{i+1} = 0$	Recover x as the decimal value of the next $\ell + 1$ bits from CS, $x = (c_{i+2} \| \ldots \| c_{i+\ell+2})_{10}$
Case 2.2	$c_i = 0, c_{i+1} = 1$	Recover x as the decimal value of the next $\ell + 2$ bits from CS plus β, $x = (c_{i+2} \| \ldots \| c_{i+\ell+3})_{10} + \beta$
Case 2.3	$c_i = 1, c_{i+1} = 0$	Recover x as the decimal value of the next $\ell + 3$ bits from CS plus 3β, $x = (c_{i+2} \| \ldots \| c_{i+\ell+4})_{10} + 3\beta$
Case 2.4	$c_i = 1, c_{i+1} = 1$	Recover x as the decimal value of the next $\lceil \log_2 M \rceil$ bits from CS, $x = (c_{i+2} \| \ldots \| c_{i+\delta+1})_{10}$, where $\delta = \lceil \log_2 M \rceil$

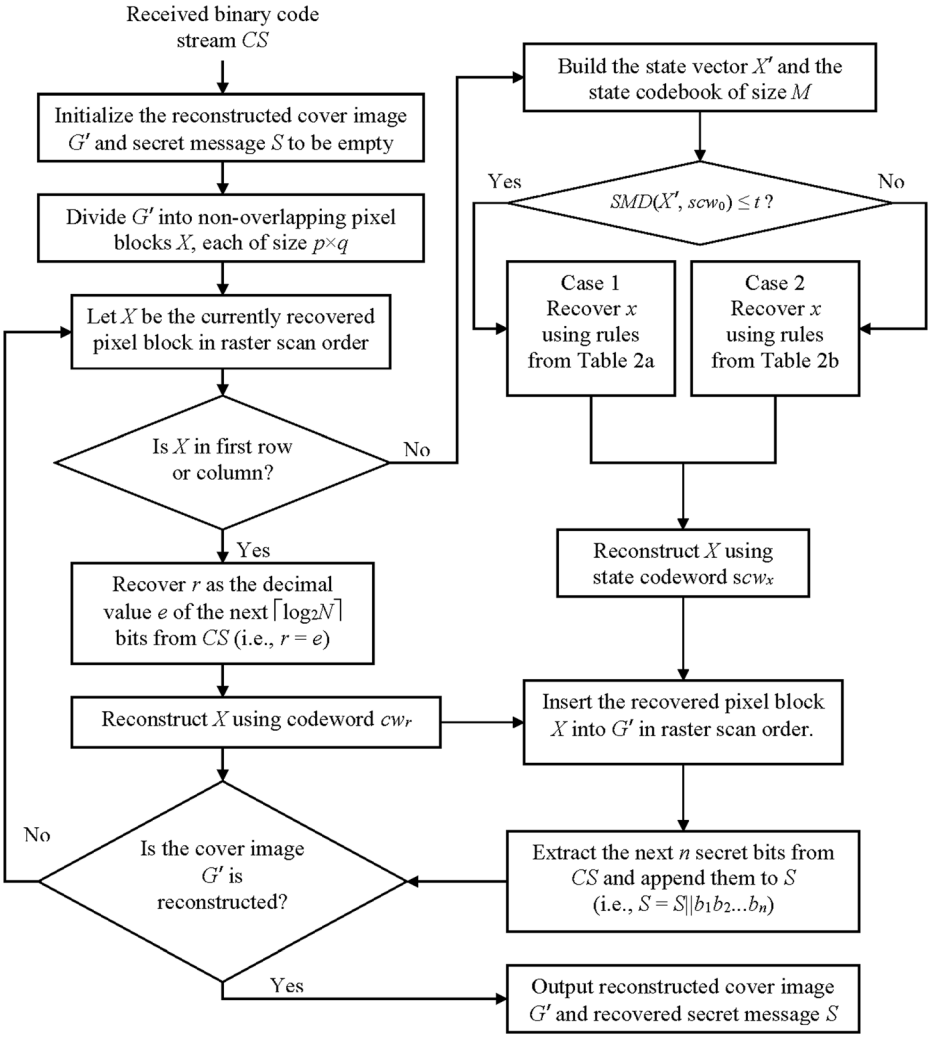


Fig. 6 The flowchart of the proposed decoding and extracting scheme

Step 4.1: Reconstruct X using the codeword cw_r.
Step 5: Else (i.e., X is not in the first row or column),
Step 5.1: Build the state vector X' and the state codebook of size M.
Step 5.2: If $SMD(X', scw_0) \le t$, then follow the rules in Table 2a to recover x.
Step 5.3: Else (i.e., $SMD(X', scw_0) > t$), use the rules in Table 2b to recover x.
Step 5.4: Reconstruct X using the state codeword scw_x.
Step 6: Insert the recovered *pixel block* X into G' in raster scan order.
Step 7: Extract the next n secret bits $b_1b_2..b_n$ from the code stream CS and append them to S (i.e., $S = S||b_1b_2...b_n$).
Step 8: Repeat steps 3 to 7 until the cover image G' is reconstructed.
Step 9: Output the extracted secret message S and reconstructed cover image G' sized $H \times W$.

4 Experimental results and discussions

This section compares the experimental results of the proposed scheme with the recent works of Chang et al. [6], Wang et al. [24], Lin et al. [16], and Chang et al. [5]. The cover images sized 512 × 512 used in all experiments are shown in Fig. 7. The four main codebooks used are of sizes $N = 128$, 256, 512, and 1024. The LBG algorithm [17] was used to generate the four main codebooks. Each main codebook contains 16-dimensional codewords (i.e., $k = 4 \times 4$). Nonoverlapping 4 × 4 blocks were used for both VQ and SMVQ (i.e., $p \times q = 4 \times 4$). All experiments were implemented using the Dev-C++ version 5.9.2 software running on Intel Core i7, 2.4 GHz CPU, and 8 GB RAM hardware platform. The library function rand() was used to generate the binary secret message S. Bit rate (BR), embedding rate (ER), embedding efficiency (EE), peak signal-to-noise ratio ($PSNR$), and execution time were used to compare the performances of the related works [5, 6, 16, 24] with the proposed scheme with $n = 1$, 2, 3, and 4.

The BR measures the number of bits required to represent one grayscale pixel in bits per pixel (bpp), and it is used to evaluate the compression performance. The BR is defined as the size of the output code stream $|CS|$ divided by the number of pixels in a grayscale cover image—that is, $BR = |CS| / (H \times W)$, where H and W are the height and width of the cover image. A lower BR indicates a better compression performance.

The ER measures the number of secret bits embedded per VQ or SMVQ index in bits per index (bpi), and it is used to evaluate embedding capacity. The ER is defined as the number of

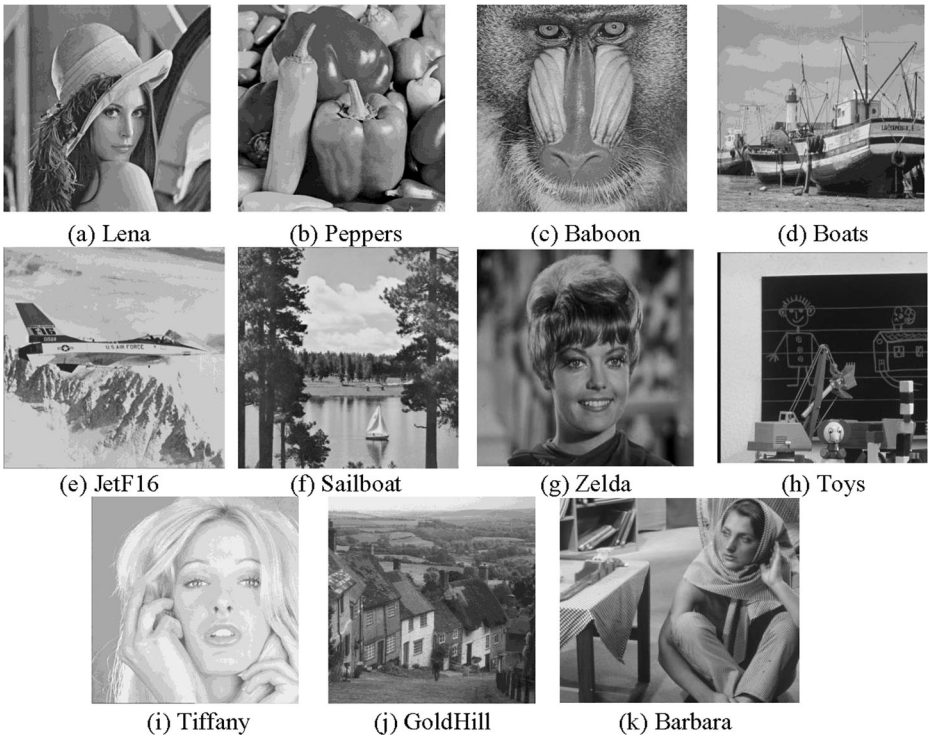


Fig. 7 Grayscale test images sized 512 × 512

concealed secret bits $|S|$ divided by the number of indices in the VQ or SMVQ index table of size $(H/p) \times (W/q)$—that is, $ER = (k \times |S|) / (H \times W)$, where $k = p \times q$. A higher ER means that a larger secret message can be transmitted.

The EE measures the number of hidden secret bits that can be transmitted when one bit of the output code stream CS is delivered. It is defined as the number of hidden secret bits $|S|$ divided by the size of the output code stream $|CS|$—that is, $EE = |S| / |CS| = ER / (k \times BR)$, where $k = p \times q$. A higher EE value indicates that more secret information is transmitted for each bit of the output code stream CS.

The $PSNR$ evaluates the visual quality of a recovered image. It expresses the difference between the original and the recovered images in decibels (dB). The $PSNR$ is calculated as

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{H \times W \times (255)^2}{\sum_{i=0}^{H-1}\sum_{j=0}^{W-1}\left(G_{ij}-G'_{ij}\right)^2}\right),$$

where H is the height of the image, W is the width of the image, and G_{ij} and G'_{ij} are the pixel values in the ith row and jth column of the original and the recovered image, respectively. A higher $PSNR$ value indicates a smaller difference between the original and the recovered images.

4.1 Selection of values for ℓ and t

The values of ℓ and t are critical factors in determining the size of the proposed scheme's output code stream, where $\alpha = 2^{\ell}$ and $\beta = 2\alpha = 2^{\ell+1}$. In order to determine the optimal values of ℓ and t, we tested the proposed scheme with all combinations of the ordered pair (ℓ, t) for the test images in Fig. 7, where $0 \le \ell \le \lfloor((\log_2 M) - 2)/15\rfloor$; $1 \le t \le \lceil 255\sqrt{7}\rceil = 675$; $M = 32$, 64, and N; and $N = 128$, 256, 512, and 1024. For a specific main codebook size N and a state codebook size M, there is only one optimal pair (ℓ, t). It is defined as the order pair that generates the lowest pure bit rate (PBR) (i.e., the BR without embedding any secret bits) for all combinations of ℓ and t. Table 3 lists the optimal values of ℓ and t for all codebook sizes. For example, the optimal pairs (ℓ, t) with main codebook sizes $N = 128$, 256, 512, and 1024 and state codebook size $M = N$ are (0, 6), (1, 10), (2, 8), and (3, 9), respectively. The experimental results in terms of bit rate of the proposed scheme with these optimal pairs and $n = 1$, 2, 3, and 4 are listed in Tables 4 and 5. The detailed experimental results of the PBRs with all combinations of ℓ and t for all codebook sizes can be found in (http://tinyurl.com/jmby5nq).

Table 3 Optimal values of ℓ and t with $M = 32$, 64, N, and $N = 128$, 256, 512, 1024

N	M					
	32		64		N	
	ℓ	t	ℓ	t	ℓ	t
128	0	4	0	4	0	6
256	0	8	0	8	1	10
512	0	4	0	4	2	8
1024	0	1	0	1	3	9

Table 4 *BR*s of the proposed scheme for $(\ell, t) = (0, 6), (1, 10)$, $n = 1, 2, 3, 4$, and $M = N = 128, 256$

Images	$M = N = 128$ $(\ell, t) = (0, 6)$				$M = N = 256$ $(\ell, t) = (1, 10)$			
	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
Lena	0.285	0.347	0.410	0.472	0.331	0.394	0.456	0.519
Peppers	0.282	0.345	0.407	0.470	0.320	0.383	0.445	0.508
Baboon	0.427	0.490	0.552	0.615	0.494	0.556	0.619	0.681
Boats	0.277	0.339	0.402	0.464	0.320	0.383	0.445	0.508
JetF16	0.257	0.319	0.382	0.444	0.301	0.364	0.426	0.489
Sailboat	0.293	0.356	0.418	0.481	0.343	0.405	0.468	0.530
Zelda	0.284	0.346	0.409	0.471	0.335	0.398	0.460	0.523
Toys	0.214	0.277	0.339	0.402	0.261	0.324	0.386	0.449
Tiffany	0.223	0.285	0.348	0.410	0.246	0.308	0.371	0.433
GoldHill	0.334	0.397	0.459	0.522	0.385	0.448	0.510	0.573
Barbara	0.345	0.407	0.470	0.532	0.403	0.465	0.528	0.590
Average	0.293	0.355	0.418	0.480	0.340	0.403	0.465	0.528

4.2 Comparison of the proposed scheme with related works

This section compares the performance of the proposed scheme with the related works [5, 6, 16, 24]. *BR*, *ER*, *EE*, and *PSNR* were used to examine each scheme's performance. For fair comparisons, we modified the proposed scheme to match the embedding capacities of each related work. When two schemes have the same embedding capacity, the scheme with the lower average *BR* has a better compression performance. The comparative results among the simulated methods are presented in Table 6.

In 2013, Chang et al. [6] designed a reversible information hiding scheme that embeds 1 bpi at very good average bit rates (e.g., 0.407 bpp with $N = 256$). Their scheme compresses all TIVs that are less than or equal to 7—that is, it uses less than $\lceil \log_2 N \rceil$ bits to represent a TIV (i.e., fewer bits than the conventional VQ algorithm). It can be seen from Table 6 that the proposed scheme with $n = 1$ achieves lower average *BR*s for the same *ER* of 1 bpi. This indicates that the proposed scheme with $n = 1$ also obtains higher average *EE*s in all test cases. Based on these results, we can conclude that the compression performance of the proposed

Table 5 *BR*s of the proposed scheme for $(\ell, t) = (2, 8), (3, 9)$, $n = 1, 2, 3, 4$, and $M = N = 512, 1024$

Images	$M = N = 512$ $(\ell, t) = (2, 8)$				$M = N = 1024$ $(\ell, t) = (3, 9)$			
	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 1$	$n = 2$	$n = 3$	$n = 4$
Lena	0.388	0.451	0.513	0.576	0.454	0.516	0.579	0.641
Peppers	0.379	0.442	0.504	0.567	0.432	0.494	0.557	0.619
Baboon	0.550	0.612	0.675	0.737	0.610	0.672	0.735	0.797
Boats	0.374	0.437	0.499	0.562	0.430	0.492	0.555	0.617
JetF16	0.374	0.436	0.499	0.561	0.414	0.477	0.539	0.602
Sailboat	0.418	0.481	0.543	0.606	0.478	0.541	0.603	0.666
Zelda	0.386	0.448	0.511	0.573	0.450	0.513	0.575	0.638
Toys	0.305	0.367	0.430	0.492	0.353	0.416	0.478	0.541
Tiffany	0.314	0.376	0.439	0.501	0.356	0.418	0.481	0.543
GoldHill	0.440	0.502	0.565	0.627	0.497	0.559	0.622	0.684
Barbara	0.463	0.526	0.588	0.651	0.521	0.583	0.646	0.708
Average	0.399	0.462	0.524	0.587	0.454	0.516	0.579	0.641

Table 6 Summary of the average *ER*, *BR*, *EE,* and *PSNR* of Chang et al. [6], Wang et al. [24], Lin et al. [16], Chang et al. [5], and the proposed scheme for $M = 32, 64$, N; $N = 128, 256, 512, 1024$; $\psi = 7, 8, 8, 9$; and $(l, t) = (0, 4), (0, 8), (0, 1), (0, 6), (1, 10), (2, 8), (3, 9)$

Codebook		$N = 128$	$N = 256$	$N = 512$	$N = 1024$
Chang et al. [6]	*ER*	1	1	1	1
	BR	0.352	0.407	0.478	0.549
	EE	0.178	0.154	0.131	0.114
	PSNR	27.567	29.266	30.144	30.852
Wang et al. [24] (NH)	*ER*	1.667	1.306	1.007	0.798
	BR	0.354	0.378	0.414	0.456
	EE	0.294	0.216	0.152	0.109
	PSNR	27.567	29.266	30.144	30.852
Wang et al. [24] (OH)	*ER*	2.335	1.861	1.448	1.157
	BR	0.396	0.413	0.441	0.478
	EE	0.369	0.282	0.205	0.151
	PSNR	27.567	29.266	30.144	30.852
Lin et al. [16]	*ER*	2.748	3.035	2.467	2.706
	BR	0.448	0.519	0.546	0.625
	EE	0.383	0.365	0.282	0.271
	PSNR	27.567	29.266	30.144	30.852
		$w = 7$	$w = 8$	$w = 8$	$w = 9$
Chang et al. [5] with $M = 32$	*ER*	0.984	0.984	0.984	0.984
	BR	0.349	0.368	0.387	0.398
	EE	0.176	0.167	0.159	0.155
	PSNR	26.651	27.448	27.135	26.813
Chang et al. [5] with $M = 64$	*ER*	0.984	0.984	0.984	0.984
	BR	0.393	0.417	0.440	0.455
	EE	0.156	0.147	0.140	0.135
	PSNR	27.236	28.397	28.380	28.099
Proposed scheme	*ER*	0.984	0.984	0.984	0.984
($ER = 0.984$, $M = 32$)	*BR*	0.275	0.308	0.343	0.367
	EE	0.224	0.200	0.179	0.168
	PSNR	26.651	27.448	27.135	26.813
	(ℓ, t)	(0, 4)	(0, 8)	(0, 4)	(0, 1)
Proposed scheme	*ER*	0.984	0.984	0.984	0.984
($ER = 0.984$, $M = 64$)	*BR*	0.285	0.323	0.365	0.398
	EE	0.216	0.190	0.168	0.154
	PSNR	27.236	28.397	28.380	28.099
	(ℓ, t)	(0, 4)	(0, 8)	(0, 4)	(0, 1)
Proposed scheme	*ER*	1.667	1.306	1.007	0.798
(*ER* is fixed as the same as that of Wang et al. [24] [NH])	*BR*	0.335	0.359	0.400	0.441
	EE	0.311	0.227	0.157	0.113
	PSNR	27.567	29.266	30.144	30.852
	(ℓ, t)	(0, 6)	(1, 10)	(2, 8)	(3, 9)
Proposed scheme	*ER*	2.335	1.861	1.448	1.157
(*ER* is fixed as he same as that of Wang et al. [24] [OH])	*BR*	0.376	0.394	0.427	0.464
	EE	0.388	0.295	0.212	0.156
	PSNR	27.567	29.266	30.144	30.852
	(ℓ, t)	(0, 6)	(1, 10)	(2, 8)	(3, 9)
Proposed scheme	*ER*	2.748	3.035	2.467	2.706
(*ER* is fixed as the same as that of Lin et al. [16])	*BR*	0.402	0.467	0.491	0.561
	EE	0.427	0.406	0.314	0.301
	PSNR	27.567	29.266	30.144	30.852
	(ℓ, t)	(0, 6)	(1, 10)	(2, 8)	(3, 9)
Proposed scheme with $n = 1$	*ER*	1	1	1	1
	BR	0.293	0.340	0.399	0.454
	EE	0.213	0.184	0.157	0.138
	PSNR	27.567	29.266	30.144	30.852

Table 6 (continued)

Codebook		$N = 128$	$N = 256$	$N = 512$	$N = 1024$
	(ℓ, t)	(0, 6)	(1, 10)	(2, 8)	(3, 9)
Proposed scheme with $n = 2$	*ER*	2	2	2	2
	BR	0.355	0.403	0.462	0.516
	EE	0.352	0.311	0.271	0.242
	PSNR	27.567	29.266	30.144	30.852
	(ℓ, t)	(0, 6)	(1, 10)	(2, 8)	(3, 9)
Proposed scheme with $n = 3$	*ER*	3	3	3	3
	BR	0.418	0.465	0.524	0.579
	EE	0.449	0.403	0.358	0.324
	PSNR	27.567	29.266	30.144	30.852
	(ℓ, t)	(0, 6)	(1, 10)	(2, 8)	(3, 9)
Proposed scheme with $n = 4$	*ER*	4	4	4	4
	BR	0.480	0.528	0.587	0.641
	EE	0.521	0.474	0.426	0.390
	PSNR	27.567	29.266	30.144	30.852
	(ℓ, t)	(0, 6)	(1, 10)	(2, 8)	(3, 9)

scheme is better than that of Chang et al.'s scheme [6]. The experimental results in terms of bit rate of Chang et al.'s scheme [6] and the proposed method with $n = 1$ for each test image shown in Fig. 7 can be found in Tables A1 − A2 in (http://tinyurl.com/jmby5nq).

In 2014, Wang et al. [24] proposed a reversible information hiding scheme based on the SMVQ algorithm and a modified LAC algorithm. Tables 7 and 8 compare the *PBR*s of Wang et al.'s scheme [24] with those of the proposed scheme. These data indicate that the proposed scheme obtains lower *PBR*s in all test cases. Therefore, we can conclude that the proposed scheme improves the compression performance of Wang et al.'s method [24].

To fairly compare with their normal hiding (NH) and over hiding (OH) strategies, we modified the proposed scheme to embed the same number of secret bits in each test case as Wang et al.'s NH and OH [24]. Table 6 shows that, in all test cases, the proposed scheme achieves lower average *BR*s, implying that the proposed scheme also obtains higher average

Table 7 Comparison of the *PBR*s between Wang et al.'s scheme [24] and the proposed scheme with $(\ell, t) = (0, 6)$, (1, 10), and $M = N = 128, 256$

Images	$N = 128$		$N = 256$	
	Wang et al. [24]	Proposed scheme $(\ell, t) = (0, 6)$	Wang et al. [24]	Proposed scheme $(\ell, t) = (1, 10)$
Lena	0.243	0.222	0.290	0.269
Peppers	0.237	0.220	0.279	0.258
Baboon	0.366	0.365	0.432	0.431
Boats	0.240	0.214	0.279	0.258
JetF16	0.223	0.194	0.266	0.239
Sailboat	0.250	0.231	0.302	0.280
Zelda	0.236	0.221	0.286	0.273
Toys	0.188	0.152	0.225	0.199
Tiffany	0.190	0.160	0.220	0.183
GoldHill	0.281	0.272	0.329	0.323
Barbara	0.296	0.282	0.354	0.340
Average	0.250	0.230	0.297	0.278

Table 8 Comparison of the *PBR*s between Wang et al.'s scheme [24] and the proposed scheme with (ℓ, t) = (2, 8), (3, 9), and $M = N$ = 512, 1024

Images	N = 512		N = 1024	
	Wang et al. [24]	Proposed scheme (ℓ, t) = (2, 8)	Wang et al. [24]	Proposed scheme (ℓ, t) = (3, 9)
Lena	0.339	0.326	0.399	0.391
Peppers	0.328	0.317	0.376	0.369
Baboon	0.509	0.487	0.589	0.547
Boats	0.330	0.312	0.381	0.367
JetF16	0.329	0.311	0.361	0.352
Sailboat	0.371	0.356	0.435	0.416
Zelda	0.327	0.323	0.391	0.388
Toys	0.251	0.242	0.299	0.291
Tiffany	0.269	0.251	0.297	0.293
GoldHill	0.386	0.377	0.450	0.434
Barbara	0.419	0.401	0.484	0.458
Average	0.351	0.337	0.406	0.391

*EE*s. Based on these results, we can conclude that the proposed scheme improves the compression performance of Wang et al.'s scheme [24]. The experimental results with respect to the bit rate of Wang et al.'s scheme [24] and that of the proposed method for each test image shown in Fig. 7 can be referenced from Tables A3 − A6 in (http://tinyurl.com/jmby5nq).

In 2015, Lin et al. [16] proposed a VQ-based information hiding scheme combining the SOC [10] and the SCM [22]. The compression performance of their scheme relies on the number of VQ indices within the SOC path and the size of the state codebook generated by the SCM, denoted as SOL and M, respectively. They reported that the optimal values of SOL and M are 4 and 8 for most images. Tables 9 and 10 compare the *PBR*s between Lin et al.'s scheme [16] and the proposed scheme for SOL = 4 and M = 8. In all test cases, the proposed scheme achieves better *PBR*s. This suggests that, for the same embedding capacities, the proposed scheme will achieve lower bit rates.

Table 9 Comparison of *PBR*s between Lin et al.'s scheme [16] and the proposed scheme for SOL = 4, M = 8, (ℓ, t) = (0, 6), (1, 10), and $M = N$ = 128, 256

N	128		256	
Images	Proposed Scheme $M = N$ = 128 (ℓ, t) = (0, 6)	Lin et al. [16]	Proposed Scheme $M = N$ = 256 (ℓ, t) = (1, 10)	Lin et al. [16]
Lena	0.222	0.268	0.269	0.324
Peppers	0.220	0.266	0.258	0.318
Baboon	0.365	0.388	0.431	0.467
Boats	0.214	0.269	0.258	0.311
JetF16	0.194	0.261	0.239	0.304
Sailboat	0.231	0.283	0.280	0.346
Zelda	0.221	0.265	0.273	0.320
Toys	0.152	0.221	0.199	0.252
Tiffany	0.160	0.209	0.183	0.253
GoldHill	0.272	0.306	0.323	0.370
Barbara	0.282	0.307	0.340	0.363
Average	0.230	0.277	0.278	0.330

Table 10 Comparison of *PBR*s between Lin et al.'s scheme [16] and the proposed scheme for SOL = 4, $M = 8$, $(\ell, t) = (2, 8)$, $(3, 9)$, and $M = N = 512$, 1024

N	512		1024	
Images	Proposed Scheme $M = N = 512$ $(\ell, t) = (2, 8)$	Lin et al. [16]	Proposed Scheme $M = N = 1024$ $(\ell, t) = (3, 9)$	Lin et al. [16]
Lena	0.326	0.389	0.391	0.466
Peppers	0.317	0.380	0.369	0.447
Baboon	0.487	0.551	0.547	0.634
Boats	0.312	0.366	0.367	0.418
JetF16	0.311	0.380	0.352	0.421
Sailboat	0.356	0.422	0.416	0.495
Zelda	0.323	0.373	0.388	0.455
Toys	0.242	0.280	0.291	0.319
Tiffany	0.251	0.303	0.293	0.339
GoldHill	0.377	0.442	0.434	0.525
Barbara	0.401	0.429	0.458	0.496
Average	0.337	0.392	0.391	0.456

Lin et al. [16] defines the per-block output length ψ to control their embedding capacity. We selected the optimal value of ψ that yields the closest average bit rate to conventional VQ (i.e., 0.438, 0.500, 0.563, and 0.625 bpp for main codebook sizes $N = 128$, 256, 512, and 1024, respectively). The selected values of ψ are 7, 8, 8, and 9 for codebook sizes $N = 128$, 256, 512, and 1024, respectively. The complete data used for this selection can be obtained from Table A7 in (http://tinyurl.com/jmby5nq).

In order to fairly compare with Lin et al.'s scheme [16], we modified our proposed scheme to conceal the same number of secret bits as their scheme. Table 6 demonstrates that, in all test cases, the proposed scheme attains lower average bit rates, indicating that it also obtains higher average *EE*s. Therefore, we can deduce that the proposed scheme improves the compression performance of Lin et al.'s scheme [16]. The experimental results with regard to the bit rate of Lin et al.'s scheme [16] and that of the proposed approach for each test image shown in Fig. 7 can be seen in Tables A8 − A11 in (http://tinyurl.com/jmby5nq).

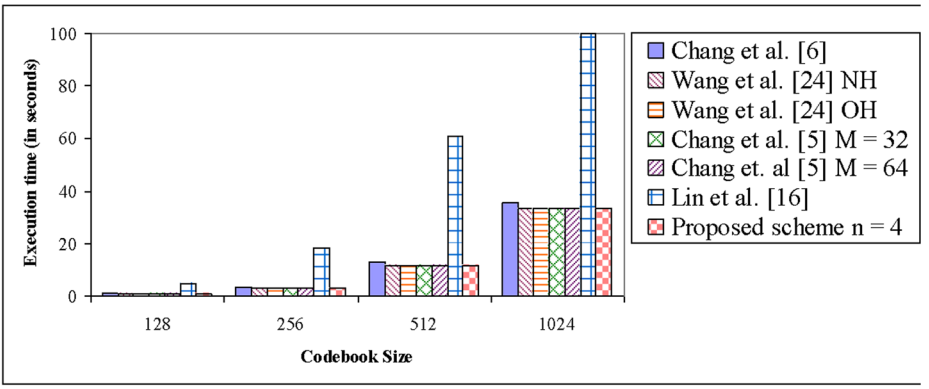
Chang et al.'s scheme [5] uses either SMVQ or 2-bit search order codes (i.e., 00, 01, and 10) to compress an SMVQ index (i.e., use fewer than $\lceil \log_2 M \rceil$ bits to represent an SMVQ index). In their scheme, the size M of the state codebook is either 32 or 64, which is smaller than the size N of the main codebook (e.g., $N = 256$). This reduces the visual quality of the recovered image since the optimal codeword may not be within a state codebook of size $M = 32$ or 64. Their embedding rate is restricted to 0.984 bpi because one secret bit is embedded into each residual SMVQ index (i.e., SMVQ indices not in the first row or column of the SMVQ index table).

For a fair comparison with Chang et al.'s scheme [5], the proposed scheme embeds 16,129 secret bits (i.e., $ER = 0.984$ bpi). The experimental results in Table 6 illustrate that the proposed scheme has lower average *BR*s than Chang et al.'s scheme [5] in all test cases with $M = 64$ and $N = 128$, 256, 512, and 1024. Similarly, in all test cases with $M = 32$ and $N = 128$ and 256, the proposed scheme also obtains lower average *BR*s than Chang et al.'s method [5]. However, with $M = 32$ and $N = 1024$, Chang et al.'s approach [5] attains lower average *BR*s than the proposed scheme for the test image Baboon (see detailed data in Table A15 in (http://tinyurl.com/jmby5nq)). Although the state codebook size is small (i.e., $M = 32$), with the test image Baboon, the values of Cases 1 and 2 SMVQ indices tend to be greater than 7. Additionally, the

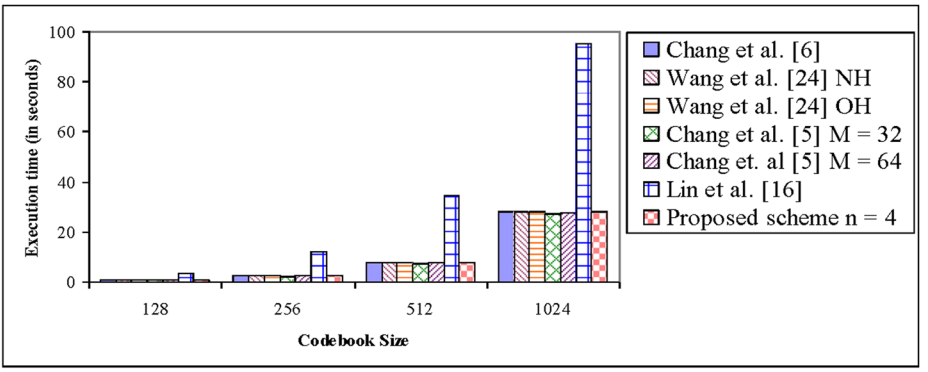
small state codebook size (i.e., $M = 32$) increases the efficiency of the SOC algorithm, causing Chang et al.'s scheme [5] to generate the smaller output code stream. This is why our *BR*s are higher than those of Chang et al.'s method [5] for the test image Baboon with $M = 32$ and $N = 1024$. Despite this, our average *BR*s are lower than those of Chang et al.'s method [5] with $M = 32$ and 64 and $N = 128$, 256, 512, and 1024. Therefore, we can infer that our proposed scheme improves the compression performance of Chang et al. [5]. The detailed experiments with regard to the bit rate of Chang et al.'s scheme [5] and that of the proposed scheme for each test image shown in Fig. 7 can be read in Tables A12 − A15 in (http://tinyurl.com/jmby5nq).

For the visual quality of the recovered images obtained by the simulated methods, Table 6 shows that the average *PSNR* values of the proposed scheme are the same as those of the works proposed by Chang et al. [6], Lin et al. [16], and Wang et al. [24]. Chang et al.'s scheme [5] has the lowest average *PSNR* values among the implemented schemes. To sum up, it can be observed in Table 6 that the proposed scheme with $n = 1$ achieves the same average *ER* as Chang et al. [6] and better average *BR* and *EE*. Similarly, with a fixed *ER* of 0.984 bpi, the proposed scheme obtains better average *BR* and *EE* than Chang et al.'s scheme [5] with $M = 32$ and 64. To fairly compare with Wang et al.'s variable bit embedding strategies [24] (i.e., normal hiding [NH] and over hiding [OH]), we modified the proposed scheme to embed the same number of secret bits as Wang et al.'s approach [24]. Compared with Wang et al.'s scheme [24], for the same average *ER*, the proposed scheme obtains the better average *BR* and *EE*. In contrast, Lin et al.'s scheme [16] achieves a slightly better average *ER* than the proposed scheme with $n = 3$ for main codebook size $N = 256$. However, the proposed scheme achieves the better average *BR* and *EE* for all codebook sizes. Finally, with $n = 4$, the proposed scheme improves the average *ER* of all the related works. Therefore, based on the analysis of the experimental results presented in this section, we can conclude that our proposed scheme improves the performance of the related works [5, 6, 16, 24].

For computational complexity analysis, we determine that the basic operation for each simulated scheme is the comparison operation between two pixel blocks X's of size $p \times q$ since each scheme requires many comparisons to build state codebooks, search for VQ and/or SMVQ indices and generate compression codes. In our implementation, we used the mergesort algorithm to build a state codebook of size M. For each input pixel block X, Chang et al.'s [5], Wang et al.'s [24] and the proposed scheme's first build a sate codebook (i.e., $N \log_2 N$ comparisons) and then search the state codebook for the SMVQ index x (i.e., M comparisons). After x is located, each scheme does a small amount of work before generating a compression code for x. Therefore, the main computational complexity of these three schemes is $M + N \times \log_2 N$ times the comparison operation. In contrast, for each X, Chang et al.'s scheme [6] first searches for a VQ index r (i.e., N comparisons), builds a state codebook (i.e., $N \log_2 N$ comparisons) and then searches the state codebook for the TIV value d (i.e., M comparisons). Therefore, their main computational complexity is $N + N \times \log_2 N + M$ times the comparison operation, which is worse than Chang et al.'s [5], Wang et al.'s [24], and the proposed schemes. Finally, the computational complexity of Lin et al.'s scheme [16] is the worst. This is because, for each X, they search for a VQ index r (i.e., N comparisons), then if r is not found in the SOC path, they build 2^v state codebooks (i.e., $2^v \times N \times \log_2 N$ comparisons), and then search these state codebooks for r (i.e., K comparisons where $1 \le K \le 2^v \times M$). However, if r is found in the SOC path, they simply generate the compression code for r. To simplify our analysis of Lin et al.'s scheme [16], we ignore K and assume that the probability that r is not found in the SOC path, P_N, is 0.306, 0.446, 0.567 and 0.654 for $N = 128$, 256, 512, and 1024, respectively. These probabilities were generated from the test images by counting the number



(a) Average encoding and embedding times



(b) Average decoding and extraction times

Fig. 8 Average execution time (in seconds) for all schemes and codebook sizes

of times r is found in the SOC path divided by the total number of indices. Therefore, the main computational complexity of Lin et al.'s method [16] is $N + P_N \times 2^v \times N \times \log_2 N$ times the comparison operation. With the optimal SOL of 4 in Lin et al.'s scheme (i.e., $v = 2$), their computational complexity is worse than all other schemes. Fig. 8 compares the average execution time for 10 times of each simulated scheme (i.e., 10 times for 11 test images). As expected, Lin et al.'s scheme [16] is the slowest while Chang et al.'s [5], Wang et al. [24], and the proposed schemes achieve similar execution times. Chang et al.'s scheme [6] is marginally slower than Chang et al.'s scheme [5], Wang et al.'s [24] and the proposed schemes but faster than Lin et al.'s scheme [16]. In general, the decoding and extraction algorithm of each scheme is faster than its respective encoding and embedding algorithm because the decoding and extraction algorithm does not need to compute VQ or SMVQ indices. Instead these indices are decoded directly from the received binary code stream.

5 Conclusions

This paper presents a novel reversible data hiding scheme by exploiting the *SMD*. The proposed scheme classifies SMVQ indices as Case 1 or 2 based on the values of the first state codeword's *SMD* and a predefined threshold t. By using this classification as an indicator, the proposed

scheme can switch between compression codes designed to compress Cases 1 and 2 SMVQ indices. This strategy achieves low average *PBR*s (i.e., the bit rate without embedding secret bits) of 0.230, 0.278, 0.337, and 0.391 bpp for the main codebook sizes $N = 128$, 256, 512, and 1024, respectively. The proposed method attains the embedding rates of 1, 2, 3, and 4 bpi at acceptable bit rates for main codebook sizes $N = 128$, 256, 512, and 1024. Additionally, for the same embedding capacities, the proposed scheme obtains lower average bit rates than recent VQ and SMVQ-based information hiding schemes [5, 6, 16, 24]. Therefore, we conclude that the proposed scheme improves the performance of these related works and is applicable for secret online communication.

References

1. Bentley JL, Sleator DD, Tarjan RE, Wei VK (1986) A locally adaptive data compression scheme. Commun ACM 29(4):320–330
2. Chang CC, Chen GM, Lin MH (2004) Information hiding based on search-order coding for VQ indices. Pattern Recogn Lett 25:1253–1261
3. Chang CC, Kieu TD, Chou YC (2009) Reversible information hiding for VQ indices based on locally adaptive coding. J Vis Commun Image R 20:57–64
4. Chang CC, Kieu TD, Wu WC (2009) A lossless data embedding technique by joint neighbouring coding. Pattern Recogn 42:1597–1603
5. Chang CC, Nguyen TS, Lin CC (2015) A reversible compression code hiding using SOC and SMVQ indices. Inf Sci 300:85–99
6. Chang CC, Nguyen TS, Lin CC (2013) A novel VQ-based reversible data hiding scheme by using hybrid encoding strategies. J Syst Softw 86:389–402
7. Chang CC, Nguyen TS, Lin CC (2011) A reversible data hiding scheme for VQ indices using locally adaptive coding. J Vis Commun Image R. 22:664–672
8. Davis RM (1978) The data encryption standard in perspective. IEEE Commun Soc Mag 16(6):5–9
9. Gray RM (1984) Vector quantization. IEEE ASSP Mag 1(2):4–29
10. Huffman DA (1952) A method for the construction of minimum-redundancy codes. Process IRE 40(9): 1098–1101
11. Hsieh CH, Tsai JC (1996) Lossless compression of VQ index with search-order coding. IEEE Trans Image Process 5(11):1579–1582
12. Kieu TD, Ramroach S (2015) A reversible steganographic scheme for VQ indices based on joint neighboring coding. Expert Syst Appl 42:713–722
13. Kim T (1992) Side match and overlap match vector quantizers for images. IEEE Trans Image Process 1(2): 170–185
14. Lee JD, Chiou YH, Guo JM (2013) Lossless data hiding for VQ indices based on neighboring correlation. Inf Sci 221:419–438
15. Lee CC, Ku WH, Huang SY (2009) A new steganographic scheme based on vector quantisation and search-order coding. IET Image Process 3(4):243–248
16. Lin CC, Liu XL, Yuan SM (2015) Reversible data hiding for VQ-compressed images based on search-order coding and state-codebook mapping. Inf Sci 293:314–326
17. Linde Y, Buzo A, Gray RM (1980) An algorithm for vector quantizer design. IEEE Trans Commun 28(1):84–95
18. Pan Z, Ma X, Deng X (2014) New reversible full-embeddable information hiding method for vector quantisation indices based on locally adaptive complete coding list. IET Image Process 9(1):22–30
19. Pan Z, Ma X, Deng X, Hu S (2013) Low bit-rate information hiding method based on search-order-coding technique. J Syst Softw 86:2863–2869
20. Qin C, Chang CC, Chiu YP (2014) A novel joint data-hiding and compression scheme based on SMVQ and image inpainting. IEEE Trans Image Process 23(3):969–978
21. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2):120–126
22. Wang WJ, Huang CT, Liu CM, Su PC, Wang SJ (2013) Data embedding for vector quantization image processing on the basis of adjoining state-codebook mapping. Inf Sci 246:69–82
23. Wang JX, Lu ZM (2009) A path optional lossless data hiding scheme based on VQ joint neighbouring coding. Inf Sci 179:3332–3348

24. Wang L, Pan Z, Ma X, Hu S (2014) A novel high-performance reversible data hiding scheme using SMVQ and improved locally adaptive coding method. J Vis Commun Image R 25:454–465
25. Yang CH, Lin YC (2010) Fractal curves to improve the reversible data embedding for VQ-indexes based on locally adaptive coding. J Vis Commun Image R 21:334–342
26. Yang CH, Wu SC, Huang SC, Lin YK (2011) Huffman-code strategies to improve MFCVQ-based reversible data hiding for VQ indexes. J Syst Softw 84:338–396

Kris Manohar received the B.S. degree in Computer Science in 2009 and M.S. degree in Computer Science in 2012 from The University of the West Indies, St. Augustine, Trinidad and Tobago. Currently, he is a Ph.D. candidate in Computer Science at The University of the West Indies, St. Augustine, Trinidad and Tobago. Since 2010, he has been with the Department of Computing and Information Technology, Faculty of Science and Technology, The University of the West Indies, St. Augustine, Trinidad and Tobago, where he is currently a teaching assistant. His research interests include information hiding, data compression, and image processing.

The Duc Kieu received the B.S. degree in Mathematics from the University of Pedagogy, Vietnam, in 1995, the B.S. degree in Information Technology from the University of Natural Sciences, Vietnam, in 1999, the M.S. degree in Computer Science from Latrobe University, Australia, in 2005, and the Ph.D. degree in Computer Science from Feng Chia University, Taiwan, in 2009. Since 2010, he has been with the Department of Computing and Information Technology, Faculty of Science and Technology, The University of the West Indies, St. Augustine, Trinidad and Tobago, where he is currently a Lecturer. His research interests include information hiding, data compression, and image processing.