Covert Channels in Online Rogue-like Games

Hasnain Lakhani [1]
Computer Science Laboratory
SRI International
Menlo Park, CA, United States 94025
Email: hasnain.lakhani@sri.com

Fareed Zaffar
School of Science and Engineering
Lahore University of Management Sciences
Lahore, Pakistan 54792
Email: fareed.zaffar@lums.edu.pk

***Abstract*—Covert channels allow two parties to exchange secret data in the presence of adversaries without disclosing the fact that there is any secret data in their communications. We propose and implement EEDGE, an improved method for steganography in mazes that builds upon the work done by Lee et al; and has a significantly higher embedding capacity. We apply EEDGE to the setting of online rogue-like games, which have randomly generated mazes as the levels for players; and show that this can be used to successfully create an efficient, error-free, high bit-rate covert channel.**

I. Introduction

In recent times, with the advent of social media and the rise of censorship in certain countries, it has become even more important for people to communicate securely without detection. People may wish to send certain messages while hiding their communications from others. Using encryption is not enough; sometimes the mere presence of an encrypted transmission is enough to arouse suspicion. Something stronger is needed: a means of communication that does not arouse suspicion while conveying a message that may be suspicious.

A covert channel is the use of a regular (overt) communication channel to communicate hidden data (covert) without being detected by any other parties who are observing the flow of information over the channel. A variety of covert channels have been proposed in the literature; based on foundations ranging from network protocols [11] to networked online games [8]. Network protocol based schemes can be fairly complex, however they generally tend to exhibit bit errors and do not have very high bit rates.

Steganography, closely related to covert channels, is the study of techniques to hide secret data in messages without detection. A lot of the literature is focused on steganography in digital images [1]. If we can hide data in a certain type of message, and use a channel along which communication of these type of messages is not suspicious, then we can easily have a demonstrable covert channel.

We propose EEDGE, which is a scheme for steganography in mazes. EEDGE builds upon the work by Lee et al [6]. Mazes depart from the traditional mediums for steganography, but we can use them to build stegosystems that are error-free and have a good embedding capacity. EEDGE has a high bit rate, and, more importantly, it is error-free; unlike some image steganography techniques which suffer from bit errors [1].

EEDGE hides data by marking certain edges in a maze based on a shared key; and then choosing to keep or remove each of those edges based on bits read from the secret data that is being hidden. The mazes that are transmitted are perfect mazes and are visually indistinguishable from random mazes. The receiver can figure out which edges were kept or removed and thus extract the hidden message.

To successfully create a covert channel based on EEDGE, we need to find a channel in which mazes are regularly sent so that sending a maze will not be considered suspicious activity. Fortunately for us, online games come to the rescue. Rogue [7] is a game that is focused around the concept of players exploring randomly generated maze-like levels. We use online multi-player versions of Rogue as the overt channel.

The rest of the paper is organised as follows. Section III covers EEDGE. Section IV describes how EEDGE can be applied to create a covert channel in online rogue-likes. Section V has an experimental evaluation of our system. Section II reviews and discusses related work on covert channels, online rogue-likes, and steganography. The paper concludes in Section VI with a discussion of future work.

II. Background

A. Covert Channels

A detailed treatment of network covert channels can be found in [11]. Simmons' Prisoner Problem [10] is the standard example of a situation where a covert channel is necessary. Alice and Bob, both prisoners, need to communicate in order to devise a plan of escape. However, Wendy the warden is listening to all their messages; and if she suspects any hidden communication, Alice and Bob will be put into solitary confinement. Thus Alice and Bob must exchange hidden information (a covert channel) over a channel full of unsuspicious messages (overt channel); such that Wendy can not notice what they are doing.

In general, Alice and Bob use two networked computers to communicate, exchanging normal, unsuspicious network traffic inside of which hidden data is present. The traffic may be routed through many intermediate nodes (Wendy's) who should not be aware of the existence of the covert channel.

B. Covert Channels in Games

Murdoch et al. in [8] propose a covert channel for collusion in an online connect-4 contest (a single human player could enter multiple programs in this contest). They developed a covert channel by creating two types of players; foxes and chickens. A fox would play the best strategy possible; while

[1] Work done while at LUMS.

chickens would intentionally lose to foxes but play their best against other players. They used redundancy in the moves of the game as their covert channel which allowed chickens to detect foxes so that they could collude to win.

Hernandez-Castro et al. propose a game-theoretic approach to steganography in games in [3]. They propose a method where the secret data to be hidden is encoded in the strategy a game player picks. They provide an implementation of their covert channel in a game of Go and analyzed the effectiveness of various steganalytic techniques against their scheme.

Both of the above schemes are mostly intended for collusion between players in order to win a game. Zander et al. propose a novel covert channel in First Person Shooter games in [12]. They encode covert bits as small, visually imperceptible, modifications of a character's movements. Their channel is noisy and has a low bit rate, but it is extremely difficult to detect the covert channel and block it; unless one wishes to block all network game traffic entirely.

Our scheme differs from all three in that it focuses on the game level itself as the medium in which secret data is sent.

C. Online Rogue-likes

Rogue [7] is a game in which the player plays the role of a fantasy adventurer exploring a large dungeon with many levels. Each level, randomly generated, is a collection of rooms connected by maze-like pathways; and has one stairway leading to the level below. The player's goal is to defeat the monsters in each level, gain treasure, and eventually get to the bottom of the dungeon. The term *Rogue-like* refers to a class of games which were inspired by Rogue and share the same basic game mechanics. There are multiplayer variants as well, including TOMENET, which allows multiple players to play on the same level and fight each other.

For our purposes, rogue-likes are important as the levels are generally maze-like, though they may contain cycles. It is not out of the ordinary to make modifications to the level generation algorithm so that levels are always mazes.

D. Steganography

Steganography is a general technique, closely related to covert channels, which aims to allow senders to embed data in messages. The general difference between a covert channel and a steganographic message is that covert channels tend to require both parties to be online and in contact; while steganography is more like a publish-subscribe service. The sender can hide data inside a message and the receiver can receive it later on without necessarily having to communicate directly with the sender. For example, the previously mentioned covert channels require both parties to be actively playing the game; while with steganography one could publish an image to a photo sharing service and the receiver could later retrieve it from there.

A lot of the steganographic techniques in the literature revolve around images [1]. Most of these techniques have a high capacity, however bit-error is a problem (due to compression artefacts) and there are steganalytic techniques that allow detection of the hidden message.

E. Steganography with Mazes

Lee et al [6] proposed a steganographic scheme which was focused on using mazes as the underlying medium, instead of images. Their technique relied on the communication of a maze with a certain identifiable path (which could be derived by parties knowing the key). The sender would remove certain edges from this path, based on the secret data it wished to send; and the receiver could identify the removed edges and thus recover the message. Their scheme focused on marking certain vertices along the path, and then, for each cell, keeping one of two edges based on the value of the bit in the message. The main difference between their scheme and EEDGE is in the embedding step; EEDGE uses an improved embedding scheme to allow higher data rates. This difference is explained in Section III.

F. Graph Classification

The graph classification problem is a task in which individual graphs (from a larger database) need to be classified into two or more categories, based on certain features of the graphs. Several approaches for graph classification have been proposed; ranging from frequent subgraph mining as in SubdueCL [2], to evolutionary computation such as in GAIA [4], to applying Boosting with naive classifiers such as in gBoost [5]. Classifiers are relevant to EEDGE since mazes can be represented as graphs. It would be important to ensure that classifiers are unable to distinguish normal mazes from mazes containing hidden data.

III. EEDGE

A. Definitions

Let M be the set of valid bit strings in some channel. A stegosystem consists of the following set of algorithms [13]:

- COVER generates a random cover message $c \in M$.
- EMBED takes a cover message $c \in M$, some secret data d, a key k, and generates a hiddentext $h \in M$.
- EXTRACT takes a hiddentext h, a key k, and outputs the secret data d.

A stegosystem is said to be *secure* if an attacker can not distinguish the output of EMBED from any valid message in the channel; that is, any attacker should not be able to figure out whether a message has data hidden inside it.

Our stegosystem, EEDGE, is based on the scheme proposed by Lee et al in [6]. We propose a scheme that is more general, and has a higher embedding capacity. Their scheme focuses on using *embeddable cells*, while we focus on *embeddable edges*; and as mazes have more edges than vertices, we can embed more data.

Let EV denote the set of *embeddable vertices* (either one-embeddable or two-embeddable) and EE denote the set of *embeddable edges*.

1) A **maze** M of size $w \times h$ is a minimal spanning tree of an undirected graph $G(V, E)$ where each vertex $v \in V$ is a point on a $w \times h$ lattice.

2) A vertex v is a **neighbour** of a vertex v' if they represent adjacent points on the lattice.
3) A **hidden tree** H is a connected tree which is a subtree of a maze M.
4) A vertex $v \in H$ is **one-embeddable** if it has exactly one neighbour $n \notin H$, such that there is no vertex $v' \neq v$, $v' \in EV$ such that v' is a neighbour of n.
5) A vertex $v \in H$ is **two-embeddable** if it has exactly two neighbours $n_1, n_2 \notin H$, such that there is no vertex $v' \neq v$, $v' \in EV$ where v' is a neighbour of n_1 or n_2.
6) An edge (v_1, v_2) is an **embeddable edge** if $v_1 \in EV$ and $v_2 \notin H$.
7) The **key** k of a hidden tree H is the set of leaf vertices of H, i.e vertices with exactly one edge.

From these definitions, we can see that a one-embeddable vertex will have one embeddable edge, while a two-embeddable vertex will have two. Note that our definition of a two-embeddable vertex is identical to the definition of an *embeddable cell* as used in [6]. They store one bit per embeddable cell, while we can store one bit per embeddable *edge*, i.e. two bits per two-embeddable vertex and one bit per one-embeddable vertex.

For a stegosystem, the COVER, EMBED, and EXTRACT algorithms need to be defined. We also define a few others:

- COVER returns a random maze M.
- HIDDEN-TREE takes a maze M, a key k, and outputs the hidden tree H corresponding to k.
- FIND-EMBEDDABLE-EDGES takes a maze M, a key k, and returns the embeddable edges in M corresponding to k.
- REGEN takes a graph $G(V, E)$, a set of edges to keep KE, and a set of edges to discard DE, and returns a maze M such that all edges in KE are in M and no edge in DE is in M.
- EMBED takes a maze M, a stream of secret bits d, a key k, and returns a hiddentext maze H.
- EXTRACT takes a hiddentext maze H, a key k, and outputs the secret data d.

B. Illustration

We will first illustrate the working of our algorithms graphically, before formally describing them and proving their correctness. This will provide intuition for the following sections. Note that the illustration follows the order of the algorithms being called. In the actual implementation, the users of EEDGE only call COVER, EMBED, and EXTRACT.

Firstly, the sender runs the COVER algorithm to generate a random cover maze M on a $w \times h$ lattice, as illustrated in Figure 1(a).

The sender then randomly picks some vertices (at least two), which will be used as the secret key k. The sender then runs the HIDDEN-TREE algorithm on the maze M and the key k to produce a hidden tree H, which is shown in Figure 1(b). The figure illustrates the path (shown as red lines) between the vertices in the key (shown as green dots).

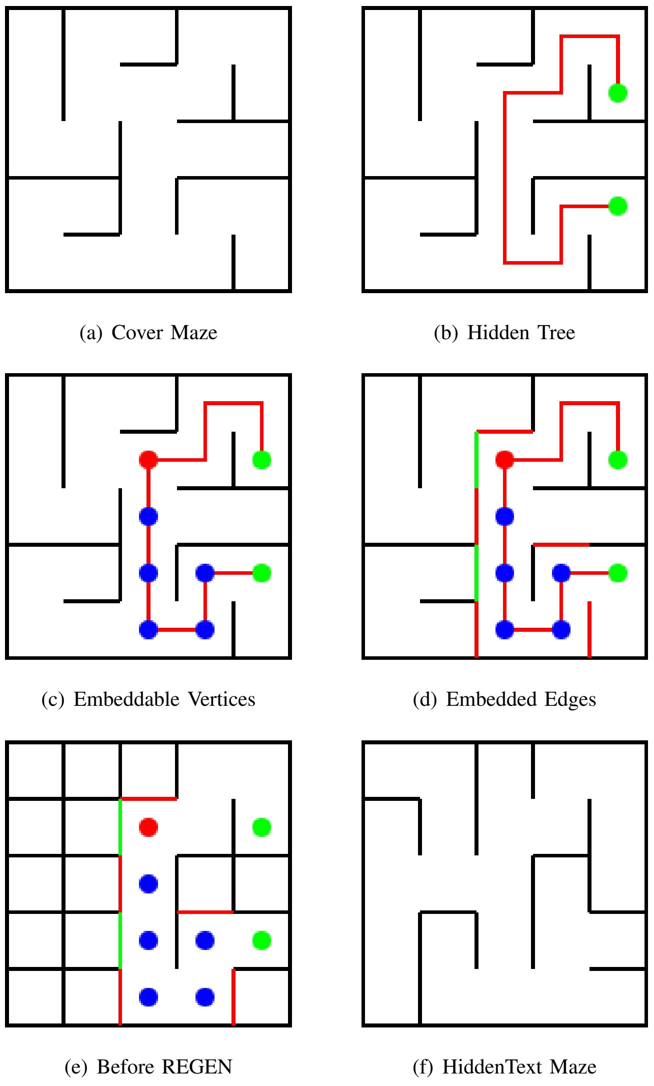
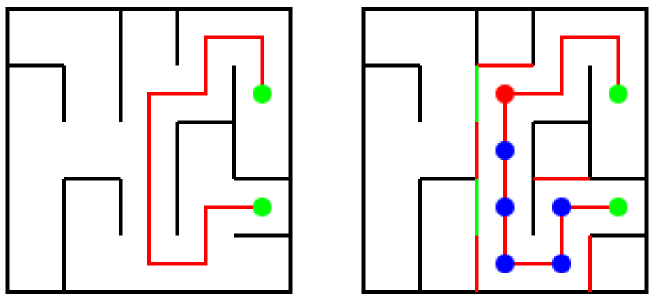

(a) Cover Maze (b) Hidden Tree
(c) Embeddable Vertices (d) Embedded Edges
(e) Before REGEN (f) HiddenText Maze

Fig. 1. Steganography Sender Illustration

After that, the sender runs the FIND-EMBEDDABLE-EDGES algorithm. This will mark embeddable vertices, as shown in Figure 1(c). One-embeddable vertices are shown as blue dots, while two-embeddable vertices are shown as red dots. The vertices have been processed in the order they appear on the solution path (the start point is the lower green dot). This algorithm returns the list of embeddable edges EE which is used later.

Next, the sender runs the EMBED algorithm. This will, based on the secret data d (one bit per edge), assign embeddable edges to the set of edges to keep (KE) or discard (DE). In Figure 1(d), edges in KE are shown in red, while edges in DE are shown in green.

Lastly, the sender will create a new graph $G(V, E)$ which has all the vertices on a $w \times h$ lattice; and a partly defined maze $M_p(V, KE \cup H.\text{edges})$ which has the edges in KE and the edges along the solution path. Figure 1(e) shows M_p. The sender will then run the REGEN algorithm to return a full maze M' such that all edges in M_p are in M' and no edge in DE is in M'. Figure 1(f) shows the hiddentext maze M'



(a) Hidden Tree (b) Embedded Edges

Fig. 2. Steganography Receiver Illustration

which is transmitted to the receiver.

The receiver will take M' and run the HIDDEN-TREE algorithm to output the hidden tree H' corresponding to the key k. The result is shown in Figure 2(a). Note that the hidden tree H' in this figure is the same as the hidden tree H in Figure 1(b). The REGEN algorithm guarantees this.

The receiver will then run the FIND-EMBEDDABLE-EDGES algorithm to mark which edges are embeddable, as shown in Figure 2(b). The receiver will then run EXTRACT, which will process each embeddable edge in order, see whether it is actually in KE (and thus in M') or in DE (and thus not in M'). In this way, it will be able to read the secret data d.

C. Algorithms

We now describe the algorithms used in EEDGE. For brevity, assume that the lattice parameters w, h are globals that are available to every algorithm.

Algorithm 1 COVER

```
function COVER
    Let $V$ be the set of all vertices on a $w \times h$ lattice.
    Let $E$ be all edges between neighbouring vertices in $V$.
    return RANDOM-MST(V,E)
end function
```

The COVER algorithm is trivial. We simply use any randomized MST (Minimal Spanning Tree) algorithm to output a maze on the $w \times h$ lattice. This will be used as the cover message within which we will hide our data.

The HIDDEN-TREE algorithm assumes the existence of a SOLVE-MAZE algorithm, which returns a list of vertices along the solution path and the list of edges along the solution path. In practice, this can be implemented using BFS or DFS.

Many Randomized MST algorithms can be easily adapted to become a suitable implementation of REGEN. We provide an implementation based on Kruskal's algorithm, with a disjoint-set data structure supporting the FIND-SET and UNION operations.

D. Correctness

We need to prove that our algorithms return correct results; which means that the secret data d should not be corrupted.

Algorithm 2 HIDDEN-TREE

```
function HIDDEN-TREE(k,M)
    $V \leftarrow k$
    $E \leftarrow \{\}$
    for $i = 2 \rightarrow k.length$ do
        $(v, e) \leftarrow$ SOLVE-MAZE$(M, k[1], k[i])$
        $V \leftarrow V \cup \{v\}$
        $E \leftarrow E \cup \{e\}$
    end for
    return $(V, E)$
end function
```

Algorithm 3 FIND-EMBEDDABLE-EDGES

```
function FIND-EMBEDDABLE-EDGES(M,k)
    $EV \leftarrow k$
    $EE \leftarrow \{\}$
    $(V, E) \leftarrow$ HIDDEN-TREE$(k, M)$
    for $v \in V$, $v \notin k$ do
        if $v$ is a one-embeddable vertex then
            $EV \leftarrow EV \cup \{v\}$
            $EE \leftarrow EE \cup \{(v, n)\}$
        end if
        if $v$ is a two-embeddable vertex then
            $EV \leftarrow EV \cup \{v\}$
            $EE \leftarrow EE \cup \{(v, n_1), (v, n_2)\}$
        end if
    end for
    return $(V, E, EE)$
end function
```

Formally:

$$\forall c, k, d \quad \text{EXTRACT}(\text{EMBED}(c, k, d), k) = d$$

Proving correctness of COVER, and HIDDEN-TREE is trivial, as they simply invoke standard algorithms. FIND-EMBEDDABLE-EDGES can similarly be shown to be correct as it just applies the embeddable vertex criteria and adds vertices to the appropriate set.

Algorithm 4 REGEN-KRUSKAL

```
function REGEN-KRUSKAL(V,E,KE,DE)
    $FE \leftarrow KE$
    $E \leftarrow$ SHUFFLE$(E - DE)$
    for $v \in V$ do
        MAKE-SET$(v)$
    end for
    for $(v_1, v_2) \in KE$ do
        UNION$(v_1, v_2)$
    end for
    for $(v_1, v_2) \in E$ do
        if FIND-SET$(v_1) \neq$ FIND-SET$(v_2)$ then
            $FE = FE \cup \{(v_1, v_2)\}$
            UNION$(v_1, v_2)$
        end if
    end for
    return $(V, FE)$
end function
```

Algorithm 5 EMBED

```
function EMBED(M,k,d)
    $(HV, HE, EE) \leftarrow$ FIND-EMBEDDABLE-EDGES$(M, k)$
    $DE \leftarrow \{\}$
    $KE \leftarrow HE$
    for $e \in EE$ do
        if READ-NEXT-BIT$(d)$ == $1$ then
            $KE \leftarrow KE \cup \{e\}$
        else
            $DE \leftarrow DE \cup \{e\}$
        end if
    end for
    Let $V$ be the set of all vertices on a $w \times h$ lattice.
    Let $E$ be all edges between neighbouring vertices in $V$.
    return REGEN$(V, E, KE, DE)$
end function
```

Algorithm 6 EXTRACT

```
function EXTRACT(M',k)
    $EE \leftarrow$ FIND-EMBEDDABLE-EDGES$(M', k)$
    $d \leftarrow []$
    for $e \in EE$ do
        if $e \in M'$.edges then
            $d$.append$(1)$
        else
            $d$.append$(0)$
        end if
    end for
    return $d$
end function
```

The REGEN-KRUSKAL algorithm should output a Random MST (V, E) such that no edge in DE is in E and all edges in KE are in E. By an analysis of the algorithm, we can see that we are simply running Kruskal's algorithm on a graph $G(V, E - DE)$ where the vertices of edges in KE have already been added to the same set. Thus, if this graph has an MST, REGEN-KRUSKAL will find it. At the end, we add in the edges in KE to this result, which will not add any cycles because Kruskal's algorithm assumed those vertices to be connected (as we had added them to the same set).

The one concern in the above proof is the **if** in the statement "if the graph has an MST". However, simple intuition shows that it is highly unlikely for such a graph (e.g. Figure 1(e)) not to have an MST, since there is likely to be a connection from the hidden tree H to the rest of the graph. We can back this notion up with our experimental results; where we have not encountered a case where an MST is not present in over 10^6 experimental trials. Thus REGEN succeeds with high probability.

Lastly, we have to show that the REGEN algorithm does not modify the hidden tree H in any way so that it can be read later. Assuming that all edges in the hidden tree are in KE (which EMBED does), these edges will be preserved as shown before. As the REGEN algorithm generates an MST with no cycles (as shown before), there will be no other path between these vertices and the hidden path will be preserved.

The correctness of EMBED and EXTRACT is similarly trivial to prove. EMBED simply adds edges to the set of edges to keep (KE) or discard (DE) based on the secret data, and then calls REGEN. EXTRACT simply calls FIND-EMBEDDABLE-EDGES and then traverses the graph and sees whether edges are present (and thus were in KE, representing a 1) or absent (and thus in DE, representing a 0).

E. Security

If the algorithms used by COVER and REGEN are the same (as in Figure 1, where the Kruskal algorithm is used) there will be no stylistic difference between a cover maze M and a hiddentext maze H. Both mazes are perfect mazes (since they are MSTs) and H could have equally likely been generated by another call of COVER. To see this, note that a randomized implementation of Kruskal's algorithm shuffles the list of edges and then processes them in order; and can result in the same maze that would be generated by a call to REGEN-KRUSKAL with a certain set of edges to keep and to discard. This is since the edges to discard are removed by moving them to the front of the list of edges to be processed, and edges to be kept are done so by moving them to the end of the list.

To back up the assertion of visual indistinguishability, we can also attempt to train a graph classifier to distinguish between cover mazes and hiddentext mazes. In Section V we present results showing that classifiers are unable to distinguish between cover and hiddentext mazes. We also present results of a user study showing that users were unable to visually distinguish between cover and hiddentext mazes. Since the hidden texts are indistinguishable from the cover messages, an attacker can not know whether a certain maze contains secret data or not.

With some suspicion about the presence of a hidden message and without knowledge of the key, the attacker is thus reduced to bruteforce guessing. For this, the attacker has to guess the size of the key, the vertices in the key, and the order of the vertices in the key. For a $w \times h$ lattice with a maximum key size of k, the number of combinations is:

$$N = \sum_{2 \le i \le k} \binom{w \cdot h}{i} \cdot i!$$

For a 64×64 lattice with $k = 4$, this equates to about $2.8 \cdot 10^{14}$ possible combinations for the attacker to check, which means that the probability of actually guessing the key correctly is very low.

IV. Covert Channel

It is trivial to apply EEDGE in the context of rogue-likes to achieve a real world covert channel. Note that we limit the scope of our covert channel to games in which the levels are mazes (and thus do not have cycles).

For the purposes of this discussion, an online rogue-like involves players $P_1 \ldots P_n$ exploring randomly-generated levels $M_1 \ldots M_n$ sent by a server S. There may be many players in the same level at the same time, or each player may be in their own level. The key point is that the levels are randomly generated by the server and sent to the players.

TABLE I. BITS EMBEDDED IN 64×64 MAZE WITH KEY LENGTH 5.

Algorithm	EEDGE	ECELL
RecursiveBacktracker	780	260
Kruskal	364	144
Prim	264	109
RecursiveDivision	449	170
GrowingTree	279	110
HuntAndKill	433	154
SideWinder	338	123

Assume Alice and Bob have shared a PRNG seed s. If Alice wants to use a covert channel to send messages to Bob, she can setup a public game server. Bob, along with Eve, Mallory, and many other players, can play the game on this server. When Alice wants to send a message to Bob, she does the following:

1) Run the COVER algorithm to get a random maze C.
2) Use the PRNG to generate a key length l.
3) Randomly pick l points to use as a key k.
4) Use EMBED to generate a maze M using k and C.
5) Broadcast M as a new level.

Since EEDGE is secure, other players will just see M as a perfectly normal level, so they will not have any idea that anything is amiss. However, Bob can use the PRNG to regenerate the key, and run EXTRACT on the key and M to recover the secret message that Alice intended to send.

V. EXPERIMENTAL EVALUATION

We implemented EEDGE, as well as the stegosystem by Lee et al in [6], hence referred to as ECELL. All experiments were run on a machine with a 2.27 GHz Quad Core i5 processor and 8GB RAM.

A. Embedding Capacity

We implemented COVER using various algorithms from the list available in [9]. Some algorithms produce longer solution paths, which result in a higher capacity for embedded data. Most notably, the RecursiveBacktracker algorithm generates mazes width long paths that don't have many branches, so they tend to have a higher capacity. We provided four implementations of REGEN, using the Kruskal, Prim, DFS, and HuntAndKill algorithms.

Table I shows a comparison of EEDGE against ECELL for various implementations of COVER. The experiment was run with 500 random mazes of size 64×64 and a key length of 5 points (which means 4 paths). The process in Section IV was followed, and the number of bits embedded in the maze was noted. The average bits embedded per maze was calculated. As can be seen from the figure, EEDGE is significantly better in terms of embedding capacity.

Figure 3 shows how the number of embedded bits in EEDGE grows as a function of both the maze size and the key length, again taken as an average over 500 random mazes. We can see that maze size is the major factor in determining the number of bits that can be embedded. The intuition behind this is that in a larger maze, the key points are further apart,

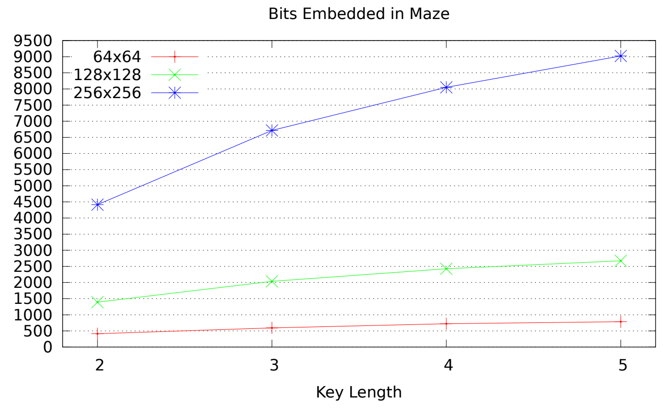


Fig. 3. Bits Embedded as a function of key length and maze size.

thus the hidden tree contains more edges; making it more likely for it to have more embeddable edges.

The ratio of cover data to actual embedded data is reasonable in EEDGE. It takes two bits to encode one cell in a maze, since we have to store the edges to the right and to the bottom. So we need 8192 bits to represent a 64×64 maze; in which we can store 780 bits (see Table I). Thus the ratio of actual data to cover data is about 9.5%.

Our simple, unoptimized implementation can embed (and then extract) approximately 21k bits per second, when repeatedly generating mazes as per the procedure given in Section IV. Most of the time is spent in the maze generation, regeneration, and solving algorithms. The algorithms are also fairly concise and simple to implement, which is a practical bonus for people looking to use EEDGE. Our implementation contains 7 generation algorithms, 4 regeneration algorithms, 3 solution algorithms, and two stegosystems, all in under a thousand lines of code.

B. Steganalysis - Classifiers

Since mazes are just special classes of graphs, if there is a systematic difference between hiddentext mazes and cover mazes, a graph classifier should be able to accurately classify a given maze into the hiddentext or cover maze categories. If graph classifiers are unable to do so, it provides a compelling argument for the security of EEDGE.

We decided to try out three graph classifiers to see whether they could detect hiddentext mazes.

For SubdueCL [2] and GAIA [4], we generated 100 random mazes of size 20×20 (larger mazes were not used due to computational limitations). 50 of these mazes were cover mazes, and 50 were hiddentext mazes (albeit generated from another set of cover mazes). SubdueCL was unable to produce a meaningful set of substructures for its classifier. It had a validation error of 0.5, with 50 false positives and 50 true positives, indicating it classified every maze as hiddentext, which is not useful at all. GAIA returned no classification, stating "graph has no discriminative feature" for each of the input graphs, also indicating the lack of meaningful distinguishing data for classification.

TABLE II. gBoost Classifier results

Accuracy	0.45
ROC AUC	0.5336
ROC EER	0.48
True Positives	19
True Negatives	26
False Positives	24
False Negatives	31

For gBoost [5], we generated 100 random mazes of size 64×64 (50 hiddentext, 50 cover) for both the training and test sets of data. The output results, containing accuracy and ROC curve statistics, are presented in table II. This classifier also performs poorly, worse than even random coin flips. The high rate of false positives and false negatives show that the results are not very useful.

C. Steganalysis - User Study

In order to support our claim that EEDGE generates hiddentext mazes that are visually indistinguishable from cover mazes, we decided to perform a user study. We designed a survey containing 10 sets of six mazes each, one of which was a hiddentext maze while others were cover mazes. Participants were told that one maze in each set was generated by a secret agent and looked slightly different from the rest, introducing them to the idea of a hiddentext maze. Their task was to try and identify the hiddentext maze in each set. We gathered 31 responses, from university students who had a background in Computer Science. We then totaled the number of correct answers for each participant. The mean was 1.54, with a standard deviation of 1.12. The highest score was 4. The mean result is approximately equal to what would be expected given random guessing (1.67); indicating that users were not able to visually distinguish between cover and hiddentext mazes.

VI. Conclusion and Future Work

We have proposed and implemented EEDGE, an improved system for steganography in mazes; and shown how it can be applied to create a covert channel in online rogue-likes. EEDGE is simple, efficient, and secure; and the resulting covert channel is both error-free and has a high bit rate. EEDGE is secure against steganalysis attacks using graph classifiers, and was shown to be indistinguishable from random mazes in a user study.

The limitation of EEDGE is that it works on mazes, and thus it can only be applied to a setting where all the game levels are mazes. In the future, we would like to investigate whether it would be possible to create a steganographic scheme that works with general levels (modeled as arbitrary graphs); which could then be applied to create covert channels in a much larger class of online games.

Acknowledgment

The authors would like to thank the participants of the user study for their feedback.

References

[1] A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt. Digital image steganography: Survey and analysis of current methods. *Signal Processing*, 90(3):727–752, 2010.

[2] J. A. Gonzalez, L. B. Holder, and D. J. Cook. Graph based concept learning. *AAAI/IAAI*, 1072, 2000.

[3] J. C. Hernandez-Castro, I. Blasco-Lopez, J. M. Estevez-Tapiador, and A. Ribagorda-Garnacho. Steganography in games: A general methodology and its application to the game of go. *computers & security*, 25(1):64–71, 2006.

[4] N. Jin, C. Young, and W. Wang. Gaia: graph classification using evolutionary computation. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 879–890. ACM, 2010.

[5] T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Advances in neural information processing systems*, pages 729–736, 2004.

[6] H. Lee, C. Lee, and L. Chen. A perfect maze based steganographic method. *Journal of Systems and Software*, 83(12):2528–2535, 2010.

[7] M. L. Mauldin, G. Jacobson, A. W. Appel, and L. G. Hamey. Rog-o-matic: a belligerent expert system. 1983.

[8] S. Murdoch and P. Zieliński. Covert channels for collusion in online computer games. In *Information Hiding*, pages 419–429. Springer, 2005.

[9] W. Pullen. Think labyrinth: Maze algorithms, Jan. 2011.

[10] G. J. Simmons. The prisoners problem and the subliminal channel. In *Advances in Cryptology. Proc. of Crypto*, volume 83, pages 51–67, 1984.

[11] S. Zander, G. Armitage, and P. Branch. A survey of covert channels and countermeasures in computer network protocols. *Communications Surveys & Tutorials, IEEE*, 9(3):44–57, 2007.

[12] S. Zander, G. Armitage, and P. Branch. Covert channels in multiplayer first person shooter online games. In *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, pages 215–222. IEEE, 2008.

[13] J. Zöllner, H. Federrath, H. Klimant, A. Pfitzmann, R. Piotraschke, A. Westfeld, G. Wicke, and G. Wolf. Modeling the security of steganographic systems. In *Information Hiding*, pages 344–354. Springer, 1998.