The 2018 International Conference on Information Technology,
Engineering, Science, and its Applications

Rubikstega: A Novel Noiseless Steganography Method in Rubik's Cube

Ade Yusuf Rahardian[1,*], *Rinaldi* Munir[2,*], and *Harlili* Harlili[3,*]

[1,2,3]School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia

Abstract. Steganography is the art and science of hiding messages in such a way that no one is aware that there is a secret message. In this paper, Rubikstega, a new steganographic method in Rubik's Cube, is proposed. By utilizing Rubik's Cube scramble notation, a steganography scheme is created to hide messages without creating any noise (noiseless steganography). Previously, noiseless steganography in the game domain has been applied in chess games, Tetris, and Minesweeper. Those researches have concerns such as the need for authentic data, message hiding capacity for one game, and inefficient long message extraction process. Therefore, Rubik's Cube is proposed to overcome those shortcomings. Experimental results show that the proposed method is undetectable and have large messages capacity per game.

1 Introduction

With the current advancement of technology, the exchange of information can be made easier. This is due to the growing media of intermediaries such as telephone, electronic mail, and social media. However, with this advancement, it is possible that the information sent can be tapped and misused by unauthorized parties. Various way of tapping can be overcome by hiding the information with steganography.

Steganography is a technique of securing the message by embedding the message to a cover such as images, audio, and video. Fundamentally, the purpose of steganography is not to deter enemies from translating secret messages, but to prevent the enemy from suspecting the existence of hidden communications [2].

According to Desoky, steganography is divided into two types, namely noisy steganography and noiseless steganography (Nostega) [2]. In noisy steganography, messages are hidden inside a cover as noise, whereas noiseless steganography is not hiding the messages inside a cover, but the message is the cover itself. With Nostega, the process of exchanging messages between parties can be done with the certain approved domain without others suspecting if the two parties exchange messages.

One of Nostega's methodologies in the game is chess steganography or Chestega [1]. Desoky wrote that there are several ways to hide messages, such as the use of chess movements, tournament names, tournament results, and player names. However, although hidden messages may be long, there are still some shortcomings in the use of cover in

* Corresponding author: [1]ade.yusuf.r@gmail.com, [2]rinaldi@informatika.org, [3]harlili@informatika.org

Chestega. For example, the illegal movement of chess will increase the suspicion of the enemy. Thus, using the movement of the chess as a parameter is limited only to legal movement. In addition, the availability of authentic data plays an important role in determining the encoding parameters.

Ou and Chen introduced a steganography method, which suits with the definition of Nostega, using the Tetris game [5]. In this method, the secret message is hidden within the tetrimino series. Ou and Chen's research requires the recipient to play Tetris to get the message sent. Although it can extract messages while playing, this results in an inefficient message extraction process because it takes a long time to get the entire tetrimino sequence needed, especially if the message is very long.

Other noiseless steganography in the domain of game is also proposed by Mahato, Yadav, and Khan [4]. Mahato et al. proposed a new method to hide data in a Minesweeper game. In this method, the position of mines on the Minesweeper grid is used to hide data. However, because of the current data hiding capacity (17.6 bits/game for beginner level and 214 bits/game for the hard level) [4], it takes a lot of games if the sender wants to send a long message.

The shortcoming of the use of noiseless steganography paradigm in the aforementioned game can be overcome by using more flexible game within its boundaries (such limitation of legal moves or flexibility in creating fictitious data), more efficient in extracting the message, and better data hiding capacity per game. One such eligible game is Rubik's Cube. Rubik's Cube is a mechanical puzzle game found in 1974 by Erno Rubik. The purpose of the game is to solve random Rubik's Cube state and return it to the solved state. Utilizing many notations that exist in Rubik's Cube is one thing that can be used to hide messages with the Nostega paradigm.

2 Rubik's Cube basic

2.1 Rubik's Cube notation

To describe the movement in Rubik's Cube, a notation is used to tell which side is moving. Each face of the puzzle is marked with a letter **F** (Front), **B** (Back), **R** (Right), **L** (Left), **U** (Up), and **D** (Down). The Rubik's Cube movement is referred to as Singmaster notation [7]. Rubik's Cube with 3x3x3 dimensions has 18 types of half-turn metric notation [9], namely:

1. **F**, **B**, **R**, **L**, U, and **D** (turn 90 degrees clockwise)
2. **F'**, **B'**, **R'**, **L'**, **U'**, and **D'** (turn 90 degrees counterclockwise)
3. **F2**, **B2**, **R2**, **L2**, **U2**, and **D2** (turn 180 degrees)

The 18 types of notations in Fig. 1 are used as a reference in the movement of the Rubik's cube, both for scrambling and solving. In Rubik's Cube, a series of notations are applied to the puzzle to get a random/scrambled state.

2.2 Diameter of Rubik's Cube

Based on research by Rokicki et al., the diameter of Rubik's Cube is 20 [6]. In other words, with 4.3×10^{19} possible state positions, any state can be solved with the number of movement less than equal to 20 half-turn metric notation. Therefore, the current range of random scrambling generator for 3x3x3 Rubik's Cube will produce (more or less) 20 moves for a scramble.

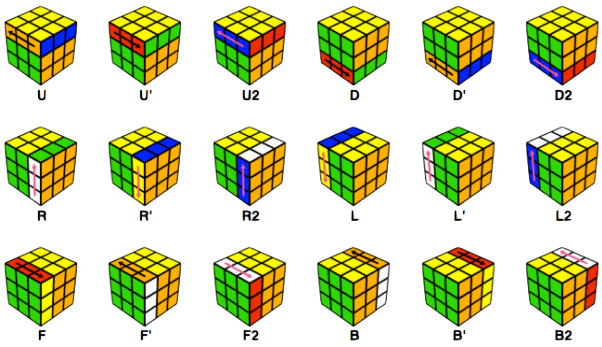

Fig. 1. Rubik's Cube notation [8].

3 Proposed method

There are some parameters in Rubik's Cube that can be used as a parameter to conceal data without creating noise (e.g. state of the puzzle, color position, solution algorithm, scramble notation). State of puzzle and color position have limitations in its possible position, while the solution algorithm may arouse suspicion because of strange movements. Therefore, scramble notation is chosen because its capability to embed a very long message and its easiness to create fictitious data.

As a note, the selection of sequences of random notation (scramble notation) can't be separated from its constraints. For example, **R** can't be continued with **R**, **R2**, or **R'** because it can be merged to one **R** face notation. Fundamentally, there are two constraints to make scramble notation:

1. A notation must be followed by another notation on a different face.
2. Notation on a face is valid if it cannot be merged with the same face notation in some previous notation. In other words, before the face is turned, there must be a face on the other axis that is turned before this face is turned again. For example, **R U2 R** is valid but **R L2 R** is not valid.

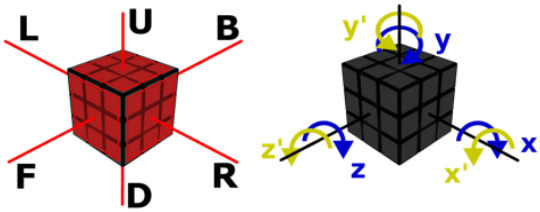

Fig. 2. Faces (left) and axes (right) on Rubik's Cube [10].

Illustration on the faces and axes of the Rubik's cube is shown in Fig. 2. Due to the limitations of scramble notations, face groupings by the axis are required to meet the previously mentioned constraints. This grouping can be seen in Table 1.

With the grouping of faces based on the axis, the 18 unique notations in the Rubik's Cube can be reduced into nine groups, each of which consists of two alternative notations, which will be randomly selected on the basis of one or two previous notation to meet the constraints. Group selection of nine is the maximum number of possible groups in order to obtain the most optimum message capacity. One example of this grouping result can be seen in the

default encoding table (Table 2). This table is known to the sender and the receiver through a secure private channel and will be used in the generation and extraction of the cover.

Table 1. Rubik's Cube notation grouping based on the axis and the face.

Axis	Face	Notation
X	L	L, L2, L'
	R	R, R2, R'
Y	F	F, F2, F'
	B	B, B2, B'
Z	U	U, U2, U'
	D	D, D2, D'

Table 2 Example of default message encoding table based on the base-9 digit.

Base-9 Digit	Notation (axis) 1	Notation (axis) 2
0	L (X)	F (Y)
1	R (X)	B (Y)
2	U (Z)	L2 (X)
3	D (Z)	R2 (X)
4	F2 (Y)	U2 (Z)
5	B2 (Y)	D2 (Z)
6	L' (X)	F' (Y)
7	R' (X)	U' (Z)
8	B' (Y)	D' (Z)

However, sending a message with only the encoding table is not safe. This is because a user will always get the same scrambles before the message is changed. To ensure that the scrambles will be generated like a standard timer or scrambles generator, then embedding a message into a scramble should fulfill two requirements. Firstly, when generating a scramble, the generated notations must be always different regardless of the secret message is changed or not. Secondly, after all scrambles which contain a secret message are generated, more scrambles should be generated randomly as a normal scrambler. So, permutation information value – to shuffle the encoding table – and length information value – to decide where the scrambles will continue randomly – are used.

Each time a cover is generated, permutation information value will be randomly generated. This will ensure the cover is always different even if the message is the same.

Cover that will be generated is a collection of scrambles with first and second scramble as a header which contains permutation information value and length information value respectively. Then after the full message which embedded into scrambles are fully generated, normal scrambles will be generated as game preserving.

2.1 Embedding algorithm

The embedding process has three basic steps. Firstly, the permutation information is embedded in the first scramble. Secondly, the length of a secret message after being encoded into Rubik's Cube notation is embedded in the second scramble. Lastly, the third scramble and so on save the encoded message.

2.1.1 Embedding the permutation information

The algorithm to generate the first scramble (h_1) which embed a permutation information consists of the following steps:

- Step 1: Generate a permutation information (*P*) for the encoding table. *P* consists of nine unique digits that are one of permutation of a series of numbers 0 through 8.
- Step 2: Generate decimal number ($h_{1,10}$) as a header with the length of *n* around 20 (Fig. 3.a). Choose randomly the first digit *i* with a value in the range of 0-9. This number is used to determine the number of sub_1 digits (the length of random number digits to be filled before inserting *P)*. The last part sub_2 is inserted after *P*. Each digit of both *sub1* and *sub2* is random.
- Step 3: Convert $h_{1,10}$ into base-9 ($h_{1,9}$).
- Step 4: Convert each digit of $h_{1,9}$to the corresponding notation in default encoding table.

The value of *P* here will be used to permute the encoding table for next scrambles.

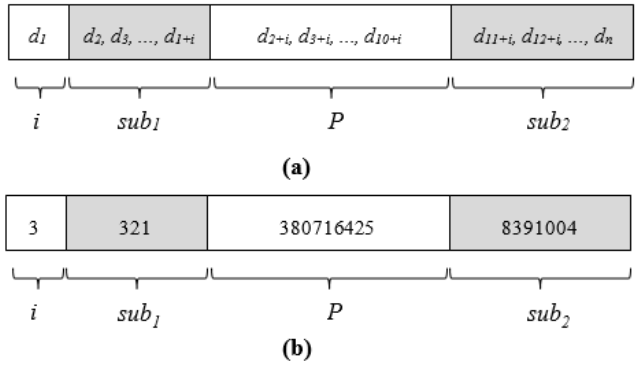

Fig. 3. (a) The header structure of a permutation information for the encoding table in decimal. (b) An example of a permutation information header in decimal.

2.1.2 Embedding the secret message

Because the second scramble has a length notation information, then the scrambles which embed the secret message (*M*) as a secret message notation (*m*) will be generated first. The algorithm to generate *m* consists of the following steps:

- Step 1: Convert each character from the secret message (*M*) into its corresponding binary string.
- Step 2: Merge all binary string into a long binary string.
- Step 3: Convert a long binary string into base-9 (m_9).
- Step 4: Convert each digit of m_9 into its corresponding notation in the permuted encoding table.

The result of *m* will be used for the third and later scramble.

2.1.3 Embedding the length information

The algorithm to generate the second scramble (h_2) which embed the length information consists of the following steps:

- Step 1: Get the value of the length information (len_m) which is the length of m.
- Step 2: Generate decimal number ($h_{2,10}$) as a header with the length of n around 20 (Fig. 4.a). Choose randomly first digit j with value in the range of 0-9. This number is used to determine the length of sub_3 (length of random number digits to between k and sub_3). k shows the length of len_m. The last part sub_4 is inserted after len_m. Each digit of both *sub3* and *sub4* is random.
- Step 3: Convert $h_{2,10}$ into base-9 ($h_{2,9}$).
- Step 4: Convert each digit of $h_{2,9}$ into its corresponding notation in the permuted encoding table.

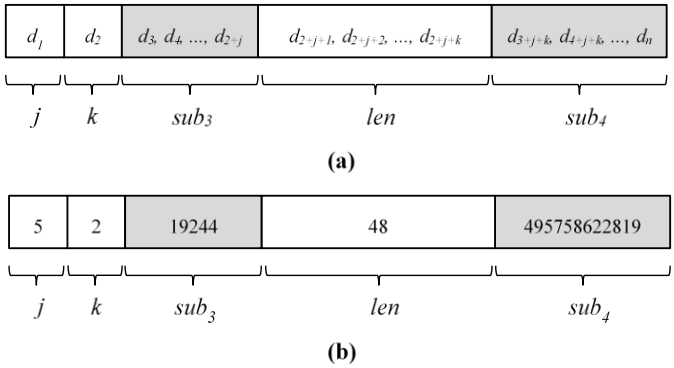

Fig. 4. (a) The header structure of the length information in decimal. (b) An example of the length header in decimal.

2.2 Extracting algorithm

From the generated cover, the receiver needs to extract the secret message.

2.2.1 Extracting the permutation information

The algorithm to extract P from the first scramble (h_1) consists of the following steps:

- Step 1: Convert h_1 into $h_{1,9}$ by converting each notation into its corresponding base-9 digit in default encoding table.
- Step 2: Convert $h_{1,9}$ into decimal ($h_{1,10}$).
- Step 3: Get the first digit of $h_{1,10}(i)$. By knowing i as the length of sub_1, the position and the value of P are obtained.
- Step 4: Use P to permute default encoding table.

The permuted notation table will be used to convert each notation for next steps.

2.2.2 Extracting the length information

The algorithm to extract len_m from the second scramble (h_2) consists of the following steps:

- Step 1: Convert h_2 into $h_{2,9}$ by converting each notation into its corresponding base-9 digit in the permuted encoding table.

- Step 2: Convert $h_{2,9}$ into decimal ($h_{2,10}$).
- Step 3: Get the first and second digit from $h_{2,10}$ as j and k respectively. Get the value of len_m at position $2+j+1$ to $2+j+k$.

2.2.3 Extracting the secret message

The algorithm to extract M consists of the following steps:

- Step 1: Get m by taking len_m notations starting from the third scramble.
- Step 2: Convert m into base-9 number m_9 by converting each notation into corresponding base-9 digit in the permuted encoding table.
- Step 3: Convert m_9 into binary (m_2).
- Step 4: Add padding bits in front of m_2 so it has the number of digits multiples of eight.
- Step 5: Partition m_2 for each 8-bit.
- Step 6: Convert each 8-bit into corresponding ASCII character.

4 Experimental results

4.1 Case study

The following case study describes the embedding process of a sample secret message. The first thing to do is generating the first scramble, which is a header that contains a permutation information value P, as follows:

- Step 1: The random generated P is 380716425. It means base-9 with value 3 in the default encoding table is changed to 0, base-9 with value 8 is changed to 1, and so on. The result of this permutation is shown in Table 3.

Table 3 Modified default message encoding table with 380716425 as permutation value.

Base-9 digit (before)	Base-9 digit (after)	Notation (axis) 1	Notation (axis) 2
3	0	D (Z)	R2 (X)
8	1	B’ (Y)	D’ (Z)
0	2	L (X)	F (Y)
7	3	R’ (X)	U’ (Z)
1	4	R (X)	B (Y)
6	5	L’ (X)	F’ (Y)
4	6	F2 (Y)	U2 (Z)
2	7	U (Z)	L2 (X)
5	8	B2 (Y)	D2 (Z)

- Step 2: The randomly generated value of i is 3. So, the first header in decimal ($h_{1,10}$) is *3xxx380716425xxxxxxx* (where each *x* is a random number). One possible value is $3321380716425839100 4_{10}$ (Fig. 3.b).
- Step 3: Convert $h_{1,10}$ to base-9 number $h_{1,9}$. Thus we get 265251284604176771037_9.
- Step 4: Convert each digit of $h_{1,9}$ into its corresponding notation we get

L2 F' B2 U D2 B U D' F2 L' F U2 R U' L' R' U' B F D U'

Steps to generate the third scramble and so on:

- Step 1: The plain text message (*M*) is "I love Rubik's Cube". The binary representation from *M* is

  01001001 00100000 01101100 01101111 01110110 01100101 00100000 01010010 01110101 01100010 01101001 01101011 00100111 01110011 00100000 01000011 01110101 01100010 01100101

- Step 2: Merge all binary string into one long binary string.

  $01001001001000000110110001101111011011001100101001000000101001001$
  $11010101100010011010010110101100100111011100110010000001000011011 1$
  01010110001001100101_2

- Step 3: Convert the merged binary string into base-9 (m_9) yields

  $226754185177501135448734617448603602068130335876_9$

- Step 4: After converting each digit of m_9 to its corresponding notation in the permuted encoding table, one possible series of notations (*m*) is

  F L U2 L2 F' B D' B2 L' B' U L2 F' R2 D' B' U' L' B R D2 L2 R' B F2 D' U R B D2 U2 R2 U' F2 R2 F D F2 B2 D' R' D R' U' F' B2 U F2

Steps to generate the second scramble:

- Step 1: The length of *m* or len_m is 48 notations.
- Step 2: The randomly generated value of *j* is 5. Because the length of len_m or *k* is 2. So, the second header in decimal ($h_{2,10}$) will be *52xxxxx48xxxxxxxxxxx* (where each *x* is random number). One possible value is

  $521924448495758622819_{10}$ (Fig. 4.b)

- Step 3: Convert $h_{2,10}$ to base-9 number $h_{2,9}$. Thus we get

  4683264454545052384656_9

- Step 4: Convert each digit of $h_{2,9}$ into its corresponding notation we get

  B U2 D2 R' F U2 B R L' B L' B L' D F' L U' B2 R F2 L' F2

Table 4. Generated scrambles with a message "I love Rubik's Cube".

No.	Scramble	Note
1	L2 F' B2 U D2 B U D' F2 L' F U2 R U' L' R' U' B F D U'	First header: contain permutation information value (*P*)
2	B U2 D2 R' F U2 B R L' B L' B L' D F' L U' B2 R F2 L' F2	Second header: contain length information (len_m)
3	F L U2 L2 F' B D' B2 L' B' U L2 F' R2 D' B' U' L' B R D2	Part of the secret message
4	L2 R' B F2 D' U R B D2 U2 R2 U' F2 R2 F D F2 B2 D' R' D	Part of the secret message
5	R' U' F' B2 U F2 *D R U L F U2 L2 D R B D' B' U L U'*	Part of the secret message with the last fifteen notations are generated randomly to preserve the game

From the generated scrambles (Table 4), the receiver will extract it from the first scramble to get *P*:

- Step 1: By converting the first scramble to its corresponding base-9 digit ($h_{1,9}$), we get
  $26525128460417677103 7_9$
- Step 2: Convert it into decimal ($h_{1,10}$).
  $33213807164258391004_{10}$
- Step 3: By knowing the structure of the first header, we get $i = 3$ and *P* is
  380716425
- Step 4: Use *P* to permute default encoding table.

Steps to get len_m:

- Step 1: By converting the second scramble to its corresponding base-9 digit ($h_{2,9}$), we get
  4683264454545052384656_9
- Step 2: Convert into decimal ($h_{2,10}$).
  $521924448495758622819_{10}$
- Step 3:We get the value of $j = 5$ and $k = 2$. Thus by taking the digit at position $2+j+1$ to $2+j+k$, we obtain the value of $len_m$is 48.

Steps to get the secret message (*M*):

- Step 1: Get *m* by taking 48 notations starting from the third scramble. We get
  F L U2 L2 F' B D' B2 L' B' U L2 F' R2 D' B' U' L' B R D2 L2 R' B F2 D' U R B D2 U2 R2 U' F2 R2 F D F2 B2 D' R' D R' U' F' B2 U F2
- Step 2: Convert it into a base-9 number (m_9). We obtain
  $22675418517750113544873461744860360206813033587 6_9$
- Step 3: Convert $m_9$into binary (m_2).
  $1001001001000000110110001101111011101100110010100100000010100100111010101100010011010010110101100100111011100110010000001000011011101010110001$
- Step 4: Add padding bits in front of m_2 so it has the number of digits multiples of eight.
  $0100100100100000011011000110111101110110011001010010000001010010011101010110001001101001011010110010011101110011001000000100001101110101011000100110010$
- Step 5: Partition m_2 for each 8-bit.
  01001001 00100000 01101100 01101111 01110110 01100101 00100000 01010010 01110101 01100010 01101001 01101011 00100111 01110011 00100000 01000011 01110101 01100010 01100101
- Step 6: By converting each 8-bit into the corresponding ASCII character we obtain a message “I love Rubik’s Cube”.

Beside text, by reading a file byte per byte, this proposed method also works well to any type of file. According to our experiments, the average capacity of the proposed algorithm – calculated by dividing the original file size by the number of generated scrambles – is 66 bits/scramble (higher than the easy level of Minesweeper, which is 17 bits/game) [4]. In other words, the capacity can reach up to 330 bits/game in a game of Rubik’s Cube that usually consists of five scrambles, which has a higher capacity per game than the hard level of Minesweeper (214 bits/game) [4].

4.2 Communication Protocol

Communication protocol explains how the cover will be delivered to the receiver without arousing suspicion against the enemy. Cover of a hidden message is a collection of scrambles. One possible step is that the sender implements a scrambler or a timer to measure the Rubik's Cube playing time in which it contains a random scrambler generator such as qqTimer (https://www.qqtimer.net) or csTimer (http://cstimer.net). From the implementation, the sender invites the receiver to play the Rubik's Cube with a scrambler or timer that has been embedded with a message by the sender.

Scrambles can also be sent with other alternatives such as sharing it in an online forum of the Rubik's Cube. However, too many scrambles might arouse an enemy's suspicion if the scrambles are posted without any clear reason. Therefore, the use of online forums as a communication protocol is more suitable for messages with small sizes (e.g. text).

4.3 Proposed simulator

To present the proposed method of noiseless steganography, there are two applications developed with Flask microframework. The first is the application used as a cover for a communication protocol, which is a web-based timer (Fig. 5), while the other is a decoder (Fig. 6) for extracting a message from text-based scrambles. The sender will embed the message/file via a secret page in the timer application.

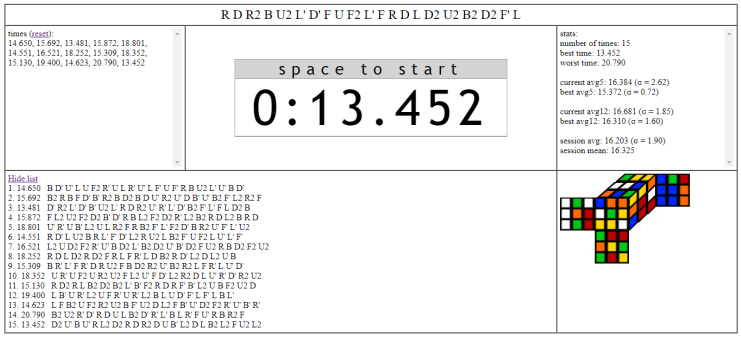

Fig. 5. The interface of the proposed simulator, which is a timer.

The timer is designed to record user solving times from the Rubik's Cube scrambles and to display statistics such as the best time, the lowest time, the average time, and other features of a standard timer for Rubik's Cube. After the receiver gets a collection of scrambles, he/she just need to copy the list of scrambles which is saved by application and paste it on the decoder to get the secret message.

Fig. 6. The interface of the decoder application.

5 Security discussion

5.1 Statistical Analysis

A secure steganographic scheme is statistically undetectable [3]. If the cover's parameter, which is Rubik's Cube notation, has similar distribution probabilities to that of a standard cover, then it prevents any suspicion from the enemy. One statistical approach is to calculate the value of entropy. In information theory, the value of entropy shows the degree of uncertainty in the system. Entropy is used to measure the degree of uncertainty of a random variable. Distribution similarities are tested by using entropy values, which measure the relationship between sample distributions. Let X be a discrete random variable with n possible samples $(x_1, x_2, ..., x_n)$. The value of entropy is defined as follows:

$$H(X) = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i) \quad (1)$$

$p(x_i)$ denotes the probability of x_i. If X is a uniform distribution, then the entropy value is the maximum of $log_2 n$ [11]. If the value of entropy approaches the maximum possible value, then the degree of uncertainty is higher, in other words, steganographic results are more secure from attack.

5.2 Experiments

Scrambles generated from the proposed method are tested statistically by comparing the results with other existing application, which is cTimer, csTimer, Cubemania, and qqTimer. Those four applications are Rubik's Cube timer that generates scrambles randomly. For each application, 10,000 scrambles were taken for comparison. Then, those results were compared to almost the same number of Rubikstega's scrambles, which generated from an image file that is embedded in a cover, including two header-scrambles.

The probability of each notation occurrence if the distribution is uniform is *100%/18=5.556%*. Based on the test results in Fig. 7, it can be seen that the probability value of Rubikstega is near 5.556%. The probability value of each notation of the proposed method can also be within the range of other timer software, which means it is between the upper and lower bounds.

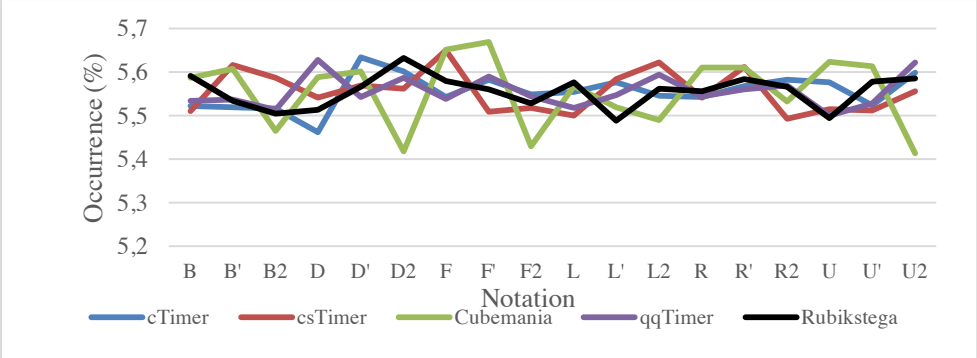

Fig. 7. The probability of occurrence each notation in different Rubik's Cube timer.

The calculation of entropy with the formula that is written on the previous subsection also gets a good result. With a maximum entropy value of $log_2 18$ or *4.169925*, Rubikstega has a value of *4,169893,* which is very close to the maximum possible value. The value is even above three other timer application (cTimer, csTimer, and, cubeTimer). In other words, the proposed Rubik's Cube timer shows that this noiseless steganography method appears similar to other standard Rubik's Cube timer.

Table 5. The entropy values in different Rubik's Cube application.

No.	Application	Entropy Value
1	cTimer	4.169890
2	csTimer	4.169875
3	Cubemania	4.169779
4	qqTimer	4.169896
5	Rubikstega	4.169893

6 Conclusion

In this paper, a novel noiseless steganography on Rubik's Cube named Rubikstega has been presented. It developed by using scrambles notation to embed the message. From 18 unique notations in 3x3x3 Rubik's Cube, it is divided into nine notation groups with each group having a corresponding base-9 value. According to the theoretical proof and experimental results, the proposed method is indistinguishable from other applications and has better per-game-capacity with other noiseless steganography approaches on the game domain. Based on the proposed scenario, extracting message is easy, the recipient just needs to collect the scrambles and copy it to the decoder. Furthermore, with the approach of using notations as a parameter, Rubikstega can be developed for other Rubik's Cube dimension, such as 2x2x2, 4x4x4, 5x5x5, etc. It can also be done in other puzzle games that utilize notations like Megaminx, Pyraminx, and Skewb.

References

[1] A. Desoky, M. Younis, Sec. Comm. Netw., *Chestega: Chess Steganography Methodology*, **2**, 555-566 (2009)

[2] A. Desoky, *Noiseless Steganography - The Key to Covert Communication*, (CRC Press, Boca Raton, 2012)

[3] I. J. Cox, M. L. Miller, J. A. Bloom, J. Fridrich dan T. Kalker, "*Digital Watermarking and Steganography, second ed.*, 54 (Morgan Kaufmann, Burlington, 2008)

[4] S. Mahato, D. K. Yadav, D. A. Khan, JISA, *A Minesweeper Game-Based Steganography Scheme,* **32**, 1-14 (2017)

[5] Z. Ou dan L. Chen, Inf. Syst., *A Steganographic Method Based on Tetris Games*, **276**, 343-353 (2013)

[6] T. Rokicki, H. Kociemba, M. Davidson and D. John, SIAM, *The Diameter of The Rubik's Cube Group is Twenty*, **27**, 1082-1105 (2013)

[7] D. Singmaster, *Notes on Rubik's Magic Cube* (Enslow Pub Inc, Berkeley Heights, 1981)

[8] https://www.cubenama.com/beginner

[9] *https://www.speedsolving.com/wiki/index.php/Half_Turn_Metric*

[10] *https://www.iberorubik.com/tutoriales/3x3x3/notation*

[11] http://www.math.uconn.edu/~kconrad/blurbs/analysis/entropypost.pdf