Maze Based Data Hiding Using Back Tracker Algorithm

T.Sukumar[1], Dr.K.R.Santha[2]
Department of IT, SVCE, Anna University, India
Department of EEE, SVCE, Anna University, India

I INTRODUCTION

Steganography is an art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity. A word steganography is of Greek origin and means "concealed writing" from the Greek word stegano means "covered or protected", and graphic means "writing". classically, the hidden message may be an invisible link between the visible lines of a private letter [1] and [2].

Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a transport layer, such as a document file, image file, program or protocol. Media files are ideal for steganographic transmission because of their large size. As a simple example, a sender might start with an innocent image file and adjust the color of every 100th pixel to correspond to a letter in the alphabet, a change so subtle that someone not specifically looking for it is unlikely to notice it [2].

A maze (See Fig 2.1) basically contains cells, walls, a starting cell and an end cell. Logically, a maze is a puzzle with complex multipath network, and a player is to find a solution path from the Starting cell to the end cell. A rectangular maze has m cells in width and n cells in height and is denoted as m x n maze, it is called perfect if there exists one and only one path between any two cells [1].

II PREVIOUS WORKS

A. *Maze Generation*

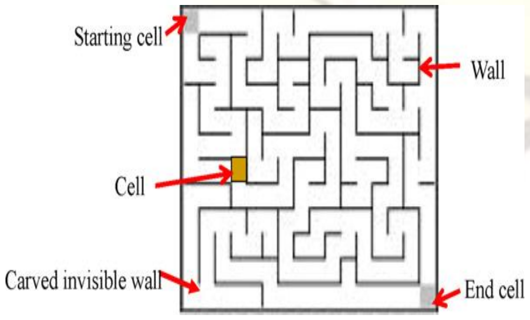


Fig.2.1. An example to illustrate a maze structure

B. *Perfect Maze And An Imperfect Maze*

Figs. 2.2 and 2.3 show a perfect maze and an imperfect maze, respectively. From our observation, most puzzles appearing in websites are rectangle and perfect. For security consideration, the created maze should look like a common one hence; here we only deal with rectangular perfect mazes. Regarding cells as nodes, carved invisible walls as links.

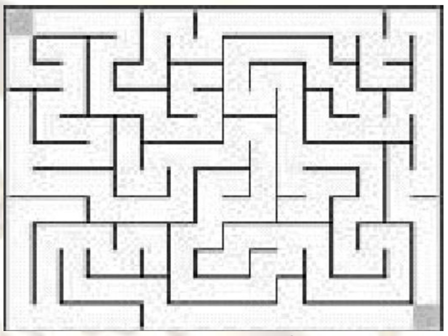


Fig.2.2. A 16×12 perfect maze

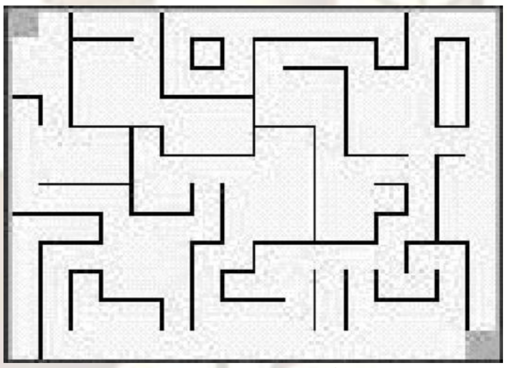


Fig.2.3. A 16×12 imperfect maze

C. *Expressing Maze As A Graph*

We can express a maze as a graph. Fig.2.4 shows an example, a number attached to a link connecting two nodes S and E.

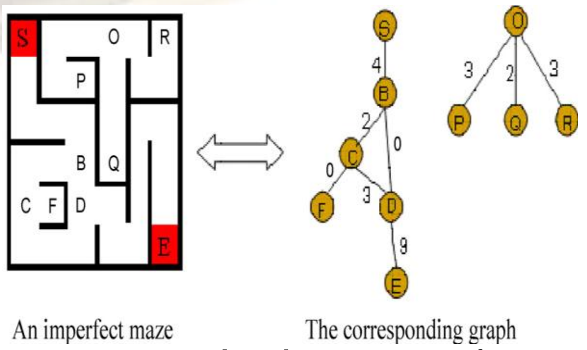


Fig.2.4 Correspondence between an imperfect maze and a graph

Based on this representation, we can find that a perfect maze corresponds to a tree (see Fig. 2.5), such relation can be used to prove whether a generated maze is perfect.

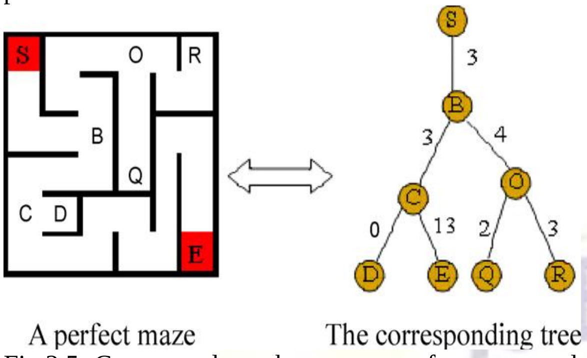


Fig.2.5. Correspondence between a perfect maze and a tree.

D. *Existing Approaches*

In this section, we will first introduce a typical maze generator, then a steganographic method based on this generator is described.

Hunt-and-Kill maze generating (HKMG) algorithm

There are a number of maze generating algorithms, Hunt-and-Kill maze generating algorithm is typical. The HKMG algorithm generates a maze by carving walls. Fig. 2.6 shows a maze generated by the HKMG algorithm.

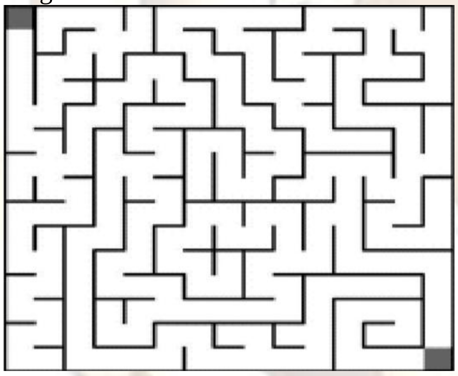


Fig 2.6 An maze generated by HKMG algorithm.

HKMG algorithm, there are three types of cells defined as follows:

• 'In' cell (I): a cell that has been processed and always keeps its type.

• 'Frontier' cell (F): a cell that is processed and is a 4-neighbor of a certain "I" cell.

• 'Out' cell (O): a cell not yet processed.

The HKMG algorithm described as follows

1. Mark all cells as O cells.
2. Mark the starting cell as I cell, and mark each O cell in the 4-neighborhood of the I cell as F cell.
3. Choose an F cell around an I cell, and carve the wall between the F cell and the I cell. Mark the F cell as I cell, and mark each O cell in the 4-neighborhood of the I cell as F cell. Repeat step 3 until there is no F cell.
4. End

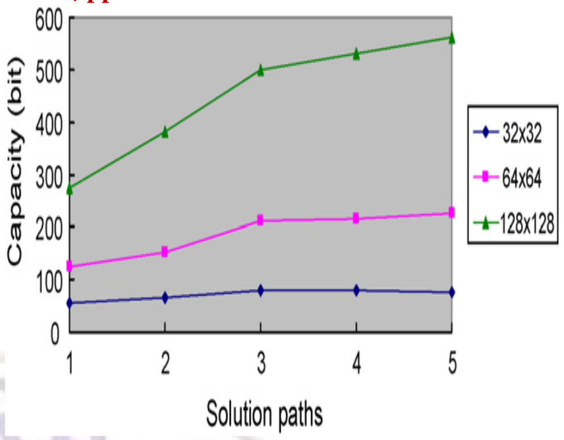


Fig 2.7 the capacities of different sizes of mazes.

The main idea of the existing method is to consider multipath rather than only the solution path to gain more embedding capacity. Before Describing the proposed method, we will define "embeddable cell" which will be used to embed a bit. Suppose the HKMG algorithm is used to generate a perfect maze, and the solution path from the starting cell to the end cell is located. All cells on the path are marked as I cells, the other cells are reset to be O cells, and all walls are rebuilt, except those in the path (see Fig.2.8).

Definition 1. An embeddable cell, A, is an I cell, which is in the solution path with exact two O cells in its 4-neighbors, and each O cell should not be a neighbor of another embeddable cell which appears before A. Fig. 2.8 shows an example. In Fig. 2.8(b), the cells with black triangles are embeddable ones and the gray cell is an O cell and is the overlapped neighbor of two I cells, thus the cell with a black solid circle is not an embeddable cell. Based on the definition, all embeddable cells can be located. According to the embedding bit, we carve the wall between an embeddable cell and one O cell around it, and mark the O cell as I cell. There are six kinds of embeddable cells shown in Fig. 2.9.

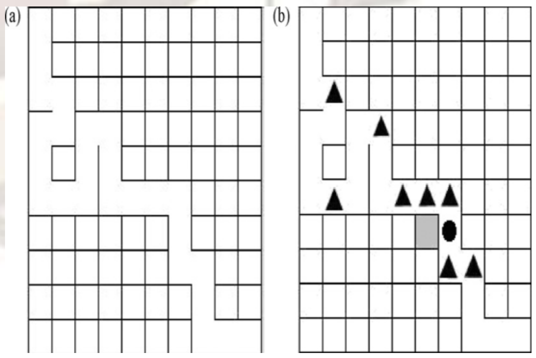


Fig. 2.8. One example to illustrate embeddable cells. (a) The white path stands for the solution path. (b) Embeddable cells marked by black triangles.

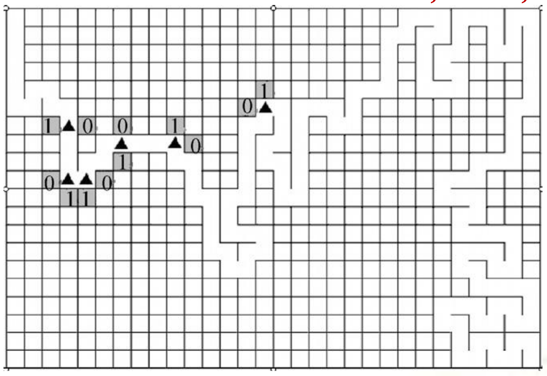


Fig. 2.9 Six kinds of embeddable cells.

Each embeddable cell has two neighboring O cells marked as 1 and 0. If a “1” bit is embedded, then the wall between the embeddable cell and the cell marked “1” is carved, and the “1” cell is marked as I cell. Otherwise the wall between the embeddable cell and the cell marked as “0” is carved, and the “0” cell is marked as I cell. To increase embedding capacity, we can embed bits into multipath instead of only one path. In the existing method, we first generate a perfect maze with HKMG algorithm. Subsequently, we choose some cells as the start cells and one cell as the common end one of the multipaths. Finally, we solve the perfect maze to obtain the corresponding multipath. This multipath sometimes will merge at some cells. The two solution paths start at different cells respectively they should have the common end cell E. The second path from maze is merged into the first path from S to E at cell A as shown in the figure Fig 2.10(d). Note that the number of solution paths and all the start cells and one end cell are chosen through a random number generator and a seed, which will be considered as the secret key [5]&[6].

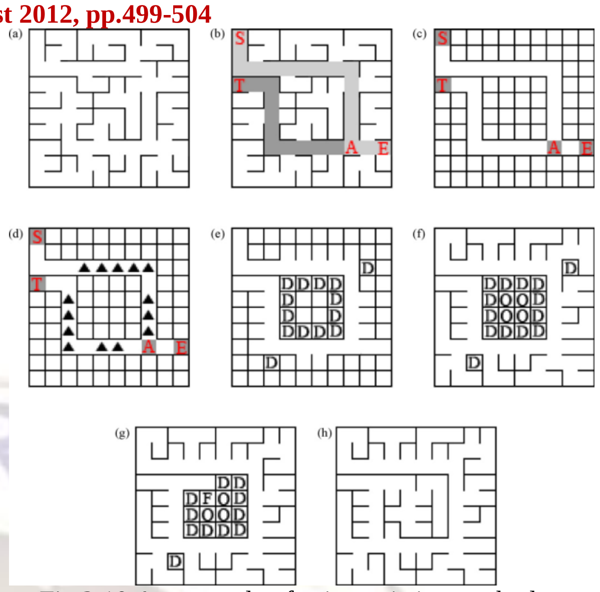
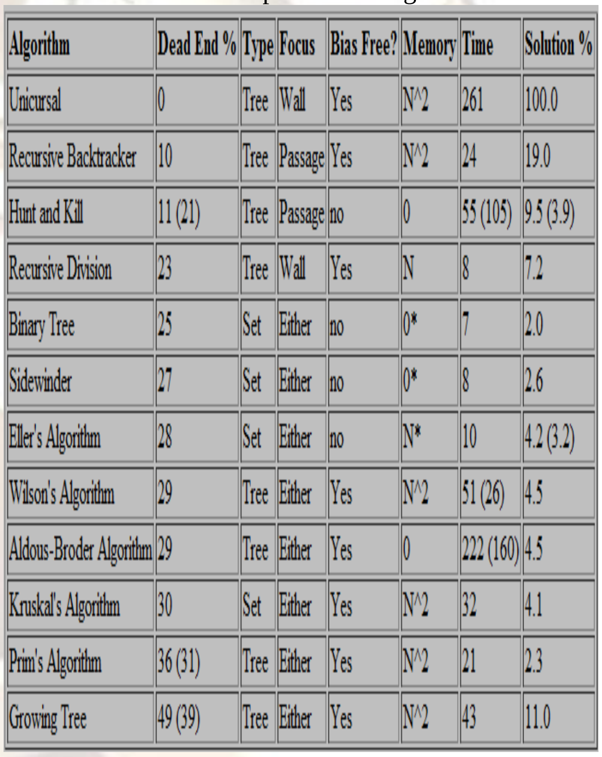


Fig.2.10 An example of using existing method to embed data in a perfect maze

(a) A perfect maze generated by HKMG algorithm.
(b) Two paths from S to E and T to E in (a) located with merged points A.
(c) The result after performing Step 3 of the proposed embedding algorithm to (b).
(d) All embeddable cells located.
(e) The result after embedding data in embeddable cells with D cells marked.
(f) The result of applying HKMG algorithm to process F cells.
(g) The immediate result of applying Step 8 of the proposed embedding algorithm to (f).
(h) The perfect maze generated after processing all Ds.

Table 3.1 Comparison of Algorithms



Algorithm	Dead End %	Type	Focus	Bias Free?	Memory	Time	Solution %
Unicursal	0	Tree	Wall	Yes	N^2	261	100.0
Recursive Backtracker	10	Tree	Passage	Yes	N^2	24	19.0
Hunt and Kill	11 (21)	Tree	Passage	no	0	55 (105)	9.5 (3.9)
Recursive Division	23	Tree	Wall	Yes	N	8	7.2
Binary Tree	25	Set	Either	no	0*	7	2.0
Sidewinder	27	Set	Either	no	0*	8	2.6
Eller's Algorithm	28	Set	Either	no	N*	10	4.2 (3.2)
Wilson's Algorithm	29	Tree	Either	Yes	N^2	51 (26)	4.5
Aldous-Broder Algorithm	29	Tree	Either	Yes	0	222 (160)	4.5
Kruskal's Algorithm	30	Set	Either	Yes	N^2	32	4.1
Prim's Algorithm	36 (31)	Tree	Either	Yes	N^2	21	2.3
Growing Tree	49 (39)	Tree	Either	Yes	N^2	43	11.0

III IMPLEMENTATION

The main idea of the proposed method is to consider SOLUTION PATH rather than only MULTI PATH to gain more embedding capacity. Before define the proposed work., we first discuss some available list of maze generation algorithms, that meet out requirement to gain more embedding capacity.

From the above table, Recursive Backtracker is the algorithm gives higher river factor comparatively from Hunt and Kill algorithm. Recursive backtracker gives larger solution path than the HKMG, and also the dead end gets decreased by one percent , by this for the same size of maze that with same start cell and same end cell by Recursive backtracker gives higher embeddable cell than by HKMG.

E. Embedding Algorithm

The details of the proposed embedding algorithm are described as follows:

1. Creating a maze using the Recursive Backtracker algorithm.

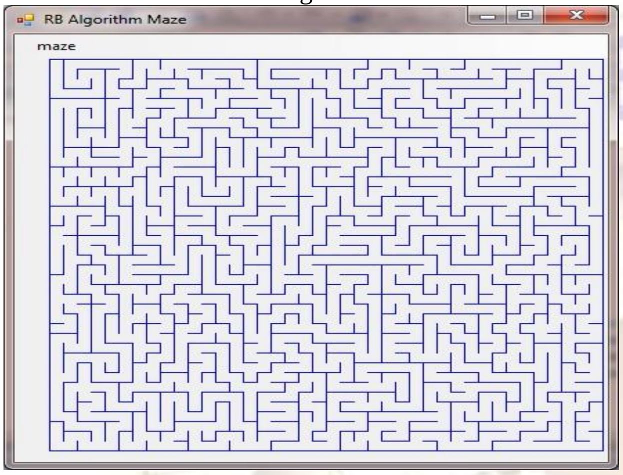


Fig 3.1Generated perfect maze by recursive backtracker

2. Choose one cell as start cells and one cell as the end cell. Solve solution path from these starting cells to the end cell
3. Reset all cells to be O cells and all walls as visible. Set those cells On solution path to be I cells. Carve each wall between two I cells

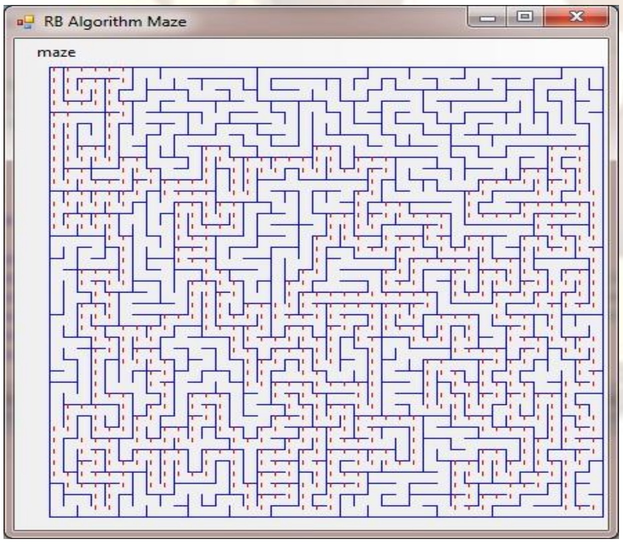


Fig 3.2 After finding the solution path

4. Find all embeddable cells in solution path and order them according to the path sequence. Note that we do not set the cell on boundary to be embeddable cell even if the cell is embeddable.

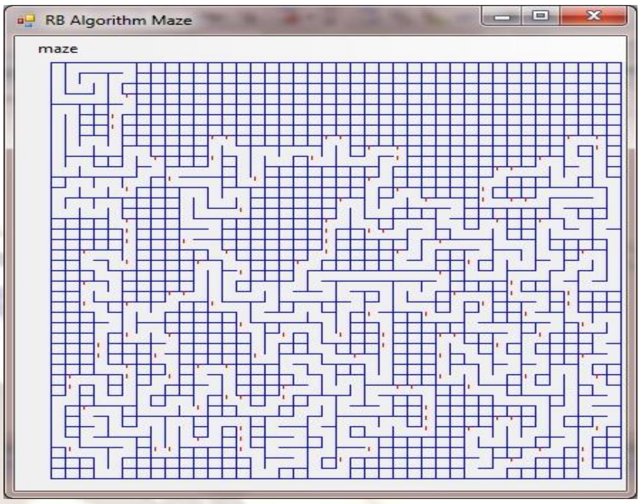


Fig 3.3 After finding the embeddable cells

5. For each embeddable cell, if the embedding bit is 1 (0), the wall between the embeddable cell and its neighboring O cell marked 1 (0) is carved and the cell marked 1 (0) is set as I cell, the other neighboring O cell marked 0 (1) is set as D cell (see D stands for the D cell).Then write the binary representation of the processed cell to a file
6. Set those O cells around I cells to be F cells.
7. Process these F cells using recursive backtracker algorithm, O stands for the O cell.

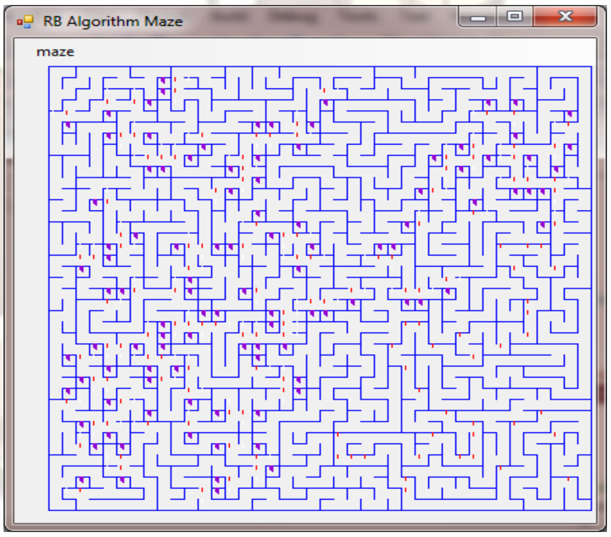


Fig 3.4 After embedding with d cell remains unprocessed

8. Process the D cells.(a) Scan the maze, and check if any D cell exists. If none, go to step 9. Otherwise, choose a D cell with one of its neighbors being an un-embeddable I cell and carve the wall between the D cell and the un-embeddable I cell. Mark the D cell as I cell, and mark each O cell in the 4-

neighborhood of the I cell as F cell (b) Check if any F cell exists. If yes, go to step 7. If none, go to step 8.

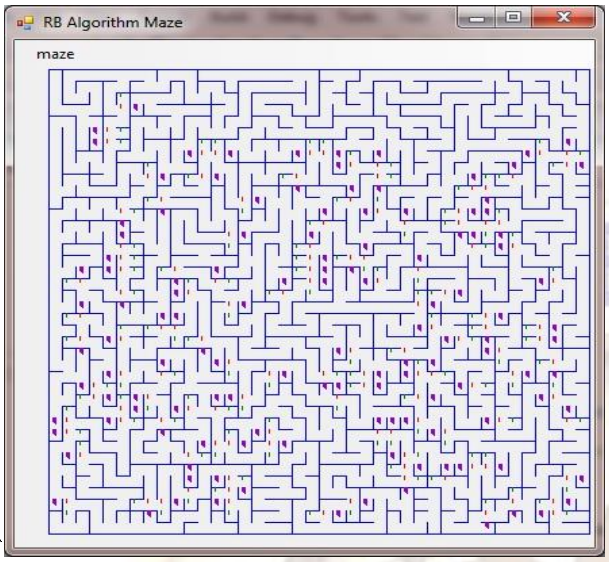


Fig 3.5 After processing of all cells with Marking

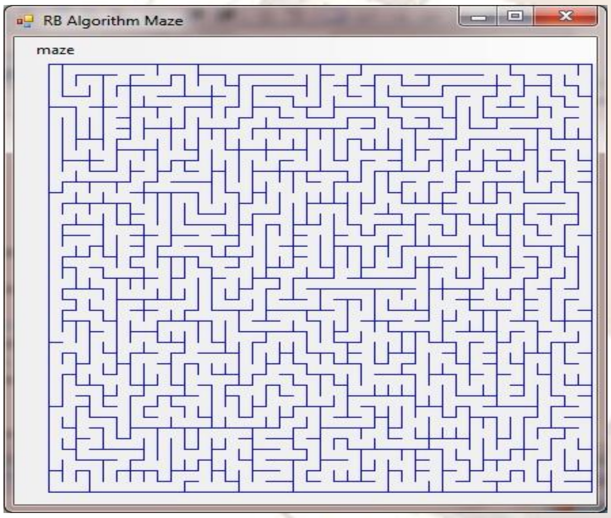


Fig 3.6 After processing of all cells without Marking

9. End

F. *Extraction Algorithm*

To extract the embedded bits, the receiver must know the start and end cells of solution path which can be extracted through the secret key and a pseudo random number generator. Since the generated maze is perfect, the receiver can get the original solution path accurately according to these start and end cells. Then, the receiver can locate all embeddable cells. Then use the Binary Representation of the maze to redraw the maze as in the sender side finally according to the direction of the branch in each embeddable cell, secret data can be extracted successfully.

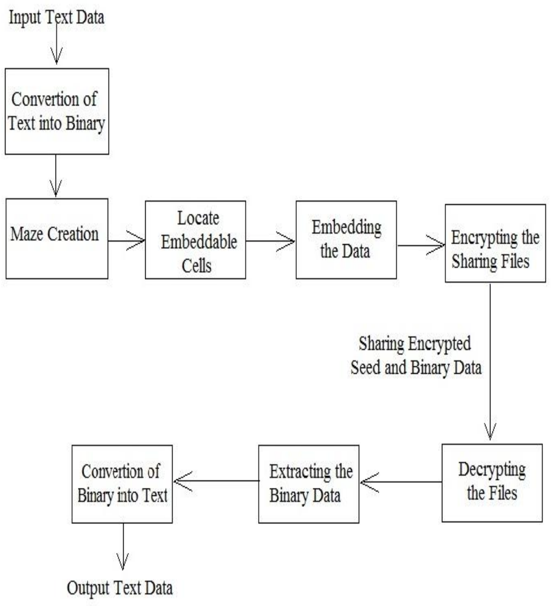

Input Text Data
Convertion of Text into Binary
Maze Creation
Locate Embeddable Cells
Embedding the Data
Encrypting the Sharing Files
Sharing Encrypted Seed and Binary Data
Decrypting the Files
Extracting the Binary Data
Convertion of Binary into Text
Output Text Data

Fig 3.7 Stegomaze Architecture

G. *Maze Generation*

Basically a maze is a collection of cells., so first generate the cells upto mentioned maze size. then address each cell to identify them. generating different mazes at every time is our objective here., so we use a random number generator and seed to achieve this. the random number generator generates the unique pattern of results for unique seed., by using the pattern the corresponding unique maze gets generated. hence the same maze will be generated for same seed any number of times.

H. *Locate Embeddable Cells*

After the maze gets generated, we first find the solution path where the secret data to be embed. For that we have to feed the starting cell and the end cell. From the start cell, find the next cell from its four neighbors and there is no wall between the current cell and next cell. Process till it reaches the end cell. Push the solution cells in a separate stack.

Rebuild the walls of all cells except the cells from the solution path. Mark the cells in the solution path as embeddable cells if the cell contains two neighbors and the neighbors should not be a neighbor of any other cell. Push the embeddable cells in a separate stack.

I. *Embedding Data*

The embeddable cells are used to embed the secret data. Using the random number generator, mark one of the neighbor for zero bit and the another for one bit. the secret data should be converted into binary form. the binary data is to be embed into the maze. if the zero bit is to be embed, carve the walls between the embeddable cell and its zero bit , else carve the walls between the embeddable cell and its one bit. after embedding , the binary representation of the embeddable cells is stored in the file. After that process all the remaining cells so that it looks like a maze.

J. *Encrypting File*

The stored binary representation of the maze is then encrypted using DES encryption standard. Read the 8 bit key from the user and by using that encrypt the file. Finally the encrypted file and the key alone is shared with the ultimate receiver.

K. *Decrypting File And Extract T Original Data*

By using the same secret key and the encrypted file as input decrypt using same DES and get the binary representation of the maze. By using the seed from the decrypted file, the original maze gets generated. Then using the binary representation of the maze, the original maze after embedding will be regenerated. Then use the start cell and end cell to find the solution path. Then find the embeddable cells, and depends on the branching on every embeddable cells, the binary bits gets extracted. Then the extracted binary data is converted to ASCII value to get the character equivalent of the binary. Thus the original data can be extracted successfully.

IV CONCLUSION

We proposed an enhancement in existing method to embed secret data into mazes. It generates perfect mazes that cannot be distinguished visually by humans from other perfect mazes commonly used. Hence, it significantly improves security over the existing HKMG embedding algorithm by providing encryption to sharing Data. When embedding bits into one solution path our new method provides approximately twice the embedding capacity of the one-path HKMG algorithm.

FUTURE ENHANCEMENT

With larger mazes our algorithm can utilize multiple paths for embedding to further increase capacity, while maintaining its “undetectability” from human vision. Data compression can be used to increase the embedding capacity. In this project we are working with maze, in future other puzzle games can be used for hiding the Data.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valued comments which helped to improve the manuscript.

REFERENCES

[1] Hui-Lung Lee, Chia-Feng Lee, Ling-Hwei Chen,”*A perfect maze based steganographic method*” The Journal of Systems and Software, August 2010

[2] Zhan-He Ou; Ling-Hwei Chen; , "*Hiding data in Tetris*," Machine Learning and Cybernetics (ICMLC), 2011 IEEE Transactions on , vol.1, no., pp.61-67, 10-13 July 2011

[3] Hopper, N.; von Ahn, L.; Langford, J.; , "*Provably Secure Steganography*," Computers, IEEE Transactions on , vol.58, no.5, pp.662-676, May 2009

[4] Artz, D."*Digital steganography: hiding data within data*," Internet Computing, IEEE , vol.5, no.3, pp.75-80, May 2007

[5] Andrew D. Ker "*Steganalysis of Embedding in Two Least-Significant Bits*," Information Forensics and Security, IEEE Transactions on , vol.2, no.1, pp.46-54, March 2007

[6] Anderson, R.J.; Petitcolas, F.A.P.; , "*On the limits of steganography*," Selected Areas in Communications, IEEE Journal on , vol.16, no.4, pp.474-481, May 2006

[7] http://www.msdn.com

[8] http://www.social.msdn.com

[9] http://www.dotnetpearls.com

[10] http://www.stackoverflow.com