Journal of Information Hiding and Multimedia Signal Processing ©2020 ISSN 2073-4212
Ubiquitous International Volume **11**, Number **1**, March **2020**

Improving Data Hiding Capacity in Code Based Steganography using Multiple Embedding

Katandawa Alex Kingsley, Ari Moesriami Barmawi

Graduate School of Informatics, School of Computing,
Telkom University
Jalan Telekomunikasi No.1, Bandung 40257, Indonesia
kingsley.students@telkomunivesity.ac.id, mbarmawi@melsa.net.id

Received July 2019; revised September 2019

Abstract. *In this modern era there is rapid increase in use of internet to exchange sensitive information. However, communication via the internet is unsecure and unrealible. Due to these factors, data hiding techniques has been proposed to increase the confidentiality and security of sensitive information. Moreover, Crandall[14] introduced Code Based Steganography, which merges steganography with coding theory. It implemented matrix encoding using linear codes to increase the visual quality of stego image by preserving high embedding capacity. Molaei et al[3] proposed a steganography scheme which implemented Reed Muller codes and modulus function in attempt to increase embedding capacity. These fault tolerant schemes have ability to recover secret messages from attacks using error detection and correction. However, existing schemes have low embedding capacity (150%) and low PSNR value (48dB). To overcome this problem, this paper proposed a multiple embedding method that aims to re-embed secrets bits on the same LSBs of the selected pixels based on a secret key. The experiments results shows that the proposed method achieved higher embedding capacity (450%) three times more than Molaei's method. The proposed method obtained a higher PSNR value of 51dB and higher error correction capability.*

Keywords: Data hiding, Reed Muller codes, Secret Sharing, Re-embedding, Embedding Capacity

1. **Introduction.** In this modern era the protection of digital data for confidentiality has increasingly emerged as a major concern and top priority. Data hiding is therefore a method of facilitating covert communication in the form of concealing information in a host media called a cover and then being able to extract the message from the cover[6]. The secret data that are applicable include text, images, videos and audio and so as is the cover media. There are two main types of data hiding techniques used for protecting sensitive data from unauthorised access and or tampering these are Watermarking and Steganography[8].

In Watermarking, the hidden or embedded data has a relation with the host or cover media. In Steganography the concealed secret data (e.g text, image etc) and the cover media (e.g text, image, audio, video etc) have no relation, the secret data hidden in stego image is required to be undetectable. Coding Theory is responsible for recovering messages sent via a noisy channel. It extends the message by adding redundant bits which enables error detection and correction[10]. Currently in Steganography, Coding theory serves a major part. Its main objective is to recover hidden messages from the attacked stego images.

A good steganography scheme consist of three characteristics: high embedding capacity, high embedding efficiency and security[19]. The embedding capacity describes the size of hidden secret message that can be transmitted. Embedding efficiency refers to the visual quality of the scheme and less image distortions are more secure because it does not raise any suspicion to adversaries. Highly secure and robust steganographic schemes have ability to resist against attacks. Current code based steganography scheme proposed by Molaei[3] have low embedding capacity (150%) and low embedding efficiency of PSNR value of 48dB. This is due to the adopted LSB embedding method of higher order significant bits of the cover pixels. The proposed encoder with code rate of $\frac{1}{2}$, extended the secret data by 2. The number of cover pixels to be embedded is increased by 2 also and because of that, the image quality reduces significantly. Therefore an introduction of data hiding techniques that improves the embedding capacity and at the same time assuring invisibility of embedded data are preferable.

This paper proposes a high capacity data hiding mechanism for gray scale images that maintains the invisibility of embedded data. It is based on multiple embedding of secret data into few randomly selected pixels to solve the low embedding capacity problem. The proposed method randomly selects a maximum of $\frac{3}{4}$ of the cover image pixels for multiple embedding to guarantee a highly imperceptible stego image. The secret data is encoded using Reed Muller error correction codes before embedding process to increase the robustness of the scheme.

A cryptographically secure pseudo-random number generator(CSPRNG) called Blum Blum shub [13] was adopted to randomly select cover pixel to be embedded. The seed used to initialize the CSPRNG and the number of embedding cycles was communicated between sender and receiver using a (2,2) secret sharing scheme[16]. This increases the security of the hidden data. The senders' secret share is encoded with Reed Muller codes before embedding into the stego image to increase its resistance against attacks. Furthermore, the Modulus function proposed in [5] and LSB embedding were adopted during embedding process to ensure good visual quality stego images. Experiment results shows that proposed method benefits the scheme by preventing burst errors which are common in existing methods. The proposed method provides high PSNR, high embedding capacity (450%) and significant error correction capabilities. The error correction capability of the proposed scheme increases as the size of the secret image increases.

2. Reed Muller Codes.

In this section the Reed Muller error correction codes are discussed. In Coding theory, error correction codes is a function that expresses numbers in a sequence such that error detection and correction is performed based on the remaining numbers after introduction of errors[15]. There are two categories of Error correction codes and these are Block and Convolution codes. Block codes encodes a message block of k bits to generate N fixed output data bits, where $N \geq k$. In this proposed method, a linear block code called Reed Muller codes are employed because of their simplicity in encoding messages and decoding received data.

Definition 2.1. *Based on [15], m and r are two positive integers and $0 \leq r \leq m$. An r^{th}-order Reed-Muller code, RM(r, m) has a message size, k, a code length, N and minimum distance, d_{min}, where*

$$k = \sum_{i=0}^{r} \binom{m}{i} \tag{1}$$

$$N = 2^m \tag{2}$$

$$d_{min} = 2^{m-r} \tag{3}$$

and the notation is written as:

$$\left[2^m,\quad k,\quad 2^{m-r}\right] - code$$

The Reed Muller code can correct a maximum number of errors t given by

$$t = \left\lfloor \frac{2^{m-r}-1}{2} \right\rfloor \tag{4}$$

The choice of m and r values determines the message size of k of the Encoder, the block size of N, minimum distance of the code and these have impact on the error correction capability of the code.

For example, given $m = 4$ and $r = 1$, then $k = 5$, $N = 16$, $d_{min} = 8$ and $t = 3$.

2.1. **Construction of Reed Muller Codes.** There are several ways that can be used to construct Reed Muller codes. In this study we define Reed Muller codes recursively. To define Reed Muller codes inductively, let the 0^{th} order, RM(0, m) defined by two vectors 0,1 over Galois field, F_2^N that are repetition codes[4]. For $0 \leq r \leq m$, the r^{th} order Reed-Muller code R(r,m) is defined recursively by

$$RM(r,m) = (u, u+v) : u \in RM(r, m-1), v \in RM(r-1, m-1) \tag{5}$$

where u and v represent the code words of the previously constructed codes. For example, if $r = 1$ and $m = 2$ i.e RM(1,2), the $(u,\ u+v)$ construction results in: $\mathbf{u} \in$ RM(1,1) and $v \in$ RM(0,1). Note that RM(1,1) = 00 01 10 11 and $u_0 = \mathbf{00}$, $u_1 = \mathbf{01}$, $u_2 = \mathbf{10}$, $u_3 = \mathbf{11}$. Also note that RM(0,1) = 00 11 and $v_0 = 00$, $v_1 = 11$. Therefore the resulting code of RM(1,2) is:

$$RM(1,2) = \left\{ \begin{array}{cccc} \mathbf{00}00 & \mathbf{01}00 & \mathbf{10}00 & \mathbf{11}00 \\ \mathbf{00}11 & \mathbf{01}11 & \mathbf{10}11 & \mathbf{11}11 \end{array} \right\}$$

2.2. **Encoding Reed Muller Codes.** This section describes how a binary message is encoded using Reed Muller Encoder. Firstly, determine the dimensions of the Generator matrix $G_{r,m}$ of an r^{th}-order Reed Muller code RM(r, m) with block length of 2^m. The Generator matrix of RM(r, m) is expressed in [4] as follows:

$$G_{r,m} = \begin{bmatrix} G_{r,m-1} & G_{r,m-1} \\ \mathbf{0} & G_{r-1,m-1} \end{bmatrix} \tag{6}$$

where $G_{0,m}$ is a vector of length 2^m with all ones and $G_{m,m}$ is an identity matrix $\mathbf{I}_{2^m}$.

To encode a message, $u \in F_2^k$, a binary multiplication of the message and the Generator matrix is performed to form a code word, $c \in F_2^N$ using equation (7) in [4]:

$$c = u * G_{r,m} \tag{7}$$

For example, assuming a message $u = (u_0, u_1, u_2, u_3, u_4) = (10101)$ is encoded using RM(1,4), the Generator Matrix $G_{1,4}$ is calculated using equation (6)

$$G_{1,4} = \begin{bmatrix} G_{1,3} & G_{1,3} \\ \mathbf{0} & G_{0,3} \end{bmatrix} = \begin{bmatrix} 0000000011111111 \\ 0000111100001111 \\ 0011001100110011 \\ 0101010101010101 \\ 1111111111111111 \end{bmatrix}$$

The code word $c = (c_1, c_1, ..., c_{15}, c_{16}) \in RM(1,4)$ is then calculated using equation (7):

$$(c_1, c_1, ..., c_{15}, c_{16}) = [u_4, u_3, u_2, u_1, u_0] * \begin{bmatrix} 0000000011111111 \\ 0000111100001111 \\ 0011001100110011 \\ 0101010101010101 \\ 1111111111111111 \end{bmatrix}$$

$$c = u * G_{r,m} = [10100] * \begin{bmatrix} 0000000011111111 \\ 0000111100001111 \\ 0011001100110011 \\ 0101010101010101 \\ 1111111111111111 \end{bmatrix} = [0000111001101011]$$

2.3. **Decoding Reed Muller Codes.** There are different techniques used for decoding Reed-Muller codes, the most common and easily implementable is majority logic decoding. The decoding process converts a code word c to message u. The process consist of 2 steps which are majority logic decoding and converting c to u, these steps are described in [17] as follows:

1. Majority decoding: This process is responsible for error correcting of the received vector[17]. To perform majority decoding, first consider the code word, c formed from encoding an input message, $u = (u_0, u_1, ..., u_{k-1})$ represented in [17] as:

$$\begin{aligned} c = (c_0, c_1, ..., c_{N-1}) = u_0 v_0 + \sum_{1 \le i_1 \le m} u_{i_1} v_{i_1} + \sum_{1 \le i_1 \le i_2 \le m} u_{i_1} u_{i_1} v_{i_1} v_{i_1} \\ +... + \sum_{1 \le i_1 \le i_2 \le ... \le i_r \le m} u_{i_1, i_2, ..., i_r} v_{i_1} v_{i_1}, ..., v_{i_r} \end{aligned} \quad (8)$$

if $x = (x_0, x_1, ..., x_{N-1})$ is the received vector, then there are r + 1 stages of the decoding process. For $1 \le i_1 \le i_2 \le ... \le i_{r-l} \le m$ where $0 \le l \le r$, the following index set described in [17], is formed as:

$$S = \{a_{i_1-1} 2^{i_1-1} + a_{i_2-1} 2^{i_2-1} + ...a_{i_{r-l}-1} 2^{i_{r-l}-1} : a_{i_j-1} \in 0, 1 for 1 \le j \le r-l\} \quad (9)$$

S is a set of 2^{r-l} non negative integers that are less than 2^m. Assuming E is a set of integers $0, 1, ..., m-1$ not in $i_1 - 1, i_2 - 1, ..., i_{r-l} - 1$, such that

$$E = \{0, 1, ..., m-1\} \backslash \{i_1 - 1, i_2 - 1, i_{r-l} - 1\} = \{j_1, j_2, ..., j_{m-r+l}\} \quad (10)$$

where $0 \le j_1 \le j_2 \le ... \le j_{m-r+l} \le m-1$. Then a set of integers S^c in [17] is formed as

$$S^c = \{d_{j_1} 2^{j_1} + d_{j_2} 2^{j_2} + ...d_{j_{m-r+l}-1} 2^{j_{m-r+l}} : d_{j_t} \in 0, 1 for 1 \le t \le m-r+l\} \quad (11)$$

And then the set of indices, B is formed for each q $\in S^c$ such that:

$$B = q + S = \{q + s : s \in S\} \quad (12)$$

Then the following equation, (14) will determine the decision equations in l-th decoding stage:

$$A^{(l)} = \sum_{t \in B} x_t^{(l)} \tag{13}$$

Relation (13) consist of 2^{m-r+l} equations at each stage and follows a binary addition rule. The input message $u_{i_1 i_2 \ldots i_{r-l}}$ is decoded as $u^*_{i_1 i_2 \ldots i_{r-l}} = 0$ if the decision equations results in zero. And it is decoded as $u^*_{i_1 i_2 \ldots i_{r-l}} = 1$ if the decision equations results in one. When decoding process has finished l stages, a modified received vector is formed as follows:

$$x^{(l)} = x^{(l-1)} - \sum_{1 \le i_1 \le \ldots \le i_{r-l+1} \le m} u^*_{i_1, i_2, \ldots, i_{r-l+1}} v_{i_1} v_{i_1}, \ldots, v_{i_{r-l+1}} \tag{14}$$

$x^{(l-1)}$ is the modified received vector in the l-th stage of decoding process and $x^{(0)} = x$. Then proceed to the next stage and repeat the above process until the end of the r + 1 stage to decode all received bits.

2. Converting r' to u: In this process, the modified received code word is converted to message vector. The following equation is used to determine the message vector:

$$u = r' * G^T_{r,m} \tag{15}$$

The following example illustrate the decoding process. From equation (8) a code word can be expressed in terms as (9) in [17]:
$(c_1, c_1, \ldots, c_{15}, c_{16}) = (u_0, u_1 + u_0, u_2 + u_0, u_2 + u_1 + u_0, u_3 + u_0, u_3 + u_1 + u_0, u_3 + u_2 + u_0, u_3 + u_2 + u_1 + u_0, u_4 + u_0, u_4 + u_1 + u_0, u_4 + u_2 + u_0, u_4 + u_2 + u_1 + u_0, u_4 + u_3 + u_0, u_4 + u_3 + u_1 + u_0, u_4 + u_3 + u_2 + u_0, u_4 + u_3 + u_2 + u_1 + u_0)$.

Using equation (11) in [10] to decide a set of indices of u and equation (12) to decide a set of indices of c for each corresponding u. Then the check sums for u_1, u_2, u_3 and u_4 are created using (14):
$u_1 = c_1 + c_2 = c_3 + c_4 = c_5 + c_6 = c_7 + c_8 = c_9 + c_{10} = c_{11} + c_{12} = c_{13} + c_{14} = c_{15} + c_{16}$
$u_2 = c_1 + c_3 = c_2 + c_4 = c_5 + c_7 = c_6 + c_8 = c_9 + c_{11} = c_{10} + c_{12} = c_{13} + c_{15} = c_{14} + c_{16}$
$u_3 = c_1 + c_5 = c_2 + c_6 = c_3 + c_7 = c_4 + c_8 = c_9 + c_{13} = c_{10} + c_{14} = c_{11} + c_{15} = c_{12} + c_{16}$
$u_4 = c_1 + c_9 = c_2 + c_{10} = c_3 + c_{11} = c_4 + c_12 = c_5 + c_{13} = c_6 + c_{14} = c_7 + c_{15} = c_8 + c_{16}$

So if the received code word is: [1 1 1 0 1 1 0 0 1 0 1 1 1 0 1 1], then the check sums for u_1, u_2, u_3 and u_4 calculated using equation (15) are
$u_1 : \{0, 1, 0, 0, 1, 0, 1, 0\}, u_2 : \{0, 1, 1, 1, 0, 1, 0, 1\}, u_3 : \{0, 0, 1, 0, 0, 0, 0, 0\}, u_4 : \{0, 1, 0, 1, 0, 1, 1, 1\}$

Considering the majority of each u_1, u_2, u_3 and u_4, we conclude that $u_1 = 0, u_2 = 1, u_3 = 0$ and $u_4 = 1$. To determine u_0, use equation (15) to calculate the modified received vector:

$$x = [1110110010111011] - [10100] * \begin{bmatrix} 0000000011111111 \\ 0000111100001111 \\ 0011001100110011 \\ 0101010101010101 \\ 1111111111111111 \end{bmatrix}$$
$$= [1110110010111011] - [0011001111001100] = [1101111101110111]$$

Using value of x, we decide that $u_0 = 1$. The corrected vector cc is calculated as:

$$cc = [0011001111001100] - [1111111111111111] = [1100110000110011]$$

The equation (15) is used to decode the code word cc, to form the message vector uu:

$$\begin{aligned} uu &= r * G_{1,4}^T \\ &= [0000111001101011] * \begin{bmatrix} 10000 \\ \cdots \\ 11001 \\ \cdots \\ 11111 \end{bmatrix} = [10101] \end{aligned}$$

3. **Molaei's Method.** In this section, Molaei's method is described. It consist of two phases: Embedding and Extraction Phase. The Embedding phase conceals a secret message into a cover image to form a stego image while the Extraction phase retrieve the secret message from the stego image. Molaei's method implements coding theory concepts, modulus function and steganography to improve embedding capacity and robustness against different kinds of noise attacks[3]. However, this method has low embedding efficiency (PSNR) and embedding capacity because it embeds secret data into the first and second LSBs of all pixels of the cover. It implements the LSB embedding using modulus function described in section (3.1).

3.1. **Modulus Function.** A steganographic technique to increase the visual quality of the stego image using modulus functions was proposed by Thien and Lin [5]. A modulus function is defined as $c = a \ mod \ b$, where a is the dividend, b is the divisor, and c is the remainder. For example, 5 mod 4 equals 1, that is, the division of 5 by 4 leaves a remainder of 1. Let x be a pixel used to hide data in the cover, z be the decimal value of the block/unit bits in the range 0 to $2^n - 1$. The embedding process of modulus function is employed to hide z_i unit in the i^{th} pixel of cover x_i. The difference dd_i between the two values is computed using the following equation in [5]:

$$dd_i = z_i - (x_i \bmod 2^n). \tag{16}$$

where n is the index of the low order bit of each pixel to be embedded. From dd_i the minimum variance is determined using the following equations in [5]:

$$dd_i' = \left\{ \begin{array}{ccc} dd_i & \text{if} & -\left\lfloor \frac{2^n-1}{2} \right\rfloor \le dd_i \le \left\lceil \frac{2^n-1}{2} \right\rceil \\ dd_i + 2^n & \text{if} & -2^n + 1 \le dd_i \le -\left\lfloor \frac{2^n-1}{2} \right\rfloor \\ dd_i - 2^n & \text{if} & \left\lceil \frac{2^n-1}{2} \right\rceil \le dd_i \le 2^n \end{array} \right\} \tag{17}$$

However, the value of dd_i' can exceed the range 0 to 255, so equation (18) in [5] is used determine the final pixel value x_i after embedding as follows:

$$x_i' = \left\{ \begin{array}{ccc} x_i + dd_i' & \text{if} & 0 \le x_i + dd_i' \le 255 \\ x_i + dd_i' + 2^n & \text{if} & x_i + dd_i' \le 0 \\ x_i + dd_i' - 2^n & \text{if} & x_i + dd_i' \ge 255 \end{array} \right\} \tag{18}$$

The extraction of the embedded data from the n order bits is done using the following equation in [5]:

$$z_i = x'_i \bmod 2^n \tag{19}$$

3.2. **Overview of Embedding phase.** The embedding phase is responsible for hiding secret data into cover media. It hides a secret binary data SD in a gray-scale cover image C of height H and width W and generates a stego image C'. The block diagram of the embedding process is shown in Fig 1.

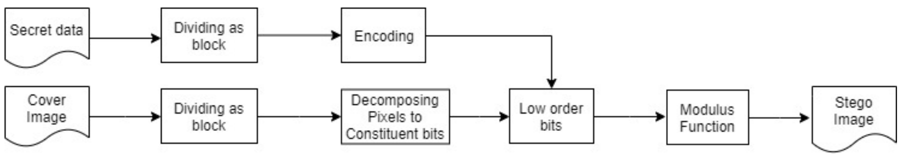


FIGURE 1. Embedding process of Molaei's Method

The embedding process is done using the following process:

1. **Dividing the pixels of I:** The secret data is divided into blocks that will enable easy encoding of the secret data. The first B bits are divided into nb_0 blocks $b = (b_0, b_1, ..., b_{k_0-1})$ of equal sizes equal to k_0, where $1 \leq B \leq \frac{H \text{ x } W}{2}$, k_0 is the message block size of the RM(1,3) code computed using equation (1). The number of blocks nb_0 to be embedded into first LSBs is calculated using equation (20):

$$nb_0 = \left\lceil \frac{B}{k_0} \right\rceil \tag{20}$$

   Considering $SD > B$, the remaining $(SD - B)$ bits are also divided into nb_1 blocks $bb = (bb_0, bb_1, ..., bb_{k_1-1})$ of sizes k_1, where $1 \leq (SD - B) \leq \frac{H \text{ x } W}{4}$, and k_1 is the message block size of RM(2,5) computed using equation (1). The number of blocks nb_1 is calculated using equation (21):

$$nb_1 = \left\lceil \frac{(SD - B)}{k_1} \right\rceil \tag{21}$$

2. **Encoding the secret blocks:** The blocks b and/or bb are encoded to enable error detection and correction to be performed from attacked stego images. Consider all nb_0 blocks, encode each block b using RM(1,3) code to form code word $c = (c_0, c_1, ..., c_{N_0-1})$ of size N_0, where N_0 is block size of RM(1,3) code calculated using equation (2). Consider all nb_1 blocks, encode each block bb using RM(2,5) codes to form code word $cc = (cc_0, cc_1, ..., cc_{N_1-1})$ of size N_1, where N_1 is block size of RM(2,5) code calculated using equation (2).
3. **LSB embedding using Modulus Function:** In this process the encoded secret blocks are hidden or embedded into the pixels of the cover image. The two sequence of code words c and cc are embedded into the first and second LSB of each cover pixel respectively. For each block of nb_0 blocks, consider the decimal value of each c_i to be embedded into each p_i and calculate the difference value dd_i using equation (16) where $n = 1$. Determine dd'_i using equation (17). The modified pixel value pp_i is calculated by equation (18).
4. Repeating procedure **3** to embed the nb_1 blocks of cc into sequence of modified pixel blocks pp where $n = 2$. The output of the embedding process are modified pixel values pp' which forms the stego image C'.

The equations (16), (17) and (18) were implemented to reduce the difference between the original pixel values and modified pixel values. This increases the visual quality of the stego. The embedding process only hides secret data in both first and second LSBs of the cover.

3.2.1. ***Determining the Embedding Capacity of Molaei's Method***. The maximum size of secret data D_{max} that can be embedded is calculated using equation (22) in [3]:

$$D_{max} = \sum_{i=0}^{n} \left\lfloor \frac{H * W}{2^m_i} \right\rfloor * k_i \tag{22}$$

where k is the message block size of RM(r, m) encoder and n is the number of LSBs used for embedding. The embedding capacity EC is then determined using equation (23):

$$P = \frac{D_{max}}{H * W} \tag{23}$$

3.3. **Overview of Extraction phase.** In this section the extraction process is discussed. The extraction phase aims to retrieve the hidden data successfully. The secret data SD is retrieved from the Stego image C'. The block diagram of the extraction process is shown in Fig 2.

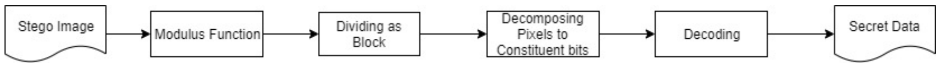


FIGURE 2. Extraction process of Molaei's Method

1. **Modulus Function**: The process aims to extract the secret encoded bits first. Equation (19) is used to extract the decimal values z_{ii} of the secret data, for $1 \leq ii \leq H$ x W and $n = 1$. Each decimal value in z_{ii} is converted to binary to form a sequence of extracted bits Ez.
2. **Dividing extracted data into blocks**: The extracted bits Ez is divided into blocks of size equal to block size of the RM(r, m) Decoder. The first F_b bits of Ez are divided into nc_0 blocks $c = (c_0, c_1, ..., c_{N_0-1})$ of equal sizes equal to N_0, where $1 \leq F_b \leq H$ x W, N_0 is the block size of the RM(1,3) code. The number of blocks nc_0 is calculated as follows:
$$nc_0 = \left\lceil \frac{B}{N_0} \right\rceil \tag{24}$$
3. **Decoding the extracted blocks**: The blocks are decoded to recover the secret data. For all the nc_0 blocks, each code word block c is decoded using RM(1,3) decoder to form a secret block $b = (b_0, b_1, ..., b_{k_0-1})$.
4. Increase n to extract data in second LSB and repeat step (1) to (3) using $1 \leq ii \leq \frac{H \text{ x } W}{2}$, number of blocks $nc_1 = \left\lceil \frac{H \text{ x } W}{2 * N_1} \right\rceil$, where N_1 is the block size of the RM(2,5) code. Decode using RM(2,5) decoder as described in section (2.3).
5. **Output:** The output of the process is a sequence of bits from the Decoder which represent the secret data.

4. **The Proposed Method.** This section discusses about a steganographic technique that offers high embedding capacity, high error correction capabilities and high visual quality. The proposed method is discussed in detail. It consist of two main processes, embedding process and extraction process. The proposed method introduced a PRNG called Blum Blum Shub to generate random sequence of integers that represent pixel positions of the Cover image. The secret message was embedded and extracted into and from pixels on these positions orderly. The seed of the PRNG was communicated using (2,2) Visual Cryptography scheme. The concepts of Blum Blum Shub and (2,2) Visual Cryptography scheme are discussed in [13] and [16] respectively.

4.1. **Overview of Embedding Process.** In this section, the embedding process is described. The process is responsible for embedding secret data securely. There are three input images to this process, the cover image CI, agreed image A and secret image SI. The Sender and Receiver agree on an image A before the embedding and extraction processes begins. Fig 3 shows how a secret message is embedded into a cover using an agreed image to generate Agreed and Stego image.

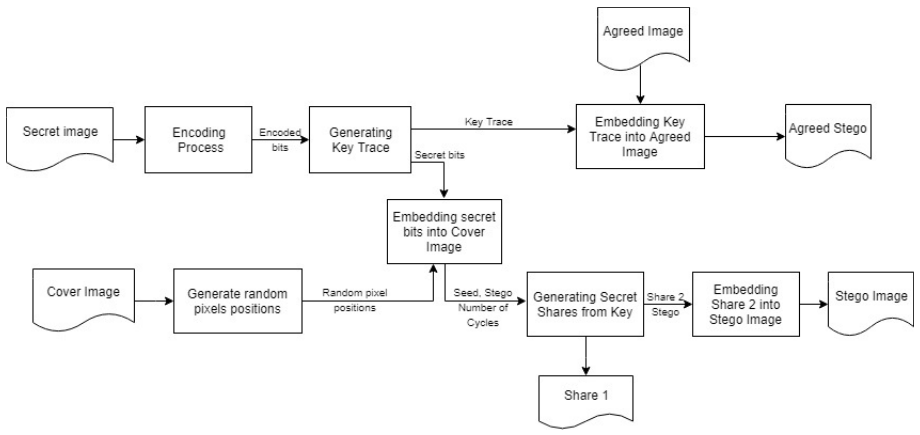

Agreed Image
Secret image
Encoding Process
Encoded bits
Generating Key Trace
Key Trace
Embedding Key Trace into Agreed Image
Agreed Stego
Secret bits
Embedding secret bits into Cover Image
Cover Image
Generate random pixels positions
Random pixel positions
Seed, Stego Number of Cycles
Generating Secret Shares from Key
Share 2 Stego
Embedding Share 2 into Stego Image
Stego Image
Share 1

Figure 3. Embedding Process of Proposed Method

The secret image pixels are first decomposed into bits and are encoded with Reed Muller error correction codes. First order Reed Muller RM (1, 4) codes have been introduced because of their high Error Correction Capacity, efficiency and relatively easy to decode. To increase the security of the secret data, a pseudorandom number generator (PRNG) was introduced to randomly generate pixels to be embedded from the cover. If the seed of the PRNG is not known then the hidden secret data cannot be extracted correctly.

The scheme implements LSB embedding using modulus function to embed the encoded secret data into $\frac{3}{4}$ of the cover image pixels. Only $\frac{3}{4}$ of the cover image pixels were used to increase the visual quality of the final Stego image. **In other words, if the size of the cover image is H * W, where H is the height and W is the width of the cover image then the number of pixels used to embed the secret image is $\frac{3}{4} * H * W$.** If the size of the encoded secret data exceeds $\frac{3}{4}$ of the cover image pixels, the data is re-embedded into the same randomly generated pixels using a key. The key contains Seed and number of cycles. The Key is securely distributed to the receiver using (2,2) VCS mentioned in [16]. This further increases the security of the embedded secret data.

The following definition discusses how the embedding capacity is determined using the proposed method.

Definition 4.1. *Given p pixels of a Cover image CI of height H and width W where $p \le \frac{3}{4} * H * W$. If n secret bits are re-embedded into each LSB of the p pixels, then the total secret bits S_T that can be embedded into the cover image CI is:*

$$S_T = p \ . \ (n+1) \tag{25}$$

and the Embedding Capacity EC expressed as a percentage is determined by

$$EC = \frac{S_T}{H * W} = \frac{p \ . \ (n+1)}{H * W} \tag{26}$$

For example, consider a Cover image of size $H = 512$ and $W = 512$. Assuming $n = 5$ secret bits are re-embedded on the LSBs of $p = \frac{3}{4} * H * W = \frac{3}{4} * 512 * 512 = 196608$ pixels. The total secret bits S_T that can be embedded into the cover image is calculated as:

$$S_T = 196608 \ . \ (5+1) = 1179648 \ bits \tag{27}$$

and the Embedding Capacity EC is calculated by

$$EC = \frac{1179648}{512 * 512} = 450\% \tag{28}$$

The following sections describes each sub process of the embedding process in detail as follows:

4.1.1. *Encoding Process.* In this process the encoding process is discussed. Reed Muller encoder discussed in section(2.3) is responsible for encoding secret data before embedding such that the secret data is recoverable using error correction if the stego image encounters attacks. The encoding process consist of the following sub processes as shown on Fig 4.

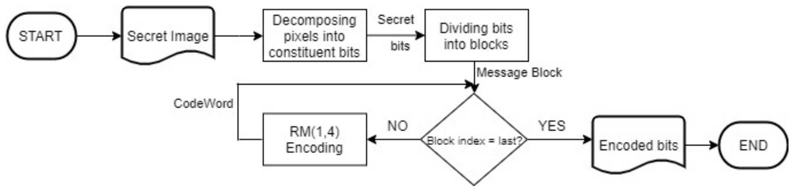


FIGURE 4. Encoding Process

The secret image SI with height H_0 and width W_0 is decomposed into bits and encoded by RM(1,4) Encoder to generate sequence of code words C using the following three sub-processes:

1. *Decomposing pixels into Constituent Bits*: The pixel values of the secret image are converted into bits because of the binary Reed Muller codes that has been introduced. Each pixel of SI (s_i, $1 \le i \le H_0 * W_0$) is converted to a byte. Each byte is broken down into bits and the output of the process is a sequence of all bits $sb = (sb_0, sb_1, ..., sb_Y)$ where $Y = 8 * H_0 * W_0$.
2. *Dividing bits into blocks*: The process divides secret bits into equal sized blocks as required by the Encoder such that the encoding process is more efficient. The input to the process is a sequence of secret bits sb and message length k_2 of the RM(r_1,m_1) encoder.

Definition 4.2. *Let $Y \nmid k_2$, and the secret bits sb being divided into n blocks $mb^{(q)} = (mb_0^{(q)}, mb_1^{(q)}, ..., mb_{k_2-1}^{(q)})$ of length k_2 where $n = \left\lceil \frac{Y}{k_2} \right\rceil$ and $(1 \le q \le n)$.*

The number of zero bits Z_n to be added to the n^{th} block are:

$$Z_n = Y \mod k_2 \tag{29}$$

For example, consider secret bits $sb = [1010111100110]$ are to be encoded by RM(1,4) encoder. Using equation (1), the message length $k_2 = 5$. The secret bits size $Y = 13$. Therefore $13 \nmid 5$. The number of blocks n is calculated as follows:

$$n = \left\lceil \frac{Y}{k_2} \right\rceil = \left\lceil \frac{13}{5} \right\rceil = 3$$

The number of zero bits Z_n added to 3^{th} block are determined by:

$$Z_n = (3 * 5) - 13 = 15 - 13 = 2. \tag{30}$$

Therefore the blocks to be encoded are : $mb^{(1)} = [10101]$, $mb^{(2)} = [11100]$ and $mb^{(3)} = [110\mathbf{00}]$

3. *RM (r_1,m_1) Encoding*: The Encoder is responsible for converting the secret message bits into code words which will enable error detection and error correction. The input to the process at each instance is a block $mb^{(q)}$. For all n blocks, encode each $mb^{(q)}$ using RM(r_1,m_1) error correction code using equation (7) to form codewords $cw^{(q)} = (cw_1^{(q)}, cw_2^{(q)}, ..., cw_{N_2}^{(q)})$ where $N_2 = 2^{m_2}$. Decompose the code words $cw^{(q)}$ into constituent bits, for $(1 \leq q \leq n)$. Therefore the output of the Encoder is a sequence of bits, $eb = eb_0, eb_1, ..., eb_{E_T}$ formed from decomposing all code words, where the total encoded bits size E_T after encoding process is: $E_T = \frac{8 * H_0 * W_0 * N_2}{k_2}$

4.1.2. *Generating Key Trace.* This segment discusses about how the Key Trace is generated. Key Trace are the sequence of bits generated from XORing the encoded bits and the chosen Reference Key bit value. The main purpose of this process is to increase the overall hiding capacity of the scheme by re-embedding secret bits on the pixels of the cover image. Fig 5 shows how the Key Trace bits are generated.

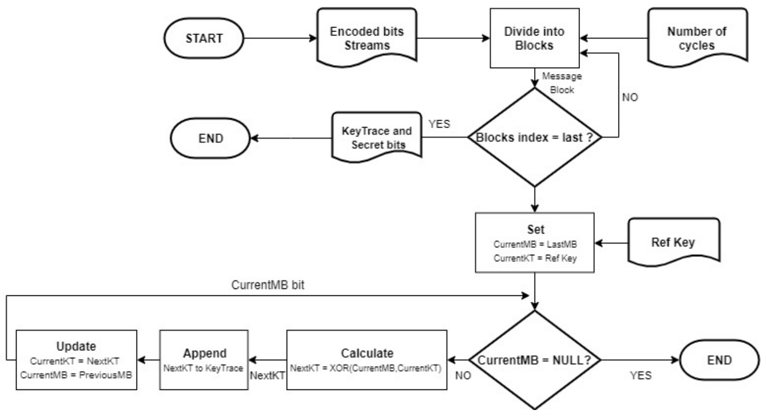


Figure 5. Key Trace Generation

This process is executed if and only if the size of the encoded bits E_T is greater than $\frac{3}{4} * H_1 * W_1$, where H_1 and W_1 are the height and width of the cover image I. The key trace K_T is generated from the encoded bits eb using the following 4 steps:

Step 1. Inputting sequence of encoded bits eb and Reference Key. A reference key R is any binary value of choice ie $R \in 1, 0$.
Step 2. Determining how many bits are to be embedded into each pixel of the cover, called number of cycles N_c defined as follows:

Theorem 4.1. *Given the total number of encoded bits E_T embedded on maximum capacity of single LSB which is equal to $\frac{3}{4} * H_1 * W_1$ (based on definition on page 12). The number* *of cycles N_c is:*

$$N_c = \frac{E_T}{\frac{3}{4} * H_1 * W_1} \tag{31}$$

Proof. The number of cycles is a function of the total number of encoded bits E_T. Since the total number of encoded bits E_T can be greater than the maximum capacity of single LSB $(\frac{3}{4} * H_1 * W_1)$. This results in more than 1 bit to be re-embedded into each pixel. Therefore, in the case $E_T \geq (\frac{3}{4} * H_1 * W_1)$, the number of cycles re-embedded into each pixel N_c is a equal to the number of encoded bits E_T divides by the maximum capacity of single LSB which is equal to $\frac{3}{4} * H_1 * W_1$. Thus it is proven that:

$$N_c = \frac{E_T}{\frac{3}{4} * H_1 * W_1}$$

□

Step 3. Dividing the encoded bits *eb* into block with the length of $\lceil N_c \rceil$ and the remainder is divided into length of $\lfloor N_c \rfloor$. The following theorem defines the number of pixels that can be re-embedded with $\lceil N_c \rceil$ bits h and he number of pixels that can be re-embedded with $\lfloor N_c \rfloor$ bits h'.

Theorem 4.2. *Suppose the number of secret bits eb and the number of cycles N_c is an integer which cannot divide eb, in other words $N_c \nmid eb$. Thus there are pixels which are re-embedded with length $\lceil N_c \rceil$ bits and others with $\lfloor N_c \rfloor$ bits. So the number of pixels that can be re-embedded with $\lceil N_c \rceil$ bits is:*

$$h = \frac{3 * H_1 * W_1 * (N_c - \lfloor N_c \rfloor)}{4} \tag{32}$$

and the number of pixels that can be re-embedded with $\lfloor N_c \rfloor$ bits is:

$$h' = \frac{3 * H_1 * W_1}{4} - h \tag{33}$$

Proof. From Theorem 4.1, each pixel from the $(\frac{3}{4} * H_1 * W_1)$ of the cover pixels is re-embedded with N_c cycles then it means that the number of bits that can be embedded is $(N_c * \frac{3}{4} * H_1 * W_1)$. Since N_c may not be an integer i.e $N_c \notin \mathbb{Z}$, there is a set of pixels that is re-embedded $\lceil N_c \rceil$ times and another $\lfloor N_c \rfloor$ times. In case of *db* where the size is $\lceil N_c \rceil$ then the number of pixels re-embedded with $\lceil N_c \rceil$ secret bits is equal to the difference between N_c and the float of N_c multiplied by the maximum capacity of single LSB $(\frac{3}{4} * H_1 * W_1)$. In other words, the number of pixels that can be re-embedded with $\lceil N_c \rceil$ bits is equal to:

$$h = \frac{3 * H_1 * W_1 * (N_c - \lfloor N_c \rfloor)}{4}$$

The number of pixels that can be re-embedded with $\lfloor N_c \rfloor$ bits is the difference between maximum capacity of single LSB and he number of pixels that can be re-embedded with $\lceil N_c \rceil$ bits h which is:

$$h' = \frac{3 * H_1 * W_1}{4} - h$$

□

Step 4. Considering each of the blocks db and db' , the Key Trace block CK and Cover embeddable bit E are determined using algorithm 1:

Algorithm 1: Generation of Key Trace block and Cover Embeddable Bit

```
   Input: Blocks $db$ or $db'$, Number of cycles $N_c$, Reference Key $R$
   Output: KeyTrace block $CK$ and Cover Embeddable bit $E$
1  $t \leftarrow N_C$
2  $CK_{t-1} \leftarrow db_t \oplus R$
3  while $t > 0$ do
4      if $t == 1$ then
5          $E \leftarrow CK_t \oplus db_t$
6          Append $E$ to $E_m$
7      else
8          $CK_{t-1} \leftarrow db_t \oplus CK_t$
9          $t \leftarrow t - 1$
```

For each block of size n, the process creates a key Trace block of size n - 1. The output are two sequences of bits,
a). Key Trace, K_T which is a decomposition of all blocks CK_i into constituent bits.

Definition 4.3. *Let N_c be the number of cycles per pixel as defined in Theorem (4.1). From Theorem 4.2, the total number of bits to be re-embedded on all pixels called Key Trace K_T is:*

$$|K_T| = (h. \lceil N_c \rceil) + (h'.N_c) \tag{34}$$

b). And E_m bits that are embedded into the cover image.

4.1.3. *Generating random pixel positions.* This segment discusses how the random pixel positions are generated. The process uses, Blum Blum Shub, a cryptography pseudo random number generator, PRNG described in [13] to generate random sequence of bits. The bits are grouped into 18-bit block and converted to integers between 1 and H_1 * W_1. The generated integers represent pixel positions on the cover image that are used for embedding secret bits E_m.

The secret bits are embedded according to the order of the sequence generated. If the seed of the PRNG is not known then the embedded pixel positions are not known and the secret message cannot be extracted correctly. This process was introduced to increase the security of the embedded secret message. To generate a sequence of random pixel positions, $pp = pp_1, pp_2, ..., pp_L$, the function inputs
a). Seed
b). Length L of pp where,

$$L = \left\{ \begin{array}{ccc} E_T & \text{if} & 1 \leq E_T \leq (\frac{3}{4} * H_1 * W_1) \\ \\ \mid E_m \mid & \text{if} & E_T \geq (\frac{3}{4} * H_1 * W_1) \\ \\ \end{array} \right\} \tag{35}$$

The output is a random sequence of integers values $pp = (pp_1, pp_2, ..., pp_L)$ where $1 \leq pp_i \leq H_1 * W_1$. For example: 120132 145 9987 97 103110 125 14285 92 117 197 203210.

4.1.4. *Embedding secret bits into Cover Image.* This segment discusses how the secret bits are embedded into the cover image. It is responsible for efficiently hiding the encoded

secret bits on randomly generated Cover Image pixels.
The process require two inputs :
a). The sequence of embedding bits EB, where:

$$EB = \left\{ \begin{array}{lll} eb & \text{if} & 1 \leq E_T \leq (\frac{3}{4} * H_1 * W_1) \\ \\ E_m & \text{if} & E_T \geq (\frac{3}{4} * H_1 * W_1) \\ \end{array} \right\} \tag{36}$$

b). Sequence of randomly generated integers pp

LSB embedding using modulus function described in section (3.1) was used to embed all bits of EB on Cover pixels of index pp. It is implemented because of its simplicity in embedding and extraction. The embedding process is done as follows:

Step 1: Considering the decimal value of each bit in EB to be embedded on each pixel px_i on index pp_i.
Step 2: For each EB_i and px_i with $n = 1$, calculating the difference value dd_i using equation (16). Determine dd_i' using equation (17). And lastly, the modified pixel value pp_i' is calculated by equation (18).
Step 3: Performing the same procedure for $1 \leq i \leq L$. The output of the embedding process are modified pixel values pp' which forms the stego image I'.

4.1.5. *Embedding Key Trace into Agreed Image.* In this segment, a sequence of Key Trace bits K_T are embedded into the image A of height H_2 and width W_2. The image A is a grayscale image that has been agreed upon between Sender and Receiver before communication of the secret message. To embed K_T the following process was performed:

1. Determining the number of LSBs nn to be embedded: The value of nn is calculated as follows:
$$nn = \frac{\mid K_T \mid}{H_2 \text{ x } W_2} \tag{37}$$
If nn is a not an integer, then there are two sets of block sizes,
   (a) The first $(\lceil nn \rceil * H_2 * W_2)$ bits are divided into blocks w of size $\lceil nn \rceil$, such that $w = (w_1, w_2, ..., w_{\lceil nn \rceil})$ and
   (b) The remaining $|K_T| - (\lceil n \rceil * H_2 * W_2)$ bits are divided into blocks w of size n, such that $w = (w_1, w_2, ..., w_n)$ where $n \geq 1$.

   But if nn is an integer, then all K_T bits are divided into blocks w of size nn, such that $w_i = (w_1, w_2, ..., w_{nn})$.

2. Performing LSB embedding using Modulus function: To embed the Key trace bits into the agreed image, the modulus function was applied to increase the visual quality of the resulting stego. The method reduces the difference between the original and final pixel values using equations (16) and (17). To embed the all blocks w, the following process was performed:

   (a) Considering the decimal value z of each block in w to be embedded into each pixel pa of image A. For the first $((nn - \lfloor nn \rfloor) * H_2 * W_2)$ decimal values we have $n = \lceil nn \rceil$, calculate the difference value dd_i using equation (16). Determine dd_i' using equation (17).The modified pixel value pa_i' is calculated by equation (18).
   (b) Performing the same procedure for the last $((H_2 * W_2) - (nn - \lfloor nn \rfloor) * H_2 * W_2)$ decimal values where $n = \lfloor nn \rfloor$. The output of the embedding process are modified pixel values pa' which forms the stego image A'.

4.1.6. *Generating Shares from Secret Key.* In this segment, the process is designed to securely communicate the seed, number of cycles N_c and height H_0 and width W_0 of the secret image. Fig 6 shows how the Key Trace bits are generated.

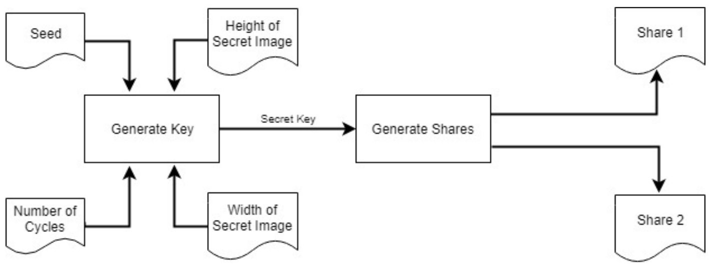


FIGURE 6. Share Generation

A Key in form of binary image is generated based on these 4 parameters. (2,2) visual cryptography was later implemented on the Key to generate shares because of its low complexity nature in encoding and decoding. The following generation of key and shares is described as follows:

1. *Generating Key:* The function of this component is to create a secret key during the embedding process to enable easy and successful extraction of the hidden secret image, SI. The process require 3 input texts:
   (a) *Seed, SE*: An integer that initialize the PRNG.
   (b) *Number of Cycles N_c*: It is the result after dividing the Total encoded secret bits by $\frac{3}{4}$ of the cover image.
   (c) Height H_0 and Width W_0 of the secret image

   The output of the process is a binary image consisting of SE, N_c, H_0 and W_0. The three input text are converted from text to binary image using the following process:
   (a) Concatenating SE, N_c, H_0 and W_0 separating each with a space character to form, Txt. Such that, $Txt = SE + N_c + H_0 + W_0$.
   (b) Representing each character in Txt by (0, 1)-matrix, T_g of dimensions m' x n' with 0s and 1s arranged to show the character Txt_g visually, where $1 \leq g \leq |Txt|$, $m' = 16$ and $n' = 14$.
   (c) Combining all T_i to form one binary matrix T' of size m'' x n'', where $m'' = m' * |Txt|$ and $n'' = n' * |Txt|$

   The binary matrix T' is referred to as the Secret Key.

2. *Generating Share:* (2, 2) visual cryptography is implemented on T' as discussed in [16] to increase the security of the secret key while requiring less computation time. The input to the process is a binary matrix, T' of size mm x nn. The process perform encryption of T' as described in [16] to generate 2 random binary matrices, T'_1 and T'_2 of sizes mm x $2nn$, referred to as Share 1 $S1$ and Share 2$S2$ respectively. $S1$ is distributed securely to the receiver while $S2$ is embedded into the stego image.

   For example, assuming the seed SE = 262139, Number of cycles N_c = 0.01, Height of secret image H_0 = 10 and Width of secret image W_0 = 6. The string Txt = 262139 0.01 10 6, and the binary image T' is shown in Fig 7. (2,2) Visual cryptography creates two shares $S1$ and $S2$ shown in Fig 8.

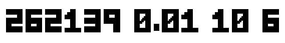

262139 0.01 10 6

FIGURE 7. Example of Key Generation

(A) Share 1

(B) Share 2

FIGURE 8. The generated Shares

4.1.7. *Embedding Share 2 into Stego Image.* In this segment, the embedding of Share 2 $S2$ is described. Firstly, $S2$ is encoded with RM(r_1,m_1) code to enable error correction of error bits. The encoded $S2$ is embedded into Stego I' such that the correct receiver, possessing $S1$, can extract $S2$, decode and be able to stack both shares to retrieve SE, N_c, H_0 and W_0. The embedding of $S2$ to generate a new stego image is done by the following process:

1. Inputting the Stego image I', T_2' and message length k_2 of RM(r_1,m_1) encoder.
2. Dividing $S2$ into linear blocks of size k_2: The binary matrix, T_2' is decomposed to single linear block of bits, $T_2'' = t_1'', t_2'', ..., t_{n''*2m''}''$. Divide T_2'' to nt blocks, $tb = (tb_0, tb_1, ..., tb_{k_1-1})$ of size k_1, where $nt = \left\lceil \frac{n''*2*m''}{k_1} \right\rceil$.
3. Encoding each tb with RM(r_1,m_1) error correction code to form code words $ct = (ct_1, ct_2, ..., ct_{N_1})$ where $N_1 = 2^{m_1}$.
4. Outputting is a sequence of bits CT of size $ET_T = \frac{n''*2*m''*N_1}{k_1}$ formed after decomposing all code words ct.
5. LSB embedding using modulus function: The method described in section (3.1) was used to embed all bits of CT into Stego I' pixels. The embedding process is performed as follows:

   (a) Considering the decimal value of each bit in CT to be embedded into each pixel pp'. For each CT_f and pp_f' with $n = 1$, calculate the difference value dd_f using equation (16). Determine dd_f' using equation (17). The modified pixel value pp_f'' is calculated by equation (18).
   (b) Performing the same procedure for $1 \leq f \leq ET_T$. The output of the embedding process are modified pixel values pp'' which forms the final stego image I''.

4.2. **Overview of Extraction Phase.** In this section the extraction process is discussed. The process aim to retrieve the embedded secret message. It requires a stego image, Share 1, share 2 and the Key Trace to successfully recover the secret message. Fig 9 shows how the extraction process is conducted. To begin the extraction process Share 1 is overlapped/stacked with Share 2 to construct the Key. Once the key is successfully constructed the Seed and the number of cycles becomes visible i.e the stacked image will show the seed and Number of Cycles. The pseudo random number generator with the correct seed produces integers that represent positions on the stego to extract the secret message using the key trace. Each of the sub processes of the extraction process is discussed in detail as follows:

4.2.1. *Generating Key.* This segment discusses how the key image is regenerated using (2,2) VCS. To extract a secret message successfully, a key should be reconstructed first because it possesses the seed and number of cycles used in embedding process.To reconstruct the key we overlap the extracted Share 2 and Share 1 S1 that the receiver already possess. To extract $S2$ from input Stego image I'', the following process is performed:

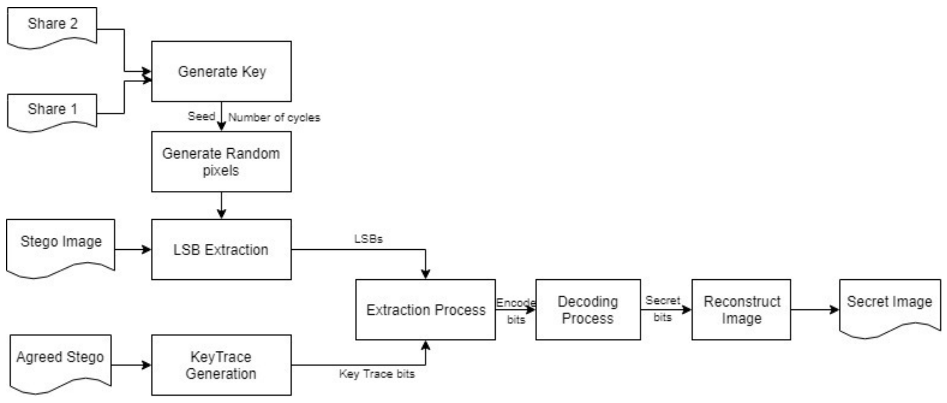


FIGURE 9. Extraction Process of Proposed Method

1. Extracting encoded Share 2: To extract $S2$ consider all stego pixels, $sp = sp_q, sp_{q+1}, ..., sp_G$ where q is the marker for starting point of embedded $S2$ and G $=H_1 * W_1$. Apply equation (19) to every sp to form a sequence of integers $sl = (sl_1, sl_2, ..., sl_E)$ where $n = 1$ and $E = (H_1 * W_1) - q$ and $sl_i = 1$ or 0.
2. Converting each sl element to binary to form a sequence of bits $sl' = (sl_1, sl_2, ..., sl_{E*8})$.
3. Dividing sequence st' to nt' blocks $ct = (ct_1, ct_2, ..., ct_{N_1})$ of length N_1 where $nt' = \frac{(H_1*W_1)-q}{N_1}$
4. Decoding each block ct with RM(r_1, m_1) as shown in section (2.2) to form message blocks $tb' = tb'_1, tb'_2, ..., tb'_{k_1}$.
5. Decomposing all message blocks tb' to form a single linear block B.
6. Using dimensions of Share 1, n" x m", create a binary matrix/image, S2 from sequence B.
   If S2 and S1 are stacked and aligned correctly as described in [16], the Seed SE, number of cycles N'_c, and dimensions of secret image, H'_0 x W'_0 becomes visible.

4.2.2. *Generating random pixels.* The purpose of this function is to generate same integer values as in embedding process. The generated integers represent the indices/positions of the pixels embedded with secret bits. The seed S' initializes Blum Blum Shrub to generate the random sequence of integers, $pp = pp_1, pp_2, ..., pp_L$ as discussed in [13].

4.2.3. *LSB Extraction.* The purpose of this component is to extract the encoded secret bits that were embedded into less or equal to of the cover image. If the encoded bits E_T are greater than $\frac{3}{4} * H_1 * W_1$ then the extracted bits act as a reference to the other bits embedded into that same pixel. But if E_T are less or equal to $\frac{3}{4} * H_1 * W_1$ then the extracted bits represent the encoded bits E'. The extraction of LSBs is done as follows:

1. Inputting the randomly generated pixel positions pp from stego image I''.
2. Retrieving the pixels px' on indices pp in same order. Consider all stego pixels $ps' = ps'_1, ps'_2, ..., ps'_L$. Extracting the hidden bits by applying equation (19) to every px' to form a sequence of integers $eb = eb_1, eb_2, ..., eb_L$ where $n = 1$ and $ez_i = 1$ or 0.
3. Converting each eb' to binary to form a sequence of bits $eb' = eb'_1, eb'_2, ..., eb'_L$

4.2.4. *Key Trace Extraction.* The purpose of this component is to extract bits that were embedded into Agreed image A'. These bits are called the Key trace K_T. This process is executed if and only if $E_T \geq (\frac{3}{4} * H_1 * W_1)$. In this section, a sequence of Key Trace bits K_T, is extracted from image A' of size H_2 x W_2. To extract the K_T bits the following process was performed:

1. Determining the block size value of ns by: $ns = \frac{|K_T|}{H_2 x W_2}$. If ns is a not an integer, then there are two sets of block sizes:

   (a) Applying equation (19) to the first $\lceil ns \rceil$ pixels of A', to retrieve a sequence of integers $w' = w'_1, w'_2, ..., w'_{\lceil ns \rceil}$ where $ns = \lceil ns \rceil$.
   (b) if $((H_2 * W_2) - \lceil ns \rceil) \geq 1$, apply equation (19) to the last $((H_2 * W_2) - \lceil ns \rceil$ pixels of A', to retrieve a sequence of integers $w'' = w''_1, w''_2, ..., w''_{((H_2 * W_2) - \lceil n \rceil)}$ where $ns = ns$.
2. Appending sequence w'' to w' such that $ww = w'' + w'$.
3. Converting all ww to binary to produce a combined sequence of bits $wb = wb_1, wb_2, ..., wb_{E_T}$ called Key Trace K_T.

4.2.5. *Extraction process.* In this component, the bits that were re-embedded on each pixel are retrieved. This process is only executed if the secret message embedded was greater than the $\frac{3}{4}$ of the cover image. The process inputs Key Trace bits wb, the LSBs extracted from the stego ez' and reference key R. Extract the re-embedded bits as follows:

1. Dividing the Key Trace bits wb to h blocks $kb = (kb_1, kb_2, ..., kb_{\lfloor N_c \rfloor})$ and/or h' blocks $kb' = (kb'_1, kb'_2, ..., kb'_{N_c})$, where h and h' are defined in Theorem 4.2.
2. To each created block kb, inserting wb_l on first index and R on last index of the block such that $kb = (wb_l, kb_1, kb_2, ..., kb_{\lfloor N_c \rfloor}, R)$ and/or $kb' = (wb_l, kb'_1, kb'_2, ..., kb'_{N_c}, R)$, for $1 \leq l \leq E_T$
3. Considering each of the blocks kb and kb' , the sequence of encoded bit block Eb is determined using algorithm 2:

```
Algorithm 2: Generation of Encoded bit block
  Input: Blocks $kb$ or $kb'$, $\lceil N_c \rceil$, Number of blocks $h$, $h'$
  Output: sequence of Encoded bit blocks $EB$
1 $EB \leftarrow [\,]$
2 $j \leftarrow 1$
3 $H \leftarrow h + h'$
4 for $l \leftarrow 1$ to $H$ do
5     while $j < \lceil N_c \rceil$ do
6         $ebb^l_j \leftarrow kb^l_j \oplus kb^l_{j+1}$.
7     Append $ebb^l_j$ to $EB$
```

4. Outputting a sequence of bits EB formed after decomposing all message blocks ebb to constituent bits. The sequence EB is similar to the encoded sequence bits.

4.2.6. *Decoding process.* In this process, the encoded bits are decoded. The decoder is responsible for performing error detection and correction to each of the input bits as stated in section 2.2 .The inputs to this process are the encoded sequence bits EB, and block length N_1. The decoding process is performed as follows:

1. Determining if $1 \leq E_T \leq (\frac{3}{4} * H_1 * W_1)$ then divide sequence EB to n'' blocks $cw = (cw_1, cw_2, ..., cw_{N_1})$ of length N_1 where $n'' = \frac{(H_1 * W_1) - q}{N_1}$.
2. Decoding each block of cw with RM(r_1, m_1) Decoder as shown is section (2.2) and use equation (15) to form message blocks $mb' = mb'_1, mb'_2, ..., mb'_{k_1}$.
3. Outputting a sequence of decoded bits $sb = (sb_0, sb_1, ..., sb_Y)$ formed after decomposing all message blocks mb' to constituent bits, such that $Y = 8 * H_0 * W_0$.

4.2.7. *Reconstructing Image.* In this segment, the decoded bits are reconstructed to form secret image SI. The process inputs the decoded bits sb and the secret image dimensions, H_0 x W_0 from the generated Key to generate the secret image using the following process:

1. Dividing the sequences of bits sb to bytes i.e $mb = mb_1, mb_2, ..., mb_8$.
2. Converting each byte mb to decimal to form a sequence of integer $s' = s'_1, s'_2, ..., s'_{H_0 \text{ x } W_0}$ where $(0 \leq s'_i \leq 255)$.
3. From the sequence of integers s', creating a H_0 x W_0 matrix that is the representation of the secret image.

5. **Security Model.** In this section, the security model of the system is discussed. It demonstrates how the information is exchanged between the sender and recipient. The assumption in this model is that the Agreed image A' is communicated earlier between the sender and the receiver via a private/secure channel. The security model is divided into 3 phases, Registration, Authentication and Stego exchange and are discussed as follows .

1. **Registration Phase:** It is the first phase of the security model. The purpose of this phase is to communicate S1 to receiver securely between Sender and Receiver. The Sender generates two shares S1 and S1 from the Secret Key using (2, 2) VCS as discussed in section (4.2). The Sender possesses S2 and transmit S1 to the Receiver via a secure channel. Upon receiving S1, the Receiver sends back a confirmation message "RECEIVED".
2. **Authentication Phase:** It is the second phase of the security model. In this phase, communication is done via public channel. Its function is to verify the validity of the Receiver. The Sender transfers S2 to the Receiver, and upon receiving S1, the Receiver superimpose the two shares and extract the seed visually as discussed in section (2.3.1). The extracted seed is hashed using SHA 512 algorithm to form H1. The Receiver transmits H1 to Sender so that the validity of the Receiver is verified. The Sender uses the same Hash algorithm (SHA 512) to hash the seed that he/she possesses to form H2. Upon receiving H1, the Sender verifies if H1 and H2 are the same. If the two hash values are equal then the Receiver is valid and if not then he/she is not.
3. **Stego exchange**: It is the last phase of the security model. The purpose of this phase is to communicate a Stego image between two users. It only occurs if and only the Receiver has been authenticated in phase 2. It involves all the steps as described in sections 4.5 and 4.6.

6. **Experiments and Discussion.** This section discusses the result of the experiments that were conducted for evaluating the capacity, robustness and imperceptibility as well as the analysis of results.

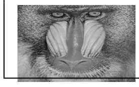
6.1. **The Experiment result for Capacity Evaluation.** The objective of the experiment was to determine the embedding capacity of the cover images for both Molaei's and proposed methods. The experiment was conducted to 30 grayscale cover images of size (512 x 512) of different histograms. Both methods were used to embed 20 grayscale secret images of different sizes on each cover image. The embedding capacity EC is expressed as percentage and is calculated using equation (38) in [12]:

$$EC = \frac{|S|}{H * W} \tag{38}$$

where $|S|$ is the number of secret bits embedded on a cover image of height H and width W. Table 1 summaries the results of the experiment.

6.2. **The Experiment result for Imperceptibility Evaluation.** The objective of the experiment was to evaluate the visual quality of the stego images for both Molaei's and proposed methods.

TABLE 1. A Capacity Comparison between Proposed Method and Previous method

Cover Image	Secret Image size(bits)	Capacity (%)		PSNR(dB)	
		Molaei's Method	Proposed Method	Molaei's Method	Proposed Method
	480	0.37	0.59	75.58	62.64
	196608	150	240	48.13	52.09
	368640		450		52.08
	480	0.37	0.59	75.61	62.36
	196608	150	240	48.12	52.03
	368640		450		52.04
	480	0.37	0.59	75.62	62.29
	196608	150	240	48.10	52.02
	368640		450		52.01

The experiment was conducted similarly to section 6.1. PSNR criterion was used to evaluate the visual quality, and is calculated using equation (39) in [12]:

$$PSNR = 10log_{10}\frac{255^2}{MSE} \tag{39}$$

where MSE refers to the difference between the pixel values of Cover and stego images. It is calculated using equation (40) in [12]:

$$MSE = \frac{1}{H*W}\sum_{i=1}^{H}\sum_{j=1}^{W}(I_{ij} - I'_{ij})^2 \tag{40}$$

where I_{ij} and I'_{ij} are the pixels values of cover and stego image respectively. The lower the MSE value means the difference between the stego and cover is lower and as a result, the higher the PSNR value which is more desirable in steganography. Table 1 summaries the results of the experiment.

6.3. **Experiment result for robustness against attacks.** In this experiment a set of 20 different images were used as the cover images and another set of 20 different images as secret images. Each cover image was embedded with 33 different sizes of each secret image. The stego image formed at each experiment was subjected to noise, (Gaussian Noise, Salt and Pepper Noise, and Speckle Noise), Cropping, Scratching(Single and Multiple), JPEG compression. The experiment was performed to recover the hidden secret image from the attacked stego image. The results of the experiments are shown in Table 2, 3, 4 and 5 respectively.

TABLE 2. Experiment Result of robustness against noise attack

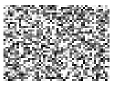
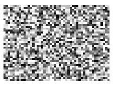
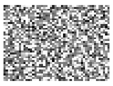
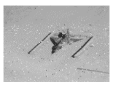
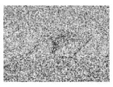
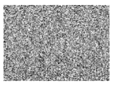
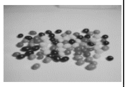
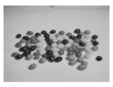
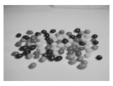
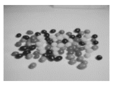
Secret Image and Size	Stego Image	Noise Attacks	Proposed Method			Molaei's Method		
			PSNR Attacked Stego(dB)	PSNR Secret Image(dB)	Recovered Secret Image	PSNR Attacked Stego(dB)	PSNR Secret Image(dB)	Recovered Secret Image
15360	20	Gaussian:Mean 0 Variance 0.01	20.218	9.359		20.213	9.336	
		SaltnPepper: Density 0.05	18.353	100		18.271	28.390	
		Speckle: variance 0.04	21.350	9.324		21.357	9.654	
196608	11	Gaussian: Mean 0 Variance 0.1	11.453	17.176		11.354	9.276	
		SaltnPepper : Density 0.5	8. 240	28.124		8. 216	11.505	
		Speckle: variance 0.4	11.912	17.690		11.620	9.842	
 368640	Stego 11	Gaussian: Mean 0 Variance 0.1	11.454	100				
		SaltnPepper : Density 0.5	8. 240	100				
		Speckle: variance 0.4	11.872	100				

Table 3. Experiment Result of robustness against cropping attack

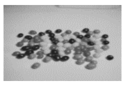
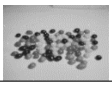
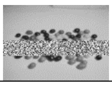
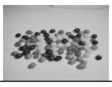
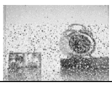
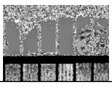
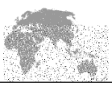
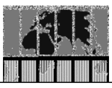
Secret Image	Stego Image	Secret Size (bits)	Cropping Area		Proposed Method		Molaei's Method	
			Height (pixel)	Width (pixel)	PSNR Secret Image(dB)	Recovered Secret Image	PSNR Secret Image(dB)	Recovered Secret Image
	3	196608	0 - 512	0 - 360	39.357		15.863	
		368640	0 - 512	0 - 360	100			
	4	196608	70 - 422	80 - 422	14.834		6.754	
		368640	70 - 422	80 - 422	100			
	7	196608	20 - 435	40 - 452	16.009		4.998	
		368640	20 - 435	40 - 452	100			

Table 4. Experiment Result of robustness against scratching attack

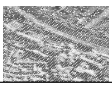
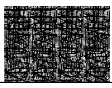
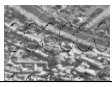
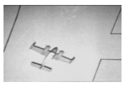
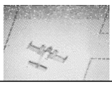
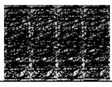
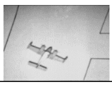
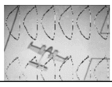
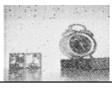
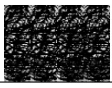
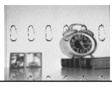
Secret Image	Stego Image	Secret Size (bits)	Scratches	Proposed Method		Molaei's Method	
				PSNR Secret Image(dB)	Recovered Secret Image	PSNR Secret Image(dB)	Recovered Secret Image
	1	131072	Multiple	15.256		6.850	
		196608	Single	59.571		20.482	
	2	131072	Multiple	20.957		3.478	
		196608	Single	100		18.243	
	4	131072	Multiple	17.717		3.390	
		196608	Single	100		21.543	

Table 5. Experiment Result of robustness against JPEG compression

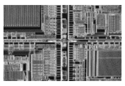
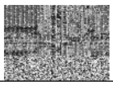
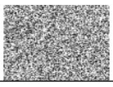
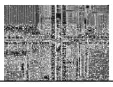
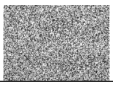
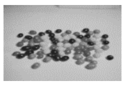
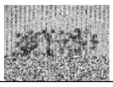
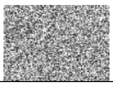
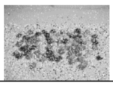
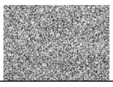
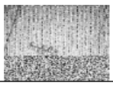
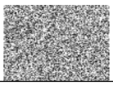
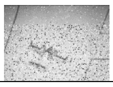
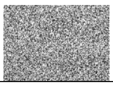
Secret Image	Stego Image	Secret Size (bits)	Proposed Method			Molaei's Method		
			PSNR Attacked Stego(dB)	PSNR Secret Image(dB)	Recovered Secret Image	PSNR Attacked Stego(dB)	PSNR Secret Image(dB)	Recovered Secret Image
	5	92160	42.699	11.176		42.351	8.175	
		196608	42.696	15.382		42.340	8.157	
	3	92160	42.680	12.119		42.455	9.156	
		196608	42.675	17.831		42.465	9.108	
	2	92160	43.388	11.820		44.279	8.909	
		196608	43.391	18.565		44.598	8.779	

Table 6. Experiment Result of Steganalysis

Cover Image	Secret Size (bits)	Molaei's Method		Proposed Method	
		True P value	Estimated P value	True P value	Estimated P value
4	15360	0.058594	0.053251	0.128125	0.10145
	196608	0.75	0.47209	0.409375	0.395543
	368640			0.405469	0.39146
	15360	0.058594	0.053862	0.128125	0.10089
	196608	0.75	0.4199	0.409375	0.392145
	368640			0.405469	0.38146
	15360	0.058594	0.052611	0.128125	0.09089
	196608	0.75	0.4012	0.409375	0.38795
	368640			0.405469	0.38001

6.4. **Experiment Results for Steganalysis.** In this experiment a set of 20 different images were used as the cover images and another set of 20 different images as secret images. Each cover image was embedded with 33 different sizes of each secret image. Each stego image formed at each experiment was subjected to steganalysis attacks to detect and estimate the embedded secret message size. The proposed steganalysis method (Triples) estimated the value of p (fraction of the secret message hidden in a given cover image). The results of the experiment is shown in the Table 6.

7. **Analysis of Experiment Results.**

In this section the analysis of experiment results is done. Analysis is performed on Capacity, Imperceptibility, robustness and steganalysis.

7.1. **Analysis of Capacity and Imperceptibility between proposed and Molaei's Methods.**

According to the experiment results of capacity obtained, Fig 10 shows that for each secret message size the proposed method achieves a greater embedding capacity than Molaei's method. This occurred because of the error correction code and the multiple embedding method adopted in the proposed method. Reed Muller code (1, 4) used in proposed method encodes a message block of size 5 into a code-word of size 16 therefore encoding expansion is $\frac{16}{5} = 3.2$. Molaei's method implemented Reed Muller (1,3) and Reed Muller (2,5) codes which encodes a message block of size 4 or 8 into a code-word of size 8 or 16 and its encoding expansion is $\frac{8}{4} = 2$. Clearly the expansion of the proposed method is higher than Molaei's method. For any given secret message size the proposed method encodes a message to code-words of greater size than the previous method by an expansion factor of $\frac{3.2}{2} = 1.6$.

The proposed method introduced multiple embedding method which only embeds message into the first LSB of some pixels unlike Molaei's method which is limited to embedding data only on the first and second LSBs of the cover's pixels. In multiple embedding method the LSB of each pixel of the cover image is embedded more than once. The embedding capacity is increased because the LSB of each pixel can be embedded more than 1 message bit. The proposed method achieved maximum embedding capacity of 450(%), while Molaei's method has a maximum capacity of 150(%) which is $\frac{450}{150} = 3$ times less than the Proposed Method. Its maximum embedding capacity of 150(%) is achieved by embedding both the first and second LSB of each pixel of the cover image.

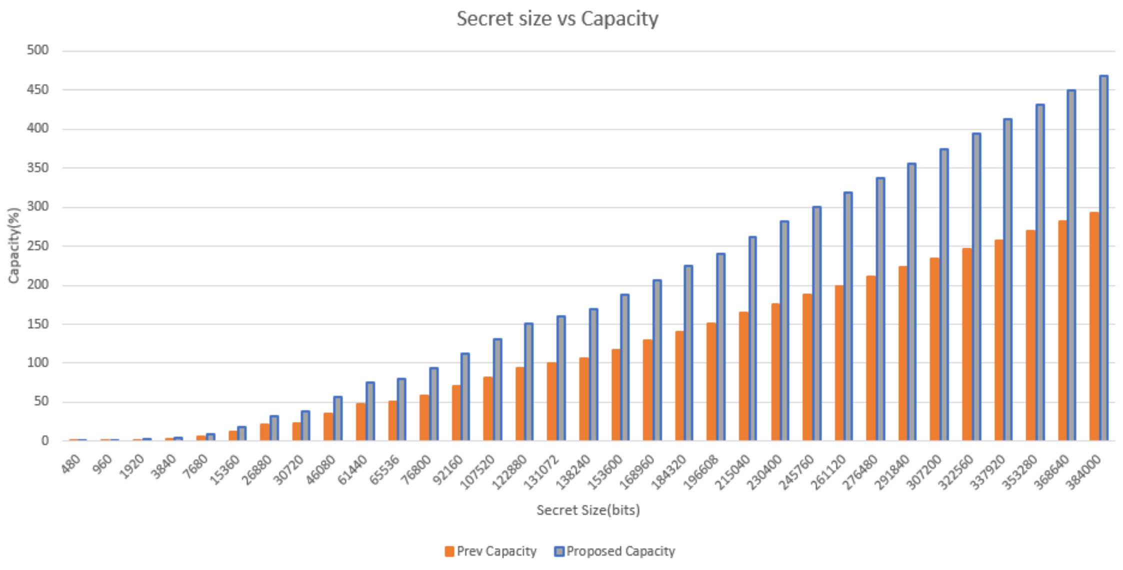


Figure 10. The experiment result of embedding capacity comparison between Molaei's Method and the Proposed Method

Embedding into the second LSBs as suggested by Molaei increases the Mean Square Error between the cover and the resulting stego image. This decreases the overall image quality (PSNR value) of the final stego to (48dB). However, the proposed method used $\frac{3}{4}$ of the pixels of each cover image and re-embedded the remaining secret bits into the same pixels that were embedded before. This guarantees that a maximum of $(\frac{3}{4} * 512 * 512) = 196608$ LSBs of the cover pixels will only be changed after embedding any secret message size. These small changes leads to a low Mean Square Error therefore multiple embedding method achieves high capacity accompanied with a good visual quality $\geq$51dB. While Molaei's method has $\leq$48dB.

7.2. **Analysis of Robustness against Attacks between Proposed and Molaei's methods.** Tables 2, 3, 4 and 5 shows that the proposed method has better error correction capability as compared to Molaei's method. This occurred because of the different error correction codes implemented in proposed and Molaei's method. The proposed method implemented Reed Muller (1,4) codes.

From definition (2.1), Reed Muller (1,4) code encode a message of size (k): $k = \sum_{i=0}^{r} \binom{m}{i} = \sum_{i=0}^{1} \binom{4}{i} = \binom{4}{0} + \binom{4}{0} = 5$ bits. And a code word of size,$N = 2^m = 2^4 = 16$ bits. The maximum number of error bits this code can correct: $t = \left\lfloor \frac{2^{4-1}-1}{2} \right\rfloor = 3$ bits. Error Correction capability, $E_c = \frac{\text{number of corrected bits}}{\text{total bits}} = \frac{3}{16} = 18.75(\%)$. While Molaei's method implemented Reed Muller (1,3) codes for first LSB embedding and Reed Muller (2,5) codes for second LSB embedding. The code has message sizes (k), 4 and 16 bits respectively. These are encoded to codewords of sizes, N, 8 and 32 bits. These codes can correct a maximum of $t_1 = \left\lfloor \frac{2^{3-1}-1}{2} \right\rfloor = 1$ error bit and $t_2 = \left\lfloor \frac{2^{5-2}-1}{2} \right\rfloor = 3$ error bits for RM(1,3) and RM(5,2) respectively.Therefore the error correction capacity for each of the codes is 12.5 (%) and 9.38 (%) respectively. Therefore the proposed method suggested an error correction code that outperforms the one suggested in Molaei's method.

The Key Trace embedded into the Agreed image play a major role in error correction. The Key Trace is free from any attacks and helps increasing the error correction capability during extraction process. The secret data size increases as the key trace data also increases, $K_T = \frac{|S|*N}{k} - \frac{3}{4} * H * W$, where $|S|$ is the size of secret in bits, N is the block size, H and W is the Height and Width of the cover image.

7.2.1. *Analysis of Robustness against Noise Attacks between Proposed and Molaei's methods.* The experiment results shows that the secret size and contrast are two main factors that affects the recovering of secret images. The Proposed Method out performs Molaei's method and successfully recovers the secret image from stego images under Salt and Pepper: Variance 0.04 for smaller secret message sizes. This occurred because Salt and Pepper has a smaller variance hence makes fewer distortions on the stego such that each retrieved codeword has number of errors $t \leq 3$ bits. These are easily correctable by the proposed Reed Muller code. While Molaei's method recovered poor quality secret images because the implemented code, Reed Muller (1,3), can correct only 1 error on each codeword block yet the recovered code words have $\geq$ 1 errors and are not correctable.

Under Gaussian and Speckle attacks, these small secret sizes cannot be retrieved (not visible) even at any level of contrast for both Molaei's and Proposed Methods. This is caused by the large number changes on pixel values of the stego which results in retrieved codewords with a number of errors greater than t. However, for the secret images with low contrast, the proposed method has better PSNR values than Molaei's method because it has a (18.75(%) - 12.5(%)) = 6.25(%) better error correction capability than that of Molaei's.

Considering secret images of size $\geq$ 107520 bits, the Proposed method out performs Molaei's method for all Noise attacks. The Proposed method can fully recover the secret messages (100dB) of any level of contrast under Salt and Pepper: Variance 0.04 while Molaei's method cannot $\leq$ 40dB. This occurred because for each block of the message to be re-embedded into each LSB of the pixel, only 1 bit of the block is embedded and the remaining block is represented as Key Trace and this Key Trace cannot be corrupted by any kind of attack. This increases the error correction capability of the code since fewer bits will be corrupted as the message size increases. This causes t to approach 3 correctable bits rapidly. However both Methods cannot recover fully the secret image under Salt and Pepper: Variance 0.4 and Gaussian Noises and Speckle Noise but the Proposed method recovers secrets with better visibility than Molaei's method.

Considering secret images of size $\geq$ 368640 bits, the Proposed method performs best for all Noise attacks. It can successfully retrieve any secret image $\geq$ 368640 bits of any level of contrast. In this case the code is able to correct all corrupted blocks. At this size the number of errors in each block of code word is exactly $t \leq 3$ and is therefore correctable successfully.

As the size of the secret image increases, the error correction capability of the proposed error correction code increases thereby increasing the recovered secret image quality, PSNR. The Key Trace embedded into the Agreed image is free from any Noise attacks and helps increasing the error correction capability during extraction process. The quality of the retrieved secret image increases as the size of secret image increases.

7.2.2. *Analysis of Robustness against Cropping Attacks between Proposed and Molaei s methods.* For reduced cropping regions, the proposed method achieves better results than Molaei s method. Although it cannot recover the secret images fully but the secret image is quite visible compared to Molaei s method. This occurred because the number of errors in most retrieved codewords caused by the cropping approaches $t \approx 3$. There are greater than the minimum number of correctable errors for each codeword, therefore cannot be fully correctable.

Considering medium sized secrets within [46080 − 353280] bits, the Proposed method retrieves better quality secret images as compared to Molaei s method. Molaei s method struggled to recover low contrast secret images, most of which are highly noisy, not visible and contains large black regions. As the size of the secret image increases, the error correction capability of the proposed error correction code increases thereby increasing the recovered secret image quality, PSNR.

Considering large secret sizes within [368640 − 384000] bits, the Proposed Method successfully recovers the hidden secret image under and any cropping region size attack. As the secret image size increases the retrieved secret image quality drastically increases under any of the applied cropping region attacks. This occurred because of the PRNG adopted to randomly embed and extract the secret images to avoid burst error which are more common in Molaei s method.

7.2.3. *Analysis of Robustness against Scratching Attacks between Proposed and Molaei s methods.* Based on Table 5, the proposed method can successfully recover the secret message on Single Scratching attacks for all high contrast secret image of any size. This because a single scratch alters few pixels on the stego and since the secret data was embedded randomly, therefore the errors does not belong to a single code word. Therefore burst errors are not encountered in the proposed method than in Molaei s method which embedded data serially. The number of errors in each retrieved codeword is $t \leq 3$ which is correctable. Molaei s method cannot recover fully secrets because as the embedded area on the cover increases, the number of alterations caused by scratching also increases. The possibility of retrieving code words with increased errors $t \geq 1$ and burst errors increases.

Considering Multiple scratch attacks, the Proposed Method cannot successfully retrieve all smaller secret image ($\leq 353\,280$) bits. This is because all of the secret data is embedded into the cover and scratching alters a larger part if not all of the data thereby increasing the value of t error bits. And if t is greater than 3 then the code words cannot be corrected successfully. The quality of the retrieved secret image increases as the size of secret image increases. This is because as the secret data size increases the key trace data also increases.

However, large secret sizes (≥ 368640 bits) can be retrieved by the Proposed Method. This is because the size of Key trace bits in each retrieved codeword increased and $t \leq 3$. When $t \leq 3$ the codeword is correctable. Molaei s method cannot retrieve any secret images under Multiple Scratches because the scratches caused many alterations such that for every retrieved code word the number of errors $t_1 \geq 1$, $t_2 \geq 3$ and are not correctable. It also causes large burst errors which are not correctable with the suggested error correction code.

7.2.4. *Analysis of Robustness against JPEG compression Attacks between Proposed and Molaei s methods.* :

Table 6 shows that Molaei s method cannot successfully recover any secret message size of any level of contrast under JPEG compression attacks. The recovered secret images have low PSNR values and not not visible. This was caused by the large changes in pixel values caused by JPEG compression which affect the embedded pixel values. They result in many alterations of the stego pixels such that for every retrieved code word the number of errors $t_1 \geq 1$, $t_2 \geq 3$ and are not correctable. It also causes burst errors which are not correctable with the suggested Reed Muller codes.

On the other hand, the Proposed method can only retrieve successfully large secret images of sizes ≥ 368640 bits. This occurred because the size of Key trace bits in each retrieved codeword increased and t ≤ 3. When t ≥ 3 then the codeword is correctable. However, the proposed method performs poorly for smaller secret images. This occurred because the value of Key trace for smaller secret data is zero which means error correction will rely only on the data retrieved from the stego. And this data is highly altered or distorted by JPEG compression, therefore the number of errors t ≥ 3 and is not correctable. The retrieved secret images of size (≥ 92160) bits are quite visible and the noise on the images decreases as the size of the secret image increases. Therefore, the retrieved secret image PSNR value under proposed method increases as the size of the secret image embedded increases.

7.3. **Steganalysis for Proposed and Molaei's method.**

An improved Steganalysis method proposed by Andrew D. Ker[1], called Triples Analysis. It is a detector used for simple LSB steganography in digital images. It uses the structural and combinatorial properties of the LSB embedding method to detect and estimate the length of the hidden messages. Triples has proved to be a more powerful and sensitive detector than other structural detectors such as RSA, SPA etc. It utilizes the effects of LSB changes on arbitrary groups of samples ie 3-tuples of pixel groups. Triples assumes that the Cover image is fixed and a random hidden message is embedded.

Suppose the hidden message has length $2pN$, where $0 \leq p \leq 0.5$ is unknown to the detector, is embedded using LSB embedding of a random selection of samples independent of the content of the cover or hidden message[1] . A trace set $C_{m,n}$ is created in which all pixel samples $s_1, , s_N$ are drawn, where $n,\ m > 0$ and N is number samples. The probability of transition from one trace subset to another is $p^i(1-p)^{(3-i)}$, where i is the length of the shortest path between them. The trace set and subsets are constructed using equations (41), (42) and (43) in [1].

$$C_{m_1,\ldots,m_{g-1},n_{g-1}} = (s_1, , s_g) \in T|[\frac{s_{i+1}}{2}] = [\frac{s_i}{2}] + m_i \text{ and } [\frac{s_{i+2}}{2}] = [\frac{s_{i+1}}{2}] + n_i \text{for each} 1 \leq i \leq g \tag{41}$$

$$E_{m_1,\ldots,m_{g-1},n_{g-1}} = (s_1, , s_g) \in T|s_{i+1} = s_i + m_i \text{ and } s_{i+2} = s_{i+1} + n_i \text{ with } m_i \text{ even} \tag{42}$$

$$O_{m_1,\ldots,m_{g-1},n_{g-1}} = (s_1, , s_g) \in T|s_{i+1} = s_i + m_i \text{ and } s_{i+2} = s_{i+1} + n_i \text{ with } m_i \text{ odd} \tag{43}$$

When a single sample has the LSB altered a tuple transitions to either of the following trace subsets $E_{2m,2n}$, $O_{2m,2n}$, $E_{2m+1,2n-1}$, $O_{2m,2n-1}$, $E_{2m,2n+1}$, $O_{2m-1,2n+1}$, $E_{2m+1,2n}$, and $O_{2m-1,2n}$. Given a stego image, consider each trace set $C_{m,n}$ in turn and count the trace subsets to make a vector x' using equations (44) in [1].

$$x'' = T_3^{-1}\ x' \tag{44}$$

Then we can hypothesize a value of p and form estimates for the sizes of the trace subsets of the cover image using the transition matrix, T_3. The parity symmetry is checked if it s true for both covers and also stego images when both m and n are even or equal. Considering one case of parity symmetry, $e_{2m+1,2n+1} = o_{2m+1,2n+1}$. To use the generalized framework to make an estimate of p, we compute error terms for each m and n, $\epsilon_{m,n} = e'_{2m+1,2n+1} - o'_{2m+1,2n+1}$. Then we find the value of p which minimizes the sum-square of the errors. Firstly, using the transition matrix, the number of errors is determined by

$$\epsilon_{m,n} = \frac{1}{8}((d_0+d_1+d_2+d_3)+q(3d_0+d_1-d_2-3d_3)+q2(3d_0-d_1-d_2+3d_3)+q3(d_0-d_1+d_2-d_3)) \tag{45}$$

Where d_i are the differences of various combinations of $e_{2m+i,2n+i}$ and $o_{2m+1,2n+1}$ subsets.

Find the value of q to minimize $S(q) = \sum_{m,n} \epsilon^2_{m,n}$. We have

$$S(q) = \frac{1}{64}\sum_{m,n} c_0^2+q(2c_0c_1)+q^2(2c_0c_2+c_1^2)+q^3(2c_0c_3+2c_1c_2)+q^4(c_2^2+2c_1c_3)+q^5(2c_2c_3)+q^6(c_3^2) \tag{46}$$

Differentiating the above equation (47) and solving for q will give up to 5 roots for q. All roots inside the range $(-10,\ 10/11)$ are discarded because they will give wrong estimates of p outside $(-0.05,\ 0.55)$). Substitute the remaining roots back into (47) to determine the location of the minimum q'. Finally, the estimate of p, $p' = \frac{1}{2}(1-\frac{1}{q})$.

Triples has been implemented to determine how the proposed and Molaei s method perform under steganalysis. Triples was able to detect the presence of a hidden message in both the proposed and Molaei s method for every Cover image used. The margin between the estimated and true p value increases as the length of the hidden secret message increases as shown in Table 6.

7.3.1. *Analysis of Triples Results using Proportion.* Proportion was used to compare the performance of Molaei s method and proposed method under steganalysis. Using the results of True p value (T_p) and Estimated p value (E_p) for each secret message size recorded on Table 7, proportion P was calculated as follows:

$$P = \frac{E_p}{T_p} \tag{47}$$

A larger value of P means that the estimated p value is closer to the true p value. It means that the probability of attacker to predict the true p value is high. A smaller P value means that there is a bigger difference between the estimated p value from the true value so the attacker cannot easily predict the true p value. A high value of proportion means that the there is a smaller difference between the Estimated p value and the True p value. That means the Triples estimated the p value almost accurately. For smaller secret sizes, the proportion values of the proposed method is always smaller than Molaei s method. This is caused by the high expansion rate of 3.2 in Reed Muller (1,4) code used in proposed method compared to 2 in Reed Muller(1,3) code used in Molaei s method. This means that for any given secret message size, the proposed method results in a larger secret size to be embedded than Molaei s method. And Triples method has been proved to be highly sensitive to smaller secret sizes than large ones, hence the proposed method performs better than Molaei s method.

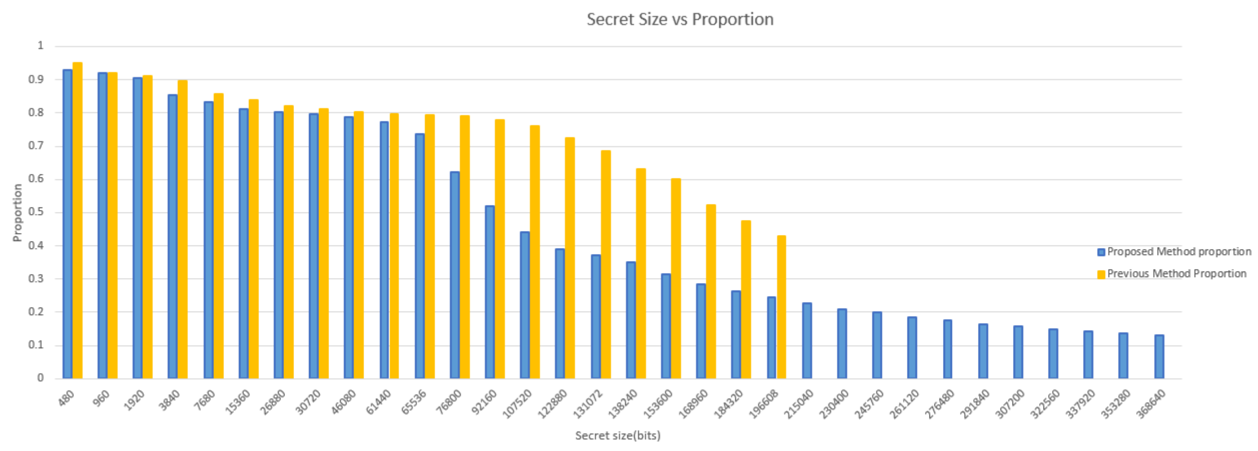


FIGURE 11. A comparison of Steganalysis results for Molaei s method and proposed method based on proportion

The proportion values of the proposed method decreased rapidly for secret sizes [61440 – 368640 bits] while the proportion values of Molaei s method decreased steadily as shown on Fig 11. This occurred because of the Multiple embedding method adopted. Multiple embedding re-substitutes different secret bits on LSB of pixels that have already been embedded, such that at the end of the embedding process each pixel has either 1 change or same LSB as of the original/initial LSB. Triples method can only detect the message length based on the last embedding cycle. Due to this, there is an increase in True p value while the Estimated value is ≈ 0.3. It is because of this that the difference between the proportion values of proposed and Molaei s methods is very large. Therefore the proposed method is less detectable under steganalysis than Molaei s method.

8. **Conclusions.**

A Multiple embedding method was proposed to achieve a high embedding capacity (450%) in Code based steganography scheme. The method contributed to an improved visual quality ($\geq$ 51dB) of the Stego image that ensured high Imperceptibility. Furthermore, Multiple embedding had a huge impact on the overall robustness of Code based steganography schemes against noise, scratching, cropping and jpeg compression attacks. All extracted secret images had an improved visual quality reaching to 100dB for high embedding capacity (450%) while Molaei s method achieved very low quality extracted secret images. The security of the hidden secret message was also improved by embedding secret message randomly across the image by using a PRNG.

REFERENCES

[1] A. D. Ker, A general framework for structural steganalysis of lsb replacement, *In International Workshop on Information Hiding*, pp. 296-311. Springer, 2005.

[2] A. M. A.-A. Selami and A. F. Fadhil, A study of the effects of gaussian noise on image features, *Kirkuk University Journal for Scientific Studies*, vol. 11, no. 3, pp. 152-169, 2016.

[3] A. Molaei, M. Sedaaghi, and H. Ebrahimnezhad, Steganography scheme based on reed-muller code with improving payload and ability to retrieval of destroyed data for digital images, *AUT Journal of Electrical Engineering*, vol. 49, no. 1, pp. 53-62, 2017. doi: 10.22060/eej.2016.814.

[4] B. Cooke, Reed-muller error correcting codes, *MIT Undergraduate Journal of Mathematics*, pp. 21-26, 1999.

[5] C.-C. Thien and J.-C. Lin, A simple and high-hiding capacity method for hiding digit-by-digit data in images based on modulus function, *Pattern recognition*, vol. 36, no. 12, pp. 2875-2881, 2003.

[6] D. Taranovsky. *Data Hiding and Digital Watermarking*, pp. 387-399. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-540-79567-4. doi: 10.1007/978-3-540-79567-4 31. URL https://doi.org/10.1007/978-3-540-79567-4_31.

[7] M.A. Hassan and M.A. Khalili, Self watermarking based on visual cryptography, *In Proceedings of World Academy of Science, Engineering and Technology*, vol. 8, pp. 159-162, 2005.

[8] H. V. Desai, Steganography, cryptography, watermarking: A comparitive study, *Journal of Global Research in Computer Science*, vol. 3, no. 12, pp. 33-35, 2013.

[9] J. C. Judge, Steganography: past, present, future. Technical report, Lawrence Livermore National Lab., CA (US), 2001.

[10] J. Van Lint and G. Van der Geer, *Introduction to coding theory and algebraic geometry*, vol. 12, Birkhauser Boston, Inc. Secaucus, NJ, USA, 2012. ISBN0817622306.

[11] M. Mansourpour, M. Rajabi, and J, Blais, Effects and performance of speckle noise reduction filters on active radar and sar images, *In Proc. ISPRS*, vol. 36, pp. W41, 2006.

[12] M. V. R. CH et al., Medical image watermarking schemes against salt and pepper noise attack, *International Journal of Bio-Science and Bio-Technology*, vol. 7, no. 6, pp. 55-64, 2015.

[13] P. Junod, Cryptographic secure pseudo-random bits generation: The blum-blum-shub generator, 1999.

[14] R. Crandall, Some notes on steganography. *Posted on steganography mailing list*, pp. 1-6, 1998. URL http://os.inf.tu-dresden.de/ westfeld/crandall.pdf.

[15] S. B. Wicker, *Error control systems for digital communication and storage*, vol. 1, Prentice hall Englewood Cliffs, 1995.

[16] S. Cimato and C.-N. Yang, *Visual cryptography and secret image sharing*, CRC press, FL, USA, 2017. ISBN 978-1-4398-3721-4.

[17] S. Lin and D. J. Costello, *Error control coding*, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2001. ISBN 0-13-200809-2.

[18] S. Kaisar and J. A. Mahmud, Salt and pepper noise detection and removal by tolerance based selective arithmetic mean filtering technique for image restoration, *International Journal of Computer Science and Network Security*, vol. 8, no. 6, pp. 271-278, 2008.

[19] Y. Zhang, J. Jiang, Y. Zha, H. Zhang, and S. Zhao, Research on embedding capacity and efficiency of information hiding based on digital images, *International Journal of Intelligence Science*, vol. 3, no. 2, pp. 77, 2013.