Winter 2015

Synthetic steganography: Methods for generating and detecting covert channels in generated media

Philip Carson Ritchey
Purdue University

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Philip Carson Ritchey

Entitled
Synthetic Steganography: Methods for Generating and Detecting Covert Channels in Generated Media

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Vernon Rego

Aditya Mathur

Samuel Wagstaff

Wojciech Szpankowski

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Vernon Rego

Approved by: William Gorman 2/11/2015

Head of the Departmental Graduate Program Date

SYNTHETIC STEGANOGRAPHY: METHODS FOR GENERATING AND DETECTING COVERT CHANNELS IN GENERATED MEDIA

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Philip Carson Ritchey

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2015

Purdue University

West Lafayette, Indiana

It's not about winning, it's about sending a message.

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Vernon Rego, for his support and advice throughout my studies at Purdue University. I would also like to thank my committee, Professors Wojciech Szpankowski, Samuel Wagstaff, and Aditya Mathur for their guidance in this work.

I would like to extend a special thank you to Professors Sunil Prabhakar, Susanne Hambrusch, and Greg Frederickson for providing me with the opportunity to teach and the motivation to finish.

Finally, I would like to thank my wife, Niki Ritchey, my parents, Pete Ritchey and Maggie Carson, and my brother, Paul Ritchey, for their continued support. I could not have done this without them.

Financial support for this work was provided by the National Science Foundation through the Center for the Science of Information, a Science and Technology Center at Purdue University (Grant No. CCF-0939370), and Grant No. CNS-0716398.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Ritchey, Philip Carson Ph.D., Purdue University, May 2015. Synthetic Steganography: Methods for Generating and Detecting Covert Channels in Generated Media. Major Professor: Vernon J. Rego.

Issues of privacy in communication are becoming increasingly important. For many people and businesses, the use of strong cryptographic protocols is sufficient to protect their communications. However, the overt use of strong cryptography may be prohibited or individual entities may be prohibited from communicating directly. In these cases, a secure alternative to the overt use of strong cryptography is required. One promising alternative is to hide the use of cryptography by transforming ciphertext into innocuous-seeming messages to be transmitted in the clear.

In this dissertation, we consider the problem of synthetic steganography: generating and detecting covert channels in generated media. We start by demonstrating how to generate synthetic time series data that not only mimic an authentic source of the data, but also hide data at any of several different locations in the reversible generation process. We then design a steganographic context-sensitive tiling system capable of hiding secret data in a variety of procedurally-generated multimedia objects. Next, we show how to securely hide data in the structure of a Huffman tree without affecting the length of the codes. Next, we present a method for hiding data in Sudoku puzzles, both in the solved board and the clue configuration. Finally, we present a general framework for exploiting steganographic capacity in structured interactions like online multiplayer games, network protocols, auctions, and negotiations. Recognizing that structured interactions represent a vast field of novel media for steganography, we also design and implement an open-source extensible software testbed for analyz-

ing steganographic interactions and use it to measure the steganographic capacity of several classic games.

We analyze the steganographic capacity and security of each method that we present and show that existing steganalysis techniques cannot accurately detect the usage of the covert channels. We develop targeted steganalysis techniques which improve detection accuracy and then use the insights gained from those methods to improve the security of the steganographic systems. We find that secure synthetic steganography, and accurate steganalysis thereof, depends on having access to an accurate model of the cover media.

1 INTRODUCTION

Privacy is a major concern in today's world. Cryptography has been, and remains, the standard solution for obtaining most forms of privacy. However, the use of strong cryptography is sometimes prohibited. Furthermore, suspicious and paranoid observers may take the usage of cryptography as evidence that the sender has a secret to hide. In the interest of preserving and promoting freedom of information, freedom of thought, and freedom of expression, people must be able to use secure communication channels safely, especially when such usage is under attack. One solution to the problem of communicating securely without the overt use of cryptography is to hide the ciphertext using techniques from the field of information hiding. Information hiding protocols, such as steganography, can hide the existence of the secure communication channel to protect the endpoints as well as the information in the channel, allowing the use of cryptography without revealing that it is being used.

1.1 Information Hiding

Information hiding is a field of information security which deals with privacy concerns. It can be thought of as part of the general and broad field of cryptography , the practice and study of techniques for secure communication in the presence of adversaries [1]. However, unlike the core techniques of cryptography which are concerned with keeping secret *what* secret information Alice communicates to Bob, information hiding is concerned with keeping secret *if* and *when* Alice and Bob are communicating secret information. While encryption obfuscates the content of a message in order to make it unreadable, information hiding techniques disguise the message in order to make it undetectable or untraceable. Two sub-fields of information hiding, steganography and anonymity, respectively deal with those two goals.

The field of anonymity is comprised of methods for keeping secret the source and destination of information. Anonymity techniques rely heavily on strong cryptography. If an adversary can break the encryption which protects the routing information in an onion routing network, for example, then they can trace the communication much more easily and violate the privacy of the users. Fortunately, strong cryptography is called "strong" for a reason: it is extremely difficult to break. But sometimes strong cryptography is prohibited, rendering ineffective any anonymizing network which relies solely on strong cryptography. This is where the field of steganography comes to the front.

Steganography is the art and science of hiding secrets in such a way that the existence of the secret is undetectable to a third party adversary or observer. There is evidence that steganography has been in use for over 2000 years [2] with two examples found in the Histories of Herodotus [3]. The first recorded usage of the term occurs in Trithemius' Steganographia, a book written circa 1499 CE which is ostensibly about magic but which is, in fact, a treatise on cryptography and steganography [4]. Before computers and the digital age, steganography was limited to physical techniques, e.g. Histiaeus tattooing a slave's shaved head and waiting until the hair grows back to send the slave to deliver the message, or Mary Queen of Scots' letters smuggled in and out of her confinement in barrels of beer [5]. However, the complexity and enormous resources offered by computers and high speed communication networks have proved to be very fertile ground for the science of steganography to grow into the vast and varied field that it is today.

The field of steganography is comprised of methods which disguise information as innocuous channel usages. Channel usage includes the objects which are sent over the channel as well as how the channel is used, such as timing. The field is relatively young and very active. Cover-objects can be any digital object and are not limited to traditional media such as images, audio and text. Cover-objects can also be compound objects, encompassing more than single object, or complex objects spread out over multiple dimensions in time or space. Taking this broad view of cover-media

allows us to include within the field of steganography a discipline which often tries to distance itself from the field to which it so clearly belongs: covert channels.

Covert channels were originally defined by Lampson as channels that are "not intended for information transfer at all" [6]. However, even channels intended for information transfer can be viewed as being covert channels themselves, e.g. covert channels in the TCP/IP protocols [7]. We may now define a covert channel as any channel between two processes that are not supposed to share information. A covert channel is created in a legitimate channel when a process uses the channel in such a way as to signal secret information, creating a *timing channel* or sends an object containing secret information through the channel, creating a *storage channel*. If process A can modify a resource that process B can access or measure (e.g. packet arrival times, file locks, system temperature, etc.), then A can send information to B through a covert channel using the shared resource [8]. Some covert channels violate the security policy of a system and many techniques, such as capacity reduction and auditing, have been developed for identifying and mitigating covert channels [9]. Channels such as those between processes that are not prohibited from communicating (because, for example, they have the same security level and no covert channels exist between either of them and any process with a different security level), but which are not intended for communication are also considered covert channels. Typically, only those covert channels which violate the security policy of the system are of concern, the rest can be safely ignored.

Until very recently, research on covert channels only focused on methods for identifying them and limiting or eliminating their capacity once identified. It is well known, since Lampson [6], that completely eliminating all covert channels in a real system is impossible. If we consider the Internet and look for covert channels there, we find that there are so many potential covert channels that just counting them all would be almost impossible, not to mention deploying countermeasures to eliminate or constrain them. Thus, another popular approach to handling covert channels, after

they have been identified and had their capacity reduced as much as possible, is to audit the covert channel to know when and whether it is being used.

Covert channels are not, themselves, typically thought of as being secure. They undermine security and we try to eradicate and constrain them as best we can to limit their damage. But, what would a secure covert channel look like, if we needed to design a covert channel to safely and securely transmit secrets? Covert channel security is not fundamentally different from steganographic security. Steganographic security is based on the computational or statistical indistinguishability of stego-objects from cover-objects [10–14]. The harder it is to tell stego-objects from cover-objects, the more secure is the stego-system. Similarly, covert channel security is based on the statistical or computational indistinguishability of covert channel usage from normal channel activity. The harder is it to tell when the channel is being used, the more secure is the covert channel.

A secure covert channel is one whose capacity cannot be eliminated and whose covert usages are indistinguishable from it's overt, authentic usages. That is, auditing of the channel will not give the observer any clues as to when or whether the channel is being used to pass secret information instead of being used for a legitimate purpose. In this sense, covert channel security is the same notion as steganographic security: a passive observer must decide whether or not an object is being used for covert communication. The object could be a singular multimedia object which has been modified or generated in such a way as to encode the hidden data, or it could be a set of objects whose relative ordering or statistical and computational properties encode the hidden data. Therefore, we can consider covert channels to be a special case of steganography: covert channels are steganographic objects which encode the secret information as signals which one process can create in a shared resource that another process can measure. These stego-objects are modifications of normal behavior for the processes which modify and measure the shared resource.

One important difference between steganography and covert channels is that covert channels require some amount of covert communication to take place before

the usage is detectable, but steganographic objects can be held and analyzed before being forwarded. This means that an adversary can use detection to eliminate steganographic channel capacity, but cannot always do so for covert channels.

1.2 Secure Covert Channels in Structured Interactions

It is well-known that many types of digital media can be used to hide information. One might be willing to even go so far as to say that it is possible to use any digital medium to hide information. A simple argument for this claim is given by Hopper, VonAhn, and Langford [13, 14], who show that it is theoretically possible to encode data using any digital medium by mapping each channel object to a bit (or string of bits) and drawing objects at random from the channel distribution. In their system, channel objects are not modified to encode data, but rather the channel usage itself is modified.

Structured interactions are protocols between one or more agents which follow a set of rules and therefore allow for only finite variation at each step of the interaction. A familiar instance of a structured interaction is a game, such as Tic-tac-toe or Chess. At each step, the set of possible next steps is finite, relatively small, and easily enumerable. Structured interactions provide an attractive domain in which to operate a covert channel. The limited space of legal next actions dramatically decreases the complexity of picking a reasonable one at random and the context of the interaction provides a plausible explanation for why that particular action was chosen.

Following from the definition of steganographic security given above, a secure covert channel for a structured interaction is one in which the steganographic interactions are statistically and computationally indistinguishable from the authentic, legitimate interactions. This means that whatever is being modified or generated to encode the secret data must conform to the properties of authentic interactions. As an example, in order to be secure, a stego-system which hides data in gameplay must

generate gameplay which very closely resembles the gameplay of authentic players. Playing randomly or making moves uncharacteristic of a human player are signs that the play is not authentic, and therefore may be transmitting hidden data. However, the ability to detect the signs of synthetic gameplay requires a model of authentic gameplaying behavior which is more accurate than the model used to generate the synthetic gameplay. Other examples of structured interactions which have been used to host covert channels include network protocols and financial transactions. Again, the security of the covert channel depends on the relative quality of the generative model used for steganography compared to the descriptive model used for steganalysis. Structured interactions represent a vast field of novel media for steganography. Methods for generating and detecting covert channels in structured interactions, especially in games, is an active and exciting field of study.

1.3 Contribution

We consider the problem of generating and detecting covert channels in generated media. We develop a relatively sophisticated and practical secret-key stego-system for generating time series such as those seen in financial markets. Our stego-system provides layers of complexity which enable information-hiding in an arbitrary series of values with control over the information density of the embedding and well as the location and size of the perturbations required to embed the secret data. We develop and analyze a novel image-steganographic scheme based on context-sensitive tilings which generates synthetic stego-objects with high levels of robustness to tampering and security against steganalytic attacks. The images are generated so that the structural content of the image is in itself the payload. The images generated by the stego-system can be said to have both structure and semantics. We develop and analyze a novel information hiding scheme which uses the structure of Huffman trees to encode secret data. This approach is in contrast to previous methods which hid data in the content of the tree. Under the right circumstances, the stego-tree is

indistinguishable from a clean tree for the same content. We develop and analyze a stego-system which hides data in generated Sudoku boards and puzzles. Our system is a significant improvement over previous work on hiding data in Sudoku puzzles as it is both more secure and able to hide five times as many bits in each puzzle. We investigate covert channels in games by implementing a general framework for hiding data in games and applying it to the game of Tic-tac-toe. We show that the capacity of the covert channel in Tic-tac-toe and other combinatorial games cannot be eliminated by enforcing any particular strategy, including that of optimal play. We follow this up by developing three steganalysis techniques for detecting covert channels in games using data collected from human players. We show that a warden with access to human gameplay data can distinguish between gameplay generated by humans and gamplay generated by rules-based software agents with a high level of accuracy. We also show that the warden's accuracy increases with the number of games observed before a decision is made, revealing a trade-off between time to detection and number of bits allowed to be transmitted. Finally, we designed and developed a software testbed for analyzing steganographic interactions. The testbed provides a framework for implementing and analyzing steganographic systems that exploit structured interactions for covert communication. We used the testbed to provide the first measurements of the capacity of covert channels in several games, including Tic-tac-toe, Chess, and Go. Our work has theoretic, empirical, and heuristic components with serious implications for the world of practical secure communication.

1.4 Organization

The remainder of this dissertation is organized as follows. The literature review in Section 2 describes the previous work related to our study of covert channels in generated media, with a particular emphasis on games and human gameplaying behavior. Section 3 presents some our work on synthetic steganography in timeseries data. Section 4 presents and analyzes a context-sensitive tiling system for generating

steganographic images. Section 5 presents and analyzes a method for hiding data in the structure of Huffman trees which prioritizes security over capacity. In Section 6 we present and analyze StegoDoku, a stego-system which hides data in Sudoku boards and puzzles. Section 7 presents the Tic-Tac-Stego system, a general framework for exploiting covert channels in games, and applies it to several games. In Section 8, three steganalysis techniques for detecting active covert channels in games are developed and used to analyze the security of the Tic-Tac-Stego system. The security analysis utilizes gameplay data collected from humans. Section 9 presents the design and implementation of a software testbed for analyzing steganographic interactions. Finally, we conclude in Section 10 with a summary of our work and suggestions for future work.

2 LITERATURE REVIEW

There are many ways to do steganography. A standard example of steganography in the digital age exploits image files [15–17]. In a bitmap image, the least significant bits are below the human visual system's sensory threshold. As such, changes to the bits in those positions are undetectable to humans. Therefore, a steganographer can replace the low order bits of an image with the bits of a secret message to embed that message into the image. This is known as least significant bit (LSB) steganography. A similar approach can be taken to hiding data in audio files as well [18, 19]. While a human cannot accurately detect such changes, computational analysis can be highly accurate at detecting this kind of modification of an image. More sophisticated methods have pushed the embedding locale from the temporal and spatial domains, where LSB steganography lives, to the frequency domain using DCT and wavelet transforms. Modifications made in this way are still undetectable by humans, but are also more resistant to computational and statistical analysis. The modifications are still made the same way they were using LSB, only now the changes are in the frequency domain and so the embedded bits get spread throughout the image or audio file when it is transformed back into the time or space domain.

Methods which modify an existing object account for the vast majority of steganographic methods currently being developed and studied. Johnson and Katzenbeisser [15] have grouped steganographic methods into five categories according to the manner in which they modify the cover-object to hide the secret information: substitution systems, transform domain techniques, spread spectrum techniques, statistical methods, and distortion techniques. However, one alternative to modification-based methods, which are susceptible to simple comparison against other versions of the same object, is to generate a cover-object so that the modified version is the only version ever released, or generating objects such that the construction of the object

itself encodes the information. Johnson and Katzenbeisser identify a sixth category for such methods. These so-called cover generation methods avoid the issue of comparison between versions, but the problem of automatically generating cover-objects that can pass convincingly as authentic cover-objects now looms large, especially in the case of media such as images, audio, and text.

2.1 Cover Generation Methods

The category of cover generation methods is for those techniques which hide information in generated cover-objects, often in the structure of the object. In contrast to other types of steganographic techniques, cover generation methods do not augment an already existing object in order to hide bits of information. The generation of novel objects is not restricted to the field of information hiding, however. Techniques in the field of computer graphics for procedurally generated media are also concerned with generating novel content which mimics some target object, such as trees and cities. When applied to information hiding, the goal is similar: to generate objects which closely resemble the target object. However, cover generation methods for steganography have a requirement that procedurally generated media do not generally have, which is *reversibility*. The generated objects must be able to be deconstructed in order to recover the hidden bits of information. There is a compromise which can be made to mitigate the difficulty of satisfying this requirement in practice: use a target for which the generation process is fully observable and therefore able to be recorded. In this way, even if the end product is not reversible when taken by itself, the fact that there exists a transcript which recorded the steps taken by the process means that deconstruction is either unnecessary, or made possible when given the transcript. We now consider a number of existing steganographic methods which belong to the category of cover generation methods.

In their work on hiding information in pseudo-random images [20], Blundo and Galdi present a method for hiding information in image mosaics. First, a database of

small images is maintained, and these are classified, according to their visual properties, for use in the image mosaic formation process. Next, the images are then arbitrarily separated into classes representing the binary digits one and zero. In this way it is possible to produce an image mosaic which approximates some input image and which hides secret information in the choices of sub-images that form the mosaic.

In his work on model-based steganography, Sallee approaches the generation of stego-objects from a compression point of view [21]. With access to a perfect model of some cover media, a perfect compressor can be constructed. Instances of the cover media can be given as input to the perfect compressor and it would return "perfectly compressed, truly random bit sequences". Thus, since access to a perfect compressor implies access to the corresponding decompressor, feeding any random bit sequence to the decompressor would produce a sensible instance of the cover media. This property can be used to convert a secret payload into an instance of the cover media by first processing the payload to turn it into some random bit sequence and then feeding that bit sequence to the decompressor. Alice can send the resulting stego-object to Bob without raising much, if any, suspicion. To recover the message, Bob compresses the object using the perfect compressor and reverses Alice's processing of the result. Perfect models are very hard to come by, even for simple objects. However, this method provides perfect security against any adversary with a model which is less perfect than the one used in the compression system.

Radhakrishnan, Kharrazi and Memon use a technique called *data masking* to process the entire secret message to make it appear statistically similar to some type of multimedia object [22]. According to their results, when linear and SVM classifiers were trained to detect stego in data masked images, it was found that the detectors achieved detection rates above 97%, with a false alarm rate less than 13%. Despite the weak security performance of the steganographic scheme based on data masking, the approach still seems a step in the right direction: from augmentation of existing images towards generation of novel images.

In [23] and [24], Sung, Tadiparthi, Mukkamala and Sueyoshi present techniques for hiding secret data in animations. A context free grammar is used to generate an animation wherein certain sequences of frames signify a 1 and other sequences of frames signify 0. The secret data is encoded as an animation by selecting a sequence of frames which corresponds to the next bit(s) of data to hide. It is shown that the system is secure against a passive warden.

In [25], we present a technique for hiding information in stochastic settings via data-synthesizing schemes based on transform-expand-sample (TES) processes. The technique is applicable in any setting where a process or application generates data that exhibits random but structured behavior and data trajectories have viable alternatives that are very difficult or impossible to verify. In such cases, a synthesizing procedure generates novel data that replaces application or process data. When information can be hidden in such data during the generation process, typical message-neutralizing attacks will fail. If synthetic process data cannot be accurately distinguished from authentic process data, then secure steganographic transmissions can be conducted.

2.2 Covert Channels in Games

One domain where the difficulty of generating convincing cover-objects does not appear to loom so large is in games. It has been known, at least since World War II, that games can harbor covert channels. During the second world war, the Office of Censorship of the United States banned international chess games by mail because the games could be used to encode secret information [2, p. 515]. With the advent of the home computer and the Internet, games can now be played by users sitting on opposite sides of the planet with the same celerity as if the players were sitting across the table from one another. Even in the age of massive multi-player online games, games like Chess and Go still enjoy great popularity online. Using online games for covert communication is a natural extension of the ability to do so on paper, but with

the benefit of having the aide of a computer for embedding and extracting hidden bits.

Some of the techniques for hiding data in chess games have been recently rediscovered, and extended to cover modern advances in chess technology. Lange, in [26], and Desoky and Younis, in [27], present methods for hiding data in chess using the popular and common PGN notation. Lange identifies several locales, one in the game and one in the notation, where data hiding could take place. In the game itself, data can be encoded as sequences of legal moves. In the notation, data can be hidden in the commentary using techniques such as whitespace manipulation and linguistic steganography (c.f. [28–31]). Desoky and Younis further camouflage the chess covers by surrounding them with commentary (which could also be hiding data) and linking multiple chess covers together using themes (such as games played by a specific player). In both works, the stego-object, which consists of a chess game in PGN notation and associated commentary, encodes the entire message that Alice wants to send to Bob.

In the last 30 years, only a few covert channels in games have been discussed in the open literature. In [32], a covert channel in the card game Bridge is discussed which allows two partners to covertly communicate information about the contents of their hand in an environment where such communication is forbidden. The covert communication takes place during the bidding round, which proceeds in turns until one player has won the bid. However, the channel is only useful for passing information about one's hand.

Some single-player games can also be used to transmit hidden data. Niwayama, in [33], and Lee, Lee and Chen, in [34] and [35], present methods for hiding data in mazes. Their methodologies use the secret data to generate an image of a maze. In [36], Shirali-Shahreza presents a method for hiding data in Sudoku puzzles by encoding the secret data as a permutation of the numbers 1 through 9 and then building a Sudoku puzzle around this permutation (used as a column or row). Farn and Chen present methods for hiding data in jigsaw [37] and jig-swap puzzles [38].

In the jigsaw puzzle system, the data is encoded in the orientation of the connection between two adjacent pieces. In the jig-swap puzzle system, the data is encoded in the permutation of the pieces when they are first presented to the player. In both systems, the player must solve the puzzle before being able to extract the data.

Ou and Chen present a system for hiding data in the game of Tetris [39]. Tetris is normally a single player game, where the computer randomly selects the next tetromino to present to the player. However, 2-player versions of the game do exist and Ou and Chen's method is well suited to information hiding in such versions. Consider, for example, a modified game of Tetris where one player selects the next tetromino to give to the other player to stack. The game is won by the first player (the selector) if the second player fails to place the given tetromino on within the boundaries of the board within a certain period of time, as in the standard single player Tetris game. The second player (the stacker) wins if she is able to successfully stack a certain number of pieces received from the selector, a variation on the standard version of Tetris. In this version of Tetris, the selector can signal information to the stacker using the sequence of tetrominos he selects. Since there are 7 tetrominos in a standard game of Tetris, the selector can send, at most, an average of 2.8 bits per tetromino.

When two-player games are used for data hiding, they can be used in a bidirectional manner, allowing Alice and Bob to communicate with each other through the covert channel as the game is being played. In [40], Murdoch and Zielinski demonstrated a covert channel in the game of Connect Four, which they used for authentication during a tournament in order to collude and win. Hernandez-Castro et al. present a general methodology for using the move-choice covert channel in games with perfect information and apply it to the game of Go [41]. Their methodology points out that, whenever Alice has a choice of which move to make next, she can encode some bits of secret data in that choice. As long as Bob knows the set of moves from which Alice chose her move, he can extract the hidden data from the move. For each board state, Alice has a set of strategies which tell her which moves she can make

and how good each move is. She can use a threshold value to decrease the size of the candidate move list, which will decrease the number of bits the move will encode but also keep her play close to optimal.

In [42], Diehl points out that it is not necessary for a steganographic player to be play optimally or to play an equilibrium strategy. In fact, it is not even desirable that a steganographic player be optimal, since the security of the channel relies on the steganographic play being difficult to distinguish from authentic human play and humans seldom play optimally. Diehl provides an information-theoretic analysis of the information hiding capacity of games with perfect information such that the security constraint that stego-play be indistinguishable from human play is not violated. In the paper, Diehl examines a covert channel in the game of Kuhn Poker, which is a simplified two-person variant of the card game Poker [43]. The paper discusses mimicking human players by learning their strategy, but does not actually use human data. Diehl assumes a human player would play various strategies according to a stationary distribution on strategies. In the experimental results, it was shown that the stego-player required several hundred games before the stego-player's strategy distribution closely matched that of the optimal player it was trying to mimic.

Covert channels in online multiplayer games have also been discussed in the literature. In such an environment, the covert channel is a broadcast channel, since any player who can observe the sender can record the transmission, although only the intended receiver will know what to do with it. In [44] and [45], Zander, Armitage, and Branch present a method for hiding data in online multiplayer first-person shooter games. They apply their system, called RFPSCC for Reliable First Person Shooter Covert Channel (the first version was just FPSCC), to a modified version of the game Quake III Arena. The covert channel is implemented using slight but continuous variations in a player's movements (specifically, changes in the angle of the character's head), which are broadcast to the other players in sight of the broadcasting player by the central game server. Experimental results show that distribution of angle changes for a player using the covert channel is very similar to the distribution of angle changes

for player not using the covert channel, which suggests that the channel would have a high degree of security.

In [46], Zander discusses countermeasures and detection schemes for the covert channel in first person shooter games. FPSCC and, by extension, RFPSCC cannot be eliminated because player movement is intrinsic to the game. Further, Alice can overpower any noise Wendy adds to the channel since Wendy must keep her noise low enough that innocent players do not detect her tampering. While RFPSCC traffic looks very similar to normal game traffic, experimental results show that Wendy can detect the usage of RFPSCC, with an accuracy greater than 95% by observing Alice for at least 45-65 seconds. This is due in part to the structure RFPSCC must impose on the bits sent in order to provide synchronization for reliable transfer. RFPSCC also requires that both players use the covert channel even if only one is transmitting, which also contributes to Wendy's ability to detect the channel's usage.

2.3 Modeling Human Gameplaying Behavior

A secure covert channel in a game involves being able to mimic authentic behavior in that game. Understanding, i.e. predicting and explaining, human behavior is the holy grail of psychology. Which is to say that it is not our intention in this dissertation to solve the problem of modeling human game playing. However, models of human game playing can be used to build the foundation of a stego-system for games which generates gameplay which is statistically and computationally indistinguishable from gameplay generated by actual humans. This is possible because accurately distinguishing between authentic human gameplay and gameplay generated by a computer model requires the classifier to have access to a more accurate model of human gameplaying than the model to which the generator has access. If some model of human game playing can be said to be the best, that is, the model is one which generates the most human-like gameplay or is amongst a set of several models which are all equally good, then that model can be used to build a stego-system which is undetectable. If

an adversary wants to gain an advantage over the stego-system, they can only do so by developing a better model, one which generates more realistic gameplay and can therefore be used to distinguish between authentic gameplay and model-generated gameplay. Thus, in order to succeed, each side must, in essence, advance the science of modeling human gameplaying. In a very real sense, promoting covert channels in games may help to motivate research into modeling human gameplaying behavior, which in turn will advance our understanding of the human mind and how humans solve problems.

Game theory is often used to predict or explain the behaviors of two agents interacting in situations which can be cast as games [47]. However, game-theoretic models have not been very successful in predicting human behavior [48]. Psychological testing indicates that humans lack the two necessary abilities required to play in the manner described by game theory: the ability to compute the optimal probabilities for each move and the ability to select moves at random according to these probabilities [49]. The field of cognitive modeling offers many alternative theories for predicting and explaining human game playing behavior.

One class of theories from the field of cognitive modeling is that of cognitive architectures, which were proposed by Newell [50]. Cognitive architectures are theories about how human cognition works. They specify the mechanisms underlying cognition, often in considerable computational detail [49]. One of the more widely used cognitive frameworks is Anderson and Lebiere's ACT-R [51, 52], which stands for Adaptive Control of Thought - Rational.

In [53], Lebiere and West model human Rock-Paper-Scissors gameplay using ACT-R. Their model predicts the opponent's next move based on the most active sequence of previous moves. The model is based on psychological limitations and inclinations instead of the ideal strategies prescribed by game theory. In the experimental results, the model was found to closely account for human behavior in Rock-Paper-Scissors gameplay.

West and Lebiere showed in [54] that humans play Rock-Paper-Scissors like lag-2 neural networks which are punished for ties. While game theory predicts that a human player should tie with both a lag-2 network that was not punished for ties and a lag-1 network that was punished for ties, the network model predicts that a human player would beat both. West and Lebiere's experimental results showed that humans were able to reliably beat the two networks, a fact for which the game-theoretic solution could not account.

In [49], West, Lebiere, and Bothell argue that humans play maximally, not optimally. Whereas an optimal player conforms to game theoretic expectations that the player move according to fixed probabilities which specify an optimal approach to the game, a maximal player attempts to adjust it's responses to exploit perceived weaknesses in their opponent's play. One way of characterizing the two types of agents is to say that optimal agents try not to lose, while maximal agents try to win by as much as possible at the risk of losing. The authors find that standard game-theoretic models do not describe human game playing well, since humans try to exploit sequential dependencies (maximizing behavior) instead of trying to play optimally. The ACT-R cognitive architecture is able to account for this behavior and their model can play many games without modifying the basic strategy (one only needs to tell the model the rules). The authors entered their model in a Rock-Paper-Scissors tournament and found that the ACT-R model of human gameplay competes well against agents created specifically to play the game, as evidenced by the high ranking their model achieved in the tournament.

3 SYNTHETIC STEGANOGRAPHY IN TIME SERIES

In this chapter, we detail a methodology that enables an agent Alice to embed secret messages in public data that is sent or broadcast to a receiving agent Bob. While we have applied the method successfully in various settings, including patterns, textures and games, we focus here on random series because they (a) yield a powerful class of applications, and (b) require a special treatment. Thus, to begin, we focus on a (random) time series that Alice sends to Bob, and we study a number of steps involving data generation, message hiding, and message retrieval. For completeness, we demonstrate the process through experimentation with real data.

The first question to ask is: where does this time series come from? In a typical setting, Alice works in an application space where complex operations, perhaps involving groups of people or multiple transactions, proceed in sequence to generate unpredictable time-series values. To drive home this point, consider that Alice is a market-maker or specialist at a financial exchange where she is in control of a relatively illiquid stock. In such situations it is well-accepted that Alice or accomplices of Alice can exhibit control of price (or rearrange group activity) for periods of time that are sufficiently long to embed secret messages of nontrivial length in given price sequences. The second question to ask is: would it not be readily apparent to data viewers that the "controlled" data generated by Alice is suspect? The answer is no, because Alice can call upon a virtually unlimited history of prior data, under similar environmental conditions, to generate on-the-fly synthetic data that is impossible to distinguish from true data which is necessarily absent because synthetic data operates in its place.

For brevity, we will thus accept that Alice can generate synthetic data at will. We will equip her with a strong methodology to do so. Because we have motivated the financial specialist setting in prior work in great detail, we will simply focus on

how Alice generates on-the-fly data that cannot be distinguished from real data that would exist otherwise, and on how she embeds secret information for recipient Bob in this data. Observe that in the case of stock data, Alice generates a steganographic price series that is broadcast to the entire world via public market services. The methodology will make clear how Bob recovers hidden messages from synthetic data.

This chapter is laid out as follows: Section 3.1 presents a simple steganographic system for hiding information in an arbitrary series of values. Section 3.2 reviews methods for modeling time series data with various distributions. Section 3.3 applies the steganographic system of Section 3.1 to the time series generated by the methods of Section 3.2. Section 3.4 reviews the TES (transform-expand-sample) method of modeling time series with arbitrary distributions. Section 3.5 applies the steganographic system of Section 3.1 to the time series generated by TES. The contents of this chapter were accepted for publication in 2014 as an article in IEEE Latin America Transactions [55].

3.1 A Simple Steganographic System

We present a systematic method for hiding information in an arbitrary series of values. A secret key stego-system [56] is defined as a quintuple $\mathcal{S} = \langle C, M, K, D_K, E_K \rangle$, where C is the set of possible cover objects, M the set of secret messages with $|C| \geq |M|$, K the set of secret keys, $E_K : C \times M \times K \to C$ and $D_K : C \times K \to M$, with the property that $D_K(E_K(c, m, k), k) = m$ for all $m \in M$, $c \in C$ and $k \in K$.

To specify a stego-system, one must define the sets C, M, K and the functions E_K and D_K. The function E_K is the embedding function which takes as input the cover-object $c \in C$ and the message $m \in M$ to be hidden in c as well as any additional (key) parameters $k \in K$ required to hide the information. It returns a stego-object containing the hidden information. The function D_K is an extracting function which takes as input a stego-object c' and key k and outputs the message m which is hidden in the stego-object.

For the purposes of this chapter, we define the set C to be the set of positive real numbers $\mathcal{R}^+$, and set M to be the set of binary strings of arbitrary length, i.e., $M = \{0,1\}^n$ for $n \geq 0$. In the remainder of this section we will develop our definitions of the embedding and extracting functions, E_K and D_K, by starting with a simple function and adding layers of complexity to create a more general function.

3.1.1 Embedding Function

A family of embedding methods emerges from a simple example of embedding. To hide a binary string — a representation of some secret information — in time series data, we divide the real number line into cells based on some set of parameters and label these cells alternately as 0 and 1, beginning by labeling the first cell 0. The information is embedded by allowing the label sequence of the data to match message bits. To do this, align the first bit of the message with the first value of the cover-object and compare each bit of the message with the label of the corresponding data value. If the data label matches the message bit, the data is left unperturbed. If the data label differs from the message bit, however, the data value is minimally augmented so that the label changes. A simple policy for this would be to bump 0s up to 1 and 1s down to 0 by adding ± 1 cell-width to the data value.

This stego scheme can have several layers of complexity based on the key (set of parameters) used. We assume the key is exchanged securely prior to stego transmission.

Layer 0: In its simplest form all cell widths are a constant value $w = \frac{1}{\alpha}$, yielding the following embedding function:

$$S'(i) = S(i) + w\,(M(i) - m(S(i), w)) \tag{3.1}$$

where $m(s, w) \equiv \left\lfloor \frac{s}{w} \right\rfloor \pmod 2$. The key for the layer-0 stego-system is $k = \{\alpha\}$.

Layer 1: In the next layer of complexity we may allow an offset value equal to some constant δ_0, so that the embedding can begin at any point in the coverdata, not just

at the very beginning. This allows us to control the location of the information within the covertext. The embedding function is:

$$S'(i) = S(i) + I_{\{i>\delta_0\}} w\left(z_i(\delta_0) - m(S(i), w)\right) \tag{3.2}$$

where $z_i(j) = M(i-j)$ and $I_{\{cond\}}$ is 1 if and only if *cond* is true and 0 otherwise. The key for this system is $k = \{\alpha, \delta_0\}$.

Layer 2: Once we have the machinery to handle a constant message offset, we can easily handle using only certain values to hide message bits. To do so, we use a monotonically increasing function as a variable offset. It must be monotonically increasing or else the function will try to embed two message bits in the same value, which would cause the first message bit to be lost. We choose whether or not to use the current data value to embed a message bit by tossing a coin that lands heads up with probability p. If this event occurs, we embed the value, and otherwise skip it. Embedding a message bit in the current value does not increase the offset. But, skipping a value does. So, we need to keep track of our offset as we go. We do this with the $\delta(i)$ function defined in Equation 3.4. This additional layer of complexity allows us to control the distribution of the message within the covertext. The embedding function is now

$$S'(i) = S(i) + I_{\{i>\delta(i)\}} w\left(z_i(\delta(i)) - m(S(i), w)\right) \tag{3.3}$$

where

$$\delta(i) = \delta(i-1) + I_{\{r_1(i)>p\}},\ \delta(0) = \lfloor r_1(0) + 0.5 \rfloor \tag{3.4}$$

The key for this system is $k = \{\alpha, p, R_1\}$, where R_1 is a random number generator which generates $r_1(i)$.

Layer 3: An additional (and, for now, final) layer of complexity can be had by allowing cell widths to vary. We accomplish this by using the function $w(i) = 1/r_2(i)$. The embedding function is now:

$$S'(i) = S(i) + I_{\{i>\delta(i)\}} w(i) \left(z_i(\delta(i)) - m(S(i), w(i))\right) \tag{3.5}$$

The key for this system is $k = \{p, R_1, R_2\}$, where R_2 is a random number generator to generate $r_2(i)$, which replaces the previously used α.

The above becomes our embedding function E_K, which allows us to specify through the key k: the granularity of the embedding (through the specification of R_2) and the spread of the information through the cover-object (via the specification of R_1 and value of p). This function completely captures the behavior of each of its predecessor layers. To implement Layer-0, let R_2 be a stream with constant value α, set $p = 1$ and let R_1 be any stream who's first value is 0.

3.1.2 Extracting Function

To extract a hidden message from a data series we need to know the key k. Once we know k, we split the data range into cells based on the parameters used and label them as before, alternating between 0 and 1. The message is the bit string obtained by simply reading off the labels of the data values. Thus, the extracting function is simply a labeling function.

Unlike the case of the embedding function, where each new layer of complexity further complicates embedding, the extracting function's form stays very much the same even despite the complexity layers. The extracting functions are listed below in quick succession as Equations 3.6-3.9.

$$M(i) \equiv \left\lfloor \frac{S'(i)}{w} \right\rfloor mod\ 2 \tag{3.6}$$

$$M(i) \equiv \left\lfloor \frac{S'(i+\delta_0)}{w} \right\rfloor mod\ 2 \tag{3.7}$$

$$M(i) \equiv \left\lfloor \frac{S'(i+\delta(i))}{w} \right\rfloor mod\ 2 \tag{3.8}$$

$$M(i) \equiv \left\lfloor \frac{S'(i+\delta(i))}{w(i+\delta(i))} \right\rfloor mod\ 2 \tag{3.9}$$

3.1.3 Good Cell Widths

Through the random number generator R_2, we have control over the range of our cell widths. Cell widths which are too large will result in large data perturbations which may be easily detectable. However, the embedding is fairly tolerant of external augmentation since a large perturbation is needed to bump data values out of their actual labels. Conversely, cell widths which are too small will result in small perturbations which succumb to external augmentation very easily. But here, however, the embedding is difficult to detect since data values exhibit negligible change, and the underlying structure of the original data is preserved.

A good cell width, then, is one which balances two requirements: (1) minimizes the evidence of embedding and (2) maximizes the tolerance of the embedding to external perturbations. If we know the maximum amount by which data values may be perturbed, we can calculate the minimum cell width we are required to have so that the probability that a value survives the perturbation is at least some value q. The cell width required is given by Equation 3.10.

$$w \geq \frac{2p_{max}}{1-q}, \tag{3.10}$$

where $q \in (0,1)$ is the probability that a value survives the maximum perturbation p_{max}.

Using this fact along with $w(i) = 1/r_2(i)$ from above, we can compute an upper and a lower bound for good random variates from R_2, shown in Equation 3.11 where w_{max} is the maximum cell width allowed.

$$\frac{1}{w_{max}} \leq r_2 \leq \frac{1-q}{2p_{max}}, \tag{3.11}$$

3.2 Methods For Modeling Time Series

To illustrate the above ideas, we focus on time series embeddings. There are two main classes of methods used to model time series data: autoregressive (AR) and moving average (MA) models. The choice of model is based on the properties of the data one is trying to model. Autoregressive models are useful in modeling stationary data, while moving average models are better equipped to handle non-stationarity. It is possible to model non-stationary data with an autoregressive model by using differencing. In this section we briefly present the first order autoregressive model and how to use it to model data with various distributions. While many time series can be modeled by a Gaussian process, many naturally occurring time series, however, are non-Gaussian. For this reason, several different methods for generating non-Gaussian time series have been constructed [57]. These include models which utilize exponential [58, 59], Laplace [60] and gamma [61] marginals.

3.2.1 Autoregressive Model

The first order autoregressive model is given by Equation 3.12, where ϕ is the parameter, c is a constant (often $c = 0$ for simplicity) and ϵ_t is the noise term. Note that ϵ_t can also be considered an error or innovation term.

$$X_t = c + \phi X_{t-1} + \epsilon_t \tag{3.12}$$

Gaussian Autoregressive Model. If the noise term ϵ_t is a Gaussian process, then X_t is also a Gaussian process. The resulting model is the first order Gaussian autoregressive model GAR(1).

Laplace Autoregressive Model. If the noise term ϵ_t is defined

$$\epsilon_t = \begin{cases} 0 & w.p. \quad \phi^2 \\ L & w.p. \quad 1 - \phi^2 \end{cases} \tag{3.13}$$

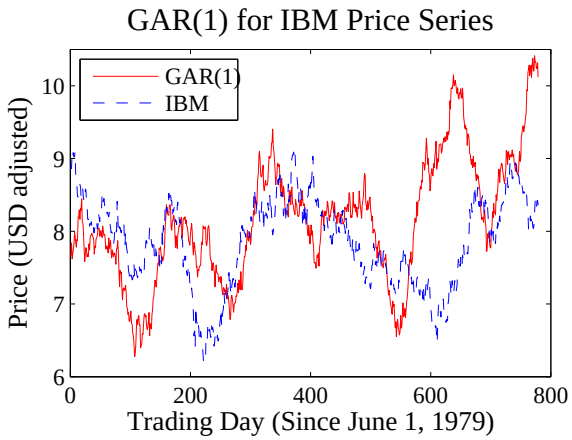


Figure 3.1. Synthesized GAR(1) for IBM.

where L is a Laplace distributed random variable, the sequence X_t is a stationary time series with Laplace marginal distribution for all t [60].

Exponential Autoregressive Model. The conventional exponential first-order autoregressive model EAR(1) [58] has the form

$$X_t = \phi X_{t-1} + \begin{cases} 0 & w.p. \quad \phi \\ E_t & w.p. \quad 1-\phi \end{cases}, t = 0, 1, 2, \cdots \tag{3.14}$$

where $0 \leq \phi < 1$ is a parameter and the $E_t, t = 0, 1, 2, \cdots$, are independent exponential variables with parameter $\lambda > 0$. This model generates paths in which large values are followed by runs having geometrically distributed lengths of falling values [59].

3.3 Steganography with AR Models

We are now ready to present steganographic methods based on synthetic data and, in particular, data generated via the methods of Section 3.2. For ease of explanation we will use the GAR(1) model as our example. Fig. 3.1 shows an example synthetic data series generated to model an IBM stock price trajectory where prices are reported at end-of-day. The methods apply to any time scale, including tick, minutes, days, weeks and months.

3.3.1 Embedding and Extracting

Embedding in data generated via an AR model such as GAR(1) can be done in at least two ways, which is to say that the message can be hidden in at least two places in the model: the innovations and the final time series. The choice of where to embed decides which information is necessary in the key in order to extract the message. Embedding in the innovations requires the key to contain ϕ whereas embedding in the final time series does not require ϕ to be known.

Extracting is accomplished by isolating the portion of the model which was used to hide the message and applying the extracting function in order to extract the hidden information.

3.4 Modeling Time Series With TES

The TES (transform-expand-sample) [62,63] methodology is a versatile method for generating a class of stochastic stationary time series which exhibit general marginal distributions and a broad range of dependence structures. TES is designed to generate sequences which satisfy the following three goodness-of-fit requirements [62], in descending order of rigor:

1. The marginal distribution of the sequence should match its empirical counterpart.
2. The autocorrelation of the sequence should approximate its empirical counterpart well.
3. The trajectory of the sequence should "resemble" that of the empirical data.

3.4.1 TES Methodology Outline

In [62], the TES methodology is enabled through execution of the follow steps:

1. Construct histogram H of the base data

2. Using H, construct the CDF of the base data
3. Using the CDF, invert H to obtain H^{-1}
4. Construct the density f_V of innovation sequence V
5. Choose the sign of the TES class (TES+ or TES–)
6. Generate initial random variate $U_0 \in (0, 1)$
7. Using U_0 and V, generate U
8. Choose stitching parameter ξ
9. Compute background process $Y_i = S_\xi(U_i)$
10. Compute foreground process $X_i = H^{-1}(Y_i)$
11. Compare the distribution and autocorrelation of X with those of the base data. If the two are similar to their empirical counterparts, compare the appearance of X with that of the base data. If X is unacceptable, repeat the above from any previous step; otherwise select X as the final TES process.

In this work, we have used a similar set of steps to implement the TES methodology. Instead of constructing a histogram off the base data, we directly construct the empirical CDF. Also, we only use the TES+ class, though nothing in our methodology precludes the use of the TES– class. The choice to use TES+ was made based on the data we are trying to model. Instead of an innovation density f_V, we use an innovation CDF F_V. Having the innovation CDF is equivalent to having the innovation density and enables a more precise path. While initial parameter values can be set arbitrarily, in this work we have chosen to precompute suitable values — a process that is explained later. Our modified TES methodology is as follows:

1. Construct the empirical CDF F of the base data
2. Choose stitching parameter ξ

3. Construct the empirical CDF F_V of innovations

4. Generate innovation sequence V

5. Compute unstitched background sequence U

6. Compute stitched background process $Y = S_\xi(U)$

7. Compute foreground process $X = F^{-1}(Y)$

8. Compare the distribution and autocorrelation of X with those of the base data. If the two are similar to their empirical counterparts, compare the appearance of X with that of the base data. If X is unacceptable, repeat the above from any previous step; otherwise select X as the final TES process.

In the remainder of this section we discuss our implementation of the TES procedure in more detail.

3.4.2 Empirical CDF

The empirical CDF is a 2-dimensional vector which tells us what proportion of the data is less than a given value. The first dimension of the vector is the X-axis and covers the range of the data values. The second dimension contains the proportion of the data which is less than the corresponding X-axis value.

$$F(x) = \frac{1}{|D|} \sum_{d \in D} I_{\{d<x\}} \tag{3.15}$$

where $I_{\{cond\}} = 1$ iff *cond* is true and 0 otherwise.

The empirical CDF can be smoothed through the use of interpolation. Since our CDF is a vector, obtaining its inversion is simple. The inverse CDF is the distortion used to transform the background process Y into the foreground process X, i.e., $Y = F(X)$ and $X = F^{-1}(Y)$.

When X is a foreground process, such as the base data, evaluating the CDF at each of the points in X recovers the stitched background process Y. We use this fact later when constructing a good innovation density.

3.4.3 Stitching Transformations

The purpose of the stitching transformation S_ξ is to smooth the generated time series while preserving uniformity. The necessity of such a function is apparent when we consider the sequence of values comprising U. Each U_n is the sum of the previous value U_{n-1} and an innovation value V_n taken modulo 1. We may regard U as a random walk around the unit circle. When we cross over the zero-one boundary of the circle, the value drops from a large fraction to a small fraction or vice versa, depending on the direction we cross the boundary. Left as it is, this large jump would result in an even larger jump in the foreground process, say from near the minimum value to near the maximum. In certain cases, this may be acceptable, but in general it is undesirable.

A stitching transformation S_ξ maps the interval $[0, 1)$ to itself and is determined by a stitching parameter $\xi \in (0, 1)$. For a given ξ, the stitching transformation is defined in [62] as

$$S_\xi(y) = \begin{cases} \dfrac{y}{\xi}, & 0 \le y \le \xi \\ \dfrac{1-y}{1-\xi}, & \xi \le y < 1 \end{cases} \tag{3.16}$$

While it is possible to invert a stitching transformation, doing so requires additional information beyond the stitched value. Due to the piecewise definition of the stitching transformation, the direction the original value was stitched is required in order to invert it. Therefore, we must record this direction if we wish to invert the stitching transformation in such a way as to obtain the true original value.

$$y = S_\xi^{-1}(x) = \begin{cases} \xi x, & d_\xi(y) = 1 \\ 1 - (1-\xi)x, & d_\xi(y) = 0 \end{cases} \tag{3.17}$$

where $x = S_\xi(y)$ and $d_\xi(y)$ is defined as

$$d_\xi(y) = \begin{cases} 1, & 0 \leq y \leq \xi \\ 0, & \xi \leq y < 1 \end{cases} \tag{3.18}$$

If the stitching directions are unknown, one of several approaches can be taken: 1) randomly choose the direction, 2) choose a string of directions and repeat it over the length of the data, or 3) choose the direction which results in the smallest difference (mod 1) from the previous unstitched value.

3.4.4 Innovation Sequence

The innovation sequence V is a sequence of random numbers drawn from the innovation density f_V. The innovation sequence tries to approximate the underlying pseudo-random process occurring in the original data. The values in the innovation sequence, called innovations, are transition values, i.e., amounts by which a next value in the sequence differs from a previous value. The innovation sequence is used to generate the background process which is then transformed into the foreground process which becomes a candidate for the final TES series.

3.4.5 Innovation Density

In [62, 63], the innovation density f_V consists of $K > 0$ non-overlapping regions, called steps, whose positive heights sum to 1. Each step k is therefore represented by a 3-tuple (L_k, R_k, P_k), where L_k and R_k are the left and right endpoints of the step, and P_k is the height. Thus,

$$f_V(x) = \sum_{k=1}^{K} 1_{[L_k, R_k)}(x) \frac{P_k}{\alpha_k}, x \in [-0.5, 0.5) \tag{3.19}$$

where $\alpha_k = R_k - L_k$ is the width of step k. The innovation density is essentially the desired histogram for the innovation sequence. To generate innovations from the

density as defined above, with probability P_k we generate a uniform random number between L_k and R_k.

Since the innovation density is one of two parameters we can use to find the best foreground process, it is important to choose a good one. Our method for finding a good innovation density gives us the empirical CDF of the innovation sequence, which is equivalent to having the density f_V.

3.4.6 Background Process

Given the explanation of stitching above, we make a distinction between the stitched background process Y and the unstitched background process U. The TES methodology divides unstitched background processes into two classes, TES+ and TES–. TES+ consists of sequences U_n^+ of the form

$$U_n^+ = \begin{cases} U_0, & n = 0 \\ \left\langle U_{n-1}^+ + V_n \right\rangle, & n > 0 \end{cases} \tag{3.20}$$

and TES- consists of random sequences U_n^- of the form

$$U_n^- = \begin{cases} U_n^+, & n \text{ even} \\ 1 - U_n^+, & n \text{ odd} \end{cases} \tag{3.21}$$

The choice of whether to use TES+ or TES– depends on the autoregression of the original data. An oscillatory autoregression is modeled more easily by TES-.

The stitched background process Y is obtained by applying a stitching transformation to U, $Y = S_\xi(U)$.

3.4.7 Foreground Process

The final product of the TES methodology is the foreground process X, which should have the same marginal distribution and autocorrelation as the base data and should also "resemble" the base data in its path. The foreground process is

generated by applying a distortion to the background process. Conceptually, while this distortion could be any monotonic function, TES uses the inverse CDF of the base data. Since the CDF is a monotonically increasing function, it is always invertible. The equation for obtaining X from the background process Y is $X = F^{-1}(Y)$.

Searching for Good Parameter Values

We compute good initial model parameters (F_V and ξ) by viewing the base data as a TES foreground process and reversing the TES methodology to find the innovation sequence. We then use the empirical CDF of the innovation sequence as the CDF of our future innovation sequences. In order to do this, we must know ξ, and so we find a suitable ξ along the way. The process works as follows:

1. Evaluate F at each of the points in the base data, obtaining Y'
2. For each of several candidate values of ξ,
   (a) Unstitch Y', obtaining U'
   (b) Difference U', obtaining V'
   (c) Construct the empirical CDF of V', $F_{V,\xi}$
   (d) Generate an innovation sequence V_ξ
   (e) Generate an unstitched background sequence U_ξ
   (f) Stitch U_ξ, obtaining Y_ξ
   (g) Evaluate the empirical inverse CDF at each of the points in Y_ξ, obtaining X_ξ
   (h) Construct the empirical CDF F_ξ of X_ξ
3. Let $\hat{\xi}$ be the ξ which minimizes $\sum |F - F_\xi|$
4. $F_V = F_{V,\hat{\xi}}$

We use the sum of the absolute values of the differences between the empirical CDF of the base data and that of a foreground process X_ξ generated with stitching parameter ξ as a goodness test for the candidate values of the stitching parameter. Once we know a good ξ, we can compute the empirical CDF of the innovation sequence obtained by differencing the unstitched background process. Other measures of goodness may also used, such as, for example, similarity of autocorrelations and visual resemblance.

When unstitching the background process, since we do not know the original direction the values were stitched (we cannot know because they were not actually stitched), we must make a guess as to what the correct directions to unstitch the values are. There are several ways to do this and we have chosen to unstitch in the direction which results in the smallest difference (mod 1) from the previous unstitched value. This approach keeps the innovation sequence values small and results in a more precise innovation density.

When differencing the unstitched background process, due to the modulo-1 arithmetic used, sometimes this difference will exceed the innovation boundaries, in which case the innovation simply needs to be in the other direction. That is, you need to cross over the zero/one boundary to get to the next value using a valid innovation value as shown in Equation 3.22.

$$V_n = \begin{cases} U_n - U_{n-1}, & -0.5 < U_n - U_{n-1} < 0.5 \\ U_n - U_{n-1} + 1, & U_n - U_{n-1} < -0.5 \\ U_n - U_{n-1} - 1, & U_n - U_{n-1} > 0.5 \end{cases} \tag{3.22}$$

3.5 Steganography with TES

We are now ready to discuss how the TES methodology can be used for a stego-system based on financial series (i.e., stock market) data.

3.5.1 Data-hiding locales

TES allows us to embed a message in several places. However, the choice of where to embed mainly affects the complexity of the key needed to recover the message. The effect on the final foreground process is negligible.

Innovation Sequence

If Alice hides the message in the innovation sequence, the stego-V (see Fig.s 3.6 through 3.9 for the different possible arrangements) is used to make the BP (background process) which is then stitched and finally transformed into the FP (foreground process). To recover the message, Bob (the receiving accomplice) must take the FP, transform it back into the stitched BP, unstitch the BP and then find the innovation sequence, which contains the message. To do all this, he must know the transformation from BP to FP, the stitching directions, the initial random variate (only really needed for very first bit; but if first bit is agreed to always be garbage, then it isn't actually required by the receiver) and the key to the secret key stego-system.

BP prior to stitching

If Alice hides the message in the BP prior to stitching, the stego-BP gets stitched then transformed into the foreground process. Bob must then know the transformation to obtain the stitched BP. Then, he must also have the stitching directions in order to unstitch the BP. Once unstitched, extracting the message using the stego-system key is simple.

BP after stitching

If Alice hides the message in the BP after it has been stitched, the stego-BP is just transformed into the FP. Bob must then know the transformation to obtain the

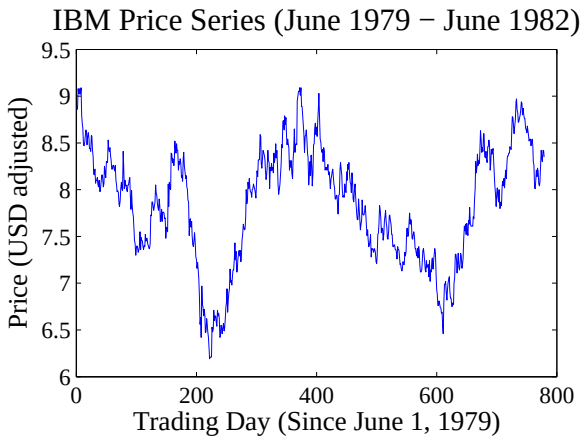


Figure 3.2. IBM price series data.

stitched BP, which contains the hidden message. Again, extracting is simple using the stego-system key.

FP

If Alice simply hides the message in the FP, our accomplice just extracts it directly from there and so he/she only needs to know the stego-system key. As should be clear from the prior explanation, TES is only used synthesize the data.

3.6 Experimental Results

We demonstrate the steganography methodology through use of an actual IBM stock price trajectory (see Fig. 3.2) for the period of June 1979 through June 1982. This series becomes our base dataset.

TES is able to match the distribution and autocorrelation of the base data closely, and also produces a trajectory that shows a good visual resemblance to the base data. This is borne out by the close agreement of the CDF curves in Fig. 3.3, and the virtually identical autocorrelations in Fig. 3.4.

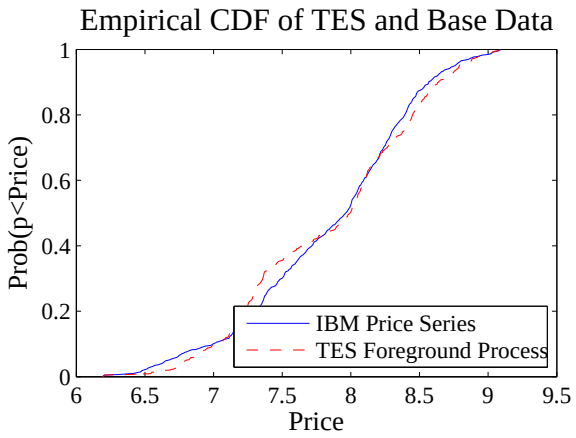


Figure 3.3. TES foreground process and base data empirical CDFs.

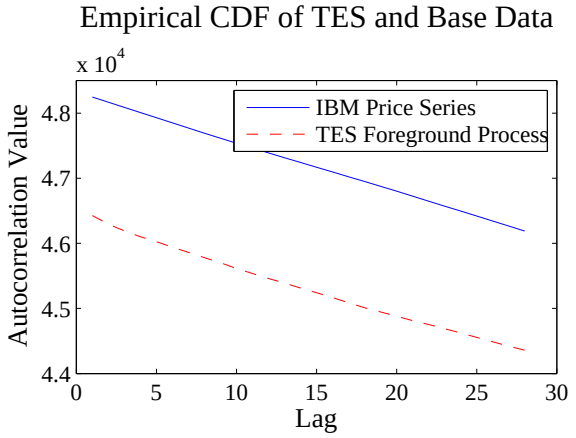


Figure 3.4. TES foreground process and base data autocorrelations.

In general, our experiments show that Alice is able to generate a suitable synthetic data series very rapidly, through either an automatic selection procedure or through visual selection. In Fig. 3.5, for example, the foreground process that TES offers Alice yields identical second order properties and close visual resemblance to patterns in the base data, which is what Alice wants. Information can be hidden in several places during the TES procedure. The embedding location chosen has a negligible effect on the foreground process.

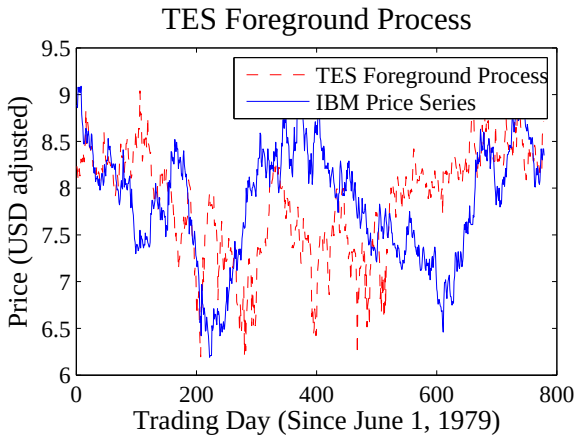


Figure 3.5. TES foreground process with good autocorrelation match.

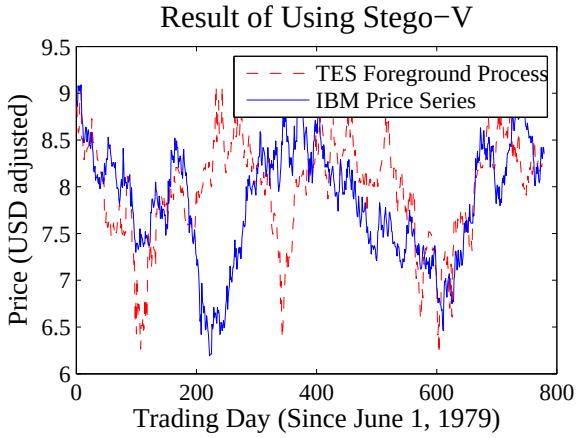


Figure 3.6. TES foreground process using stego-V.

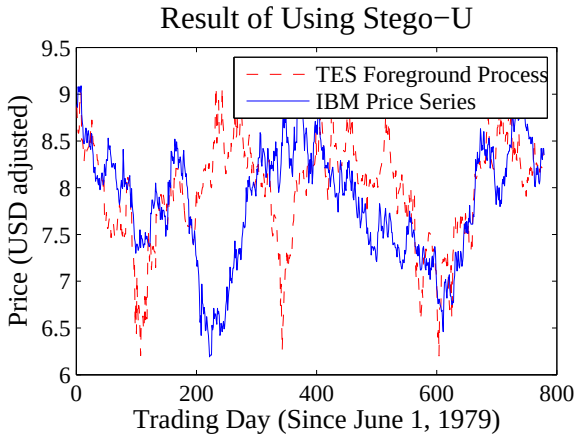


Figure 3.7. TES foreground process using stego-U.

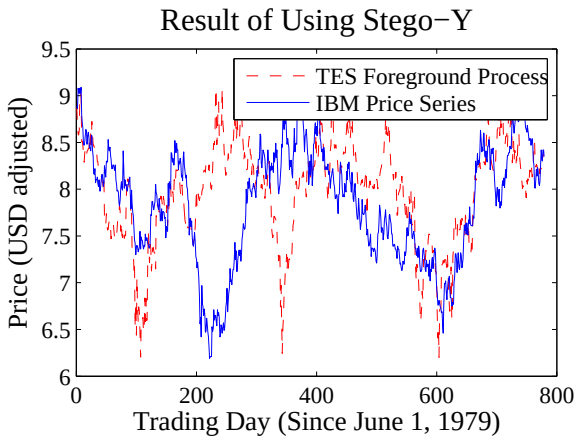


Figure 3.8. TES foreground process using stego-Y.

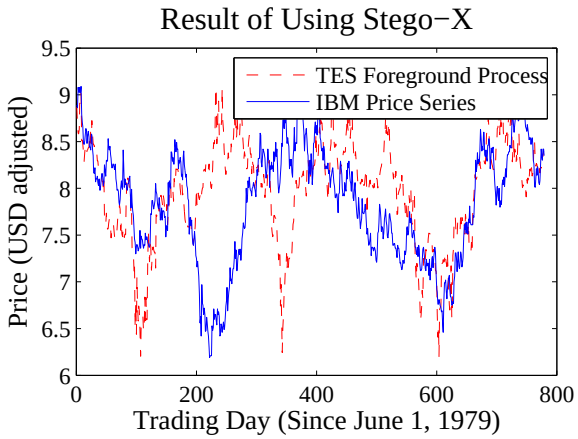


Figure 3.9. TES foreground process using stego-X.

3.7 Conclusion

Our experiments have shown that we can develop relatively sophisticated and practical secret-key stego-systems in a variety of applications including the kinds seen in financial markets. Layers of complexity enable information-hiding in an arbitrary series of values with control over information density, location and size of perturbation. We presented the TES methodology for modeling time series, with our implementation of the procedure, outlining the differences between our methods and ones laid out by Melamed [62, 63]. By piggy-backing the proposed stego-system on TES, we are able to embed information in price series at various stages of the TES procedure. Our experience is that hiding effects on the final foreground process are negligible, and the main effect of the choice of where to do the embedding shows up in the complexity of the key required to recover the hidden information.

4 A CONTEXT SENSITIVE TILING SYSTEM FOR INFORMATION HIDING

In this chapter, we explore the merits of a new type of image-steganographic scheme based on context-sensitive tilings. The goal is to drive payload embedding into the high level structure of an image in order to produce stego-objects with higher levels of robustness to tampering and security against steganalytic attacks. The class of images thus produced can be said to have both structure and semantics. The images are procedurally generated with the help of the secret payload and a set of rules, so that the structural content of the generated image is in itself the payload. Such a payload is easily recovered by reversing the embedding process. Here, we must emphasize that the secret information is not being hidden in an image. Instead, the secret payload is actually made to *define* an image which is unique to both payload and key. The contents of this chapter were published in 2012 as an article in the Journal of Information Hiding and Multimedia Signal Processing [64].

4.1 A Context Sensitive Tiling System

In this section, we define a system for tiling a rectangular grid with uniformly-sized tiles in such a way that each tile depends, to some extent, on it's neighbors. We require uniformly-sized rectangular tiles so that any $R \times C$ grid consisting of R rows and C columns of tiles will be rectangular. The basis of our context sensitive tiling system is a 2-Dimensional Context Sensitive Grammar (2D-CSG). A 2D-CSG is a generalization of a Context Sensitive Grammar (CSG) wherein the context for each production rule includes the strings above and below as well as to left and the right of the non-terminal being replaced [65].

Definition 4.1.1 (Context-Sensitive Tiling System) *A Context-Sensitive Tiling System T is a 4-tuple $T = (V, \Sigma, R, S)$ where V is a finite set of non-terminal symbols,*

Σ *is a finite set of terminal symbols disjoint with* V*,* R *is a finite set of production rules of the form*

$$\begin{array}{ccc} \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & A & \alpha_5 \\ \alpha_6 & \alpha_7 & \alpha_8 \end{array} \rightarrow \begin{array}{ccc} \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \gamma & \alpha_5 \\ \alpha_6 & \alpha_7 & \alpha_8 \end{array}$$

where $A \in V$*,* $\alpha_i \in (V \cup \Sigma)$ *and* $\gamma \in (V \cup \Sigma)$*, and* $S \in V$ *is the initial non-terminal.*

A tiling system is just a 2D-CSG with a modified production rule set which limits its production rules in two ways. First, only the 8 individual symbols surrounding the non-terminal under consideration may be used. Second, the non-terminal under consideration may be replaced only by a single symbol.

4.2 Steganography

A tiling system generates a grid of tiles using the set R of production rules. Because choice implies variety, and variety offers a potential for discernment, every situation that offers a rule choice simultaneously offers an opportunity to hide something meant only for a discerning receiver. With n candidate production rules, the choice can embed any combination of $\lfloor \log_2 n \rfloor$ bits. A natural order can be imposed on the production rule set by, for example, sorting them lexicographically by the right hand side. Thus, in choosing between at least 8 applicable rules, the choice of which rule to use can encode at least 3 bits, i.e., the embedding lies in the *choice*. The unused or excess rules can function as null generators, placing symbols to be ignored during the decoding/extracting process. Or, the would-be unused rules can be shuffled in over the course of the embedding process using a shared random number generator so that every rule is equally likely to be used (conditioned on the neighbors). The shuffling of the rules corresponds to encrypting the message using a one-time pad generated from a shared random number generator. Equivalently, we can consider choices of symbols or tiles, instead of rules, as the engine of the embedding.

Algorithm 1 presents a method for generating a stego-object using a context-sensitive tiling system and a pseudo-random number generator. Alice starts in the

Algorithm 1: Stego-Object Creation.

Input: Input: Bit-String mssg, Tiling System TS, PRNG R

Output: Grid G

G = RHS of TS's initial production rule S

loc = the initial position in grid G

repeat

T = CandidateTileSet(loc,TS)

n = min(Capacity(loc),$|mssg|$)

$d = mssg[0 : n-1]$

$mssg = mssg[n : end]$

$T2$ = RandomPermute(T,R)

dn = BinaryToInteger(d)

$G[loc] = T2[dn]$

loc = next position

until $|mssg| == 0$;

foreach *remaining loc in* G **do**

T = CandidateTileSet(loc,TS)

n = Capacity(loc)

d = RandomBitString(n)

$T2$ = RandomPermute(T,R)

dn = BinaryToInteger(d)

$G[loc] = T2[dn]$

loc = next position

end

top-left corner of the object and constructs the candidate tile set, which is the set of symbols which occur on the right-hand side of the production rules in R which are applicable to the current location. Alice then determines the number of secret message bits the location can encode, which is at most the logarithm of the size of the candidate tile set, and extracts that number of bits from the message. She then randomly permutes the candidate tile set and chooses the tile which ends up in the position indexed by the integer value of the secret message bits before moving on to the next location. This process is repeated until the message is consumed. Any remaining tiling is completed by generating random bits to use in place of the message, which is equivalent to selecting tiles at random from the candidate tile sets, until the object is completely tiled.

Algorithm 2: Stego-Object Decoding.

Input: Grid G, Tiling System TS, PRNG R

Output: Bit-String mssg

loc = the initial position in grid G

repeat

T = CandidateTileSet(loc,TS)

n = Capacity(loc)

$T2$ = RandomPermute(T,R)

dn = index of $G[loc]$ in $T2$

d = n-bit binary representation of dn

$mssg$ = Concatenate($mssg$,d)

loc = next position

until *loc not in* G;

Algorithm 2 presents the method for decoding the stego-objects generated by Algorithm 1. Starting in the same place that Alice started, Bob also constructs the candidate tile set, determines the number of bits that could have been encoded in this location and randomly permutes the candidate tile set (Alice and Bob share the

same PRNG and seed so that they will both generate the same random permutation). Bob then determines which element of the permuted set Alice chose and records the binary value of this index as the secret message bits before moving on to the next location. Bob repeats this process until the entire object has been decoded.

4.2.1 An Example of the Proposed Scheme

Suppose Alice wishes to send a message to Bob using the above proposed scheme using the tile set shown in Figure 4.3 and Rule Set 1 (shown in Equation 4.5). Let the message she wishes to send be $m = 88888888$, a decimal number having eight eights. The decimal number 88888888 is expressed in binary as `10101001100010101100011100`, which has 27 bits. Alice will send this 27 bit sequence preceded by an 8-bit content-length field that will tell Bob how many bits to read after the first 8. Thus, the full data payload that Alice will send is: `000110111010100110001010110001110 00`, which has length 35 bits. Alice knows from Table 4.1 that Rule Set 1 encodes at least 2 bits per tile and so knows that her 35-bit message will require no more than 18 tiles. Alice rounds up to the nearest square and decides her tiling will be $5 \times 5 = 25$ tiles. By convention, and to simplify this example, Alice and Bob use blank initial state tiles. The will also mirror their images horizontally and vertically. Mirroring this way makes the image more visually appealing and has the effect of making the tiling immune to rotations and reflections as well as more robust to other tampering from Wendy. Alice and Bob will use a shared key with value 12345, which is used as the seed r_0 for their pseudo-random number generators (PRNGs). For this example, Alice and Bob will use the following 32-bit linear congruential generator as their PRNG:

$$r_{n+1} = \lfloor \frac{1}{2^{16}} \left(25214903917 \cdot r_n + 11 \pmod{2^{48}}\right) \rfloor. \tag{4.1}$$

To begin, Alice sets out a 6×6 grid of nonterminal open symbols, which will be replaced with terminal tiles as the tiling is constructed. She replaces the top row and the first column of open symbols with blank tiles, per her shared convention with Bob.

Then, she begins with the first open symbol in the top right corner and considers which candidate tiles she can place at this location given the three neighbors to the North, Northwest and West. Under Rule Set 1, given three blank neighbors, the candidate tile set contains the tiles {0,3,5,6,7}, which means Alice can hide 2 bits of secret in her choice of tile for this location. She removes these 2 bits from the message (these first 2 bits are `00`) and converts the bitstring to a decimal number, in this case 0. She rolls her PRNG once and uses the result to generate a permutation of the candidate tile set, in case she obtains {0,6,3,5,7}. She selects the 0th tile from this permuted set (because the data segment she is hiding has decimal value 0) and replaces the open symbol at the current location with the selected tile. The next location is the grid cell directly adjacent to the east of the current location. Alice again considers the neighbors (all blank tiles) to obtain the candidate tile set ({0,3,5,6,7}), extracts 2 more bits of secret (bitstring is `01`), uses the PRNG once to randomly permute the candidate tile set ({7,0,5,3,6}), selects the permuted candidate tile whose index is the decimal value of the secret bitstring (another 0 tile) and finally she moves to the next location. At this location, the neighbors are all 0 tiles still, the secret bitstring is `10`, the permuted candidate tile set is {0,3,5,7,6} and so the tile Alice selects at this step is the 5 tile. At the next location, the neighbors are the tiles {0,0,5}, the secret bitstring is `11`, the permuted candidate tile set is {15,12,9,14,13,10,11} and so the tile that Alice selects at this step is the 14 tile. She will repeat this process until she runs out of secret message to send, at which point she will start hiding random bits. The state of the tiling after each iteration is shown in Figure 4.1. Once she has set the last tile, she trims the initial state tiles from the edges and mirrors the image twice, once horizontally and once vertically, before sending the final image, shown in Figure 4.2, to Bob.

When Bob receives the image, he can easily reverse the last steps that Alice took before sending the image. He undoes the mirroring (or uses it to undo any tampering by Wendy) and adds back in the initial state tiles around the edges (he knows these because he and Alice have a prior agreement on what they are or how to generate

Figure 4.1. Snapshot of the example tiled image after each tiling iteration. The 'O' symbols, which are the nonterminal open symbols, and the border are shown for illustration purposes only.

them) to obtain an image from which he can extract the secret data. Bob starts in the same location that Alice started, but instead of an open symbol Bob sees a tile in the first location. Nonetheless, Bob uses the three neighbors to construct the candidate tile set, uses the size of the set to determine how many bits to expect, uses his PRNG and the key he shares with Alice to randomly permute the candidate tile set and then finds the index in the permuted set of the tile which he finds in the current location. In this case, the permuted tile set is {0,6,3,5,7}, the tile in the current location is 0, which is in index 0, and the number of bits to expect is 2. Therefore, Bob records the first 2 bits of the secret as `00`. In the next location, the permuted tile set is {7,0,5,3,6}, the tile in the current location is 0, which is in index 1, and the number of bits to expect is 2, so Bob records the next bits of the secret as `01`. Likewise, for the third location, the permuted tile set is {0,3,5,7,6}, the tile in the current location is 5, which is in index 2, and the number of bits to expect is 2, so Bob records the next bits of the secret as `10`. At the fourth location, Bob will find that the permuted tile set is {15,12,9,14,13,10,11}, the tile in the current location is 14, which is in index 3, and the number of bits to expect is 2, and so Bob will record the 7th and 8th bits of the secret as `11`. Bob has 8 bits now and so he can find out how many more bits to expect. The bits Bob has are `00011011`, which is the number 27 and the correct number of bits to expect to extract from the rest of the tiling. Bob will repeat the procedure to extract bits from the tiling until he has extracted the expected number of bits and then will translate the bitstring of data he has extracted to something meaningful. In this case, Bob will find out that the next 27 bits are `101010011000101011000111000`, which translates to 88888888 in decimal.

4.3 Steganalysis

Numerous image steganalysis tools are freely available on the Internet, including StegSecret [66], Virtual Steganographic Laboratory (VSL) [67] and Digital Invisible Ink Toolkit (DIIT) [68]. All three tools implement the RS Steganalysis technique [69],

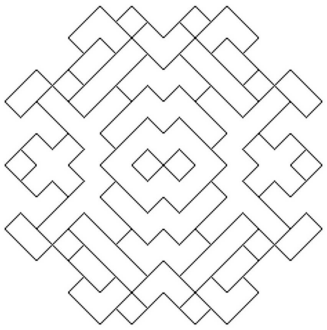


Figure 4.2. Finished line drawing encoding decimal number 88888888.

which classifies images based on a smoothness metric. The tools also implement other methods in addition to RS Steganalyis. StegSecret implements the Chi-Square Attack [70] and a technique the author of the tool created based on the ideas in [70] called Visual Attacks. The VSL tool implements a steganalysis method based on the Binary Similarity Metric [71]. And, the DIIT tool implements Laplace Graph Analysis [56].

However, when these methods are tested against objects generated by a tiling system, their accuracy is no better than guesswork. Clearly, one reason for this is that objects generated by tiling systems are a different type of image than these steganalysis methods are expecting. For this reason, successful detection of the stego-objects generated by a tiling system requires an approach that draws on methods from other domains, such as linguistic steganalysis. Two steganalysis techniques specifically designed to distinguish between clean tilings and steganographic tilings are presented here.

4.3.1 Markov Chain Equivalence Test

The first directed steganalysis method treats each tiled image as a sequence of steps drawn from a third order Markov chain. The neighboring tiles are the previous states and the tile under consideration is the state to which the process moved. The empirical transition matrix is constructed for each tiled image being tested. Using a method adapted from [72], the score for the image is computed as the Manhattan distance of the empirical transition matrix T from the model transition matrix for clean tiled images Tc,

$$s(T) = \sum_{x,y,z,w \in \Sigma} |Tc_{x,y,z,w} - T_{x,y,z,w}|. \tag{4.2}$$

The clean model is built by averaging the transition matrix over 1000 examples of clean tiled images made using a given production rule set. Allowing Wendy to know the rule set that Alice and Bob are using is not required by Kerckhoffs's law [73]

and it does make Wendy's job easier. However, if Wendy watches Alice and Bob's communication long enough, she could learn the production rule set they were using. Thus, we can assume that Wendy knows the rule set from the beginning.

4.3.2 Chi-Square Test

The other directed steganalysis technique counts the occurrences of each tile in the tiled images being tested and compares those values to the model counts for a clean image using a Chi-square test with a Laplace correction. The score for each tiled image T is then

$$s(T) = \sum_{t \in \Sigma} \frac{(O_t - E_t)^2}{E_t + 1}, \tag{4.3}$$

where O_t is the number of t tiles observed in T and E_t is the expected number of t tiles from the model of a clean image Tc.

As before, the clean model T_c is obtained by averaging the tile counts over 1000 examples of clean tiled images. Likewise, Wendy is again assumed to know the production rule set being used.

4.4 Experimental Results

Capacity, robustness and security are three key properties of covert channels. The channel capacity is the number of bits that can be sent per transmission, typically measured in bits per object or bits per symbol. The robustness of the channel is the types and the amounts of noise or tampering that the channel can endure without preventing covert communication. The steganographic security of the channel is a measure of the indistinguishability of the steganographic usages of the channel (the stego-objects) from the non-steganographic usages (the cover objects).

Experimental results on capacity, robustness and security were obtained for an example tiling system using square tiles composed of lines connecting the center of

the tile to zero or more of the corners. Four production rule sets were devised to investigate the effect of context sensitivity (the constraints placed on candidate tile sets by the production rules) on capacity, robustness and security. The example tile set is shown in Figure 4.3 and examples of tiled images generated using each production rules set are shown in Figure 4.4.

Definition 4.4.1 (Experimental Line Drawing Rule Sets) *Let $b_x(\alpha)$ be 1 if the tile α has a line connecting the center of the tile to the corner indexed by x and 0 otherwise. Then, let the notation be abused so that $b_{\{X\}}(\{A\})$ counts the number of lines which connect each of the centers of the specified tiles $\alpha_i \in A$ to the specified corners of those tiles $x_i \in X$ (one corner for each tile), which is computed as follows.*

$$b_{\{X\}}(\{A\})) = \sum_i b_{x_i}(\alpha_i) \ . \tag{4.4}$$

Then, the four rule sets used for this experiment can be written as follows, where corners are numbered from 1 to 4 in clockwise order from the top-left corner and the neighbors are the adjacent tiles to the NW, N, and W of the current location.
Rule set 1:

$$\gamma \in \begin{cases} \{0,3,5,6,7\} & b_{3,4,2}(\alpha_1,\alpha_2,\alpha_3) = 0 \\ \{9:15\} & b_{3,4,2}(\alpha_1,\alpha_2,\alpha_3) = 1 \\ \{0,3,5,6,7,9:15\} & otherwise \end{cases} \tag{4.5}$$

Rule set 2:

$$\gamma \in \begin{cases} \{0:15\} & \alpha_1 = \alpha_2 = \alpha_3 = 0 \\ \{0:7\} & b_{3,4,2}(\alpha_1,\alpha_2,\alpha_3) = 0 \\ \{8:15\} & b_{3,4,2}(\alpha_1,\alpha_2,\alpha_3)) = 1 \\ \{0:15\} & otherwise \end{cases} \tag{4.6}$$

Rule set 3:

$$\gamma \in \{0:15\} \tag{4.7}$$

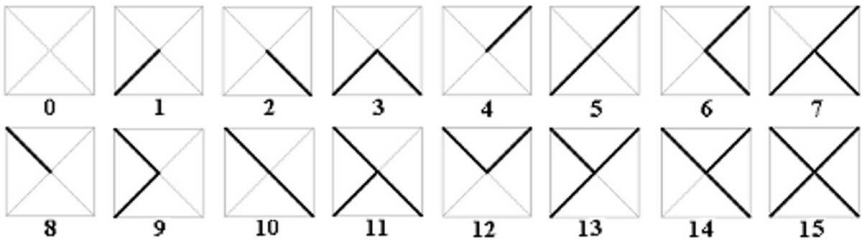
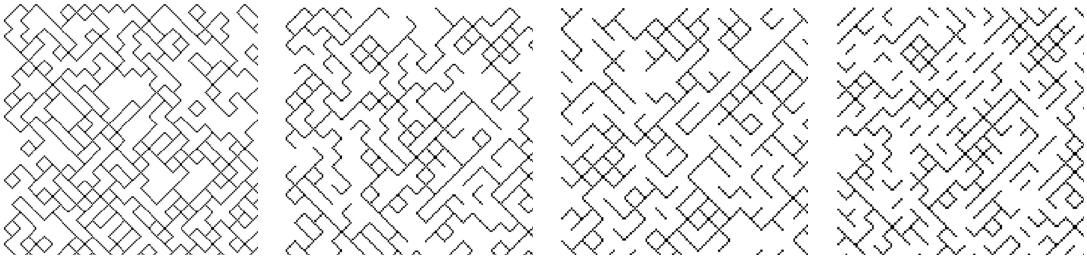


Figure 4.3. Example tile set for $N = M = 4$ as diagonal crosses.



(a) Rule Set 1 (b) Rule Set 2 (c) Rule Set 3 (d) Rule Set 4

Figure 4.4. Examples of line drawings created using each rule set.

Rule set 4:

$$\gamma \in \begin{cases} \{0:15\} & \alpha_1 = \alpha_2 = \alpha_3 = 0 \\ \{8:15\} & b_{3,4,2}(\alpha_1, \alpha_2, \alpha_3) = 0 \\ \{0:7\} & b_{3,4,2}(\alpha_1, \alpha_2, \alpha_3) = 1 \\ \{0:15\} & otherwise \end{cases} \tag{4.8}$$

4.4.1 Capacity

To measure the covert channel capacity of the tiling system, a square tiling of 400 tiles was generated 1000 times for each of the 4 experimental rule sets and 2 different embedding modes using random hidden messages. The two embedding modes are fixed-length embedding (FLE) and variable-length embedding (VLE). In the FLE

mode, each tile choice encodes the same number of bits, equal to the minimum capacity guaranteed by the production rule set, shown in Equation 4.9.

$$N_F = \min_{\{x,y,z\} \in LHS(R)} \{ \lfloor \log_2(|T(\{x,y,z\})|) \rfloor \} , \tag{4.9}$$

$T(\{x,y,z\})$ is the candidate tile set for a grid location with neighbors $\{x,y,z\}$.

In the VLE mode, each tile choice encodes a variable number of hidden message bits as shown in Equation 4.10. In this mode, the embedding capacity of each tile choice is just the logarithm of the size of the candidate tile set for the given location.

$$N_V(r,c) = \lfloor \log_2(|T(\{x,y,z\})|) \rfloor , \tag{4.10}$$

where $\{x,y,z\} = \{G(r-1,c-1), G(r-1,c), G(r,c-1)\}$ are the neighbors of $G(r,c)$ and $T(\{x,y,z\})$ is the candidate tile set for a grid location with neighbors $\{x,y,z\}$.

The 95% confidence intervals about the means for the capacity experiments are reported in Table 4.1. The results show that the VLE mode encodes fractionally more bits per tile than the FLE mode for rule sets 1, 2 and 4, about half a bit per tile. Rule set 3 is the same for both modes because it does not constrain tile choices, and thus the FLE capacity is identical to the VLE capacity in that case. Because rule set 3 puts no constraints on the relationship between neighbors, it yields the highest capacity of the 4 rule sets tested. Rule set 1 is the most constrained rule set and so has the lowest expected capacity. Rule set 2 relaxes the constraints of rule set 1 and, being less constrained than rule set 1 and more constrained than rule set 3, it's expected capacity is between those of rule set 1 and rule set 3. Rule set 4 has nearly identical capacity to that of rule set 2 because the two are constrained by the same amount.

4.4.2 Robustness

The robustness of a stego-object is a measure of the degree of tampering it can tolerate without corruption of the hidden data. Because stego-objects can be subjected

Table 4.1.
Embedding capacity of the tiling systems

	Encoding Method	
Rule Set	FLE (bits/tile)	VLE (bits/tile)
1	2.0000 ± 0.0000	2.5601 ± 0.0014
2	3.0000 ± 0.0000	3.4588 ± 0.0015
3	4.0000 ± 0.0000	4.0000 ± 0.0000
4	3.0000 ± 0.0000	3.4603 ± 0.0015

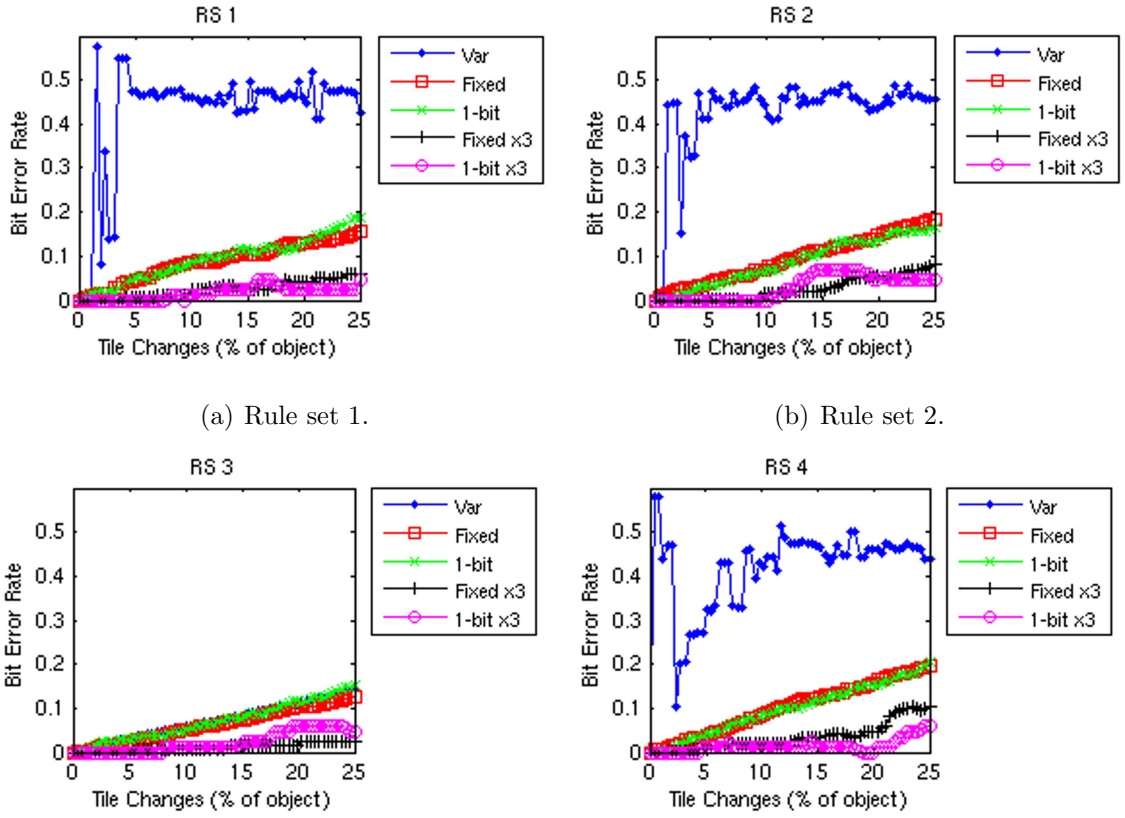
to various forms of tampering, it is important to be clear on precisely which forms of tampering a stego-object can withstand. In general, most tampering processes seek to augment the object with specific kinds of noise.

A popular method of tampering is the addition of white noise to a stego-object. The addition of such noise to an image or audio file can obliterate LSB steganography without noticeable degradation in the quality of the object. If our stego-objects are represented as binary images, we can assume that the addition of white noise is not a viable tampering technique as it would result in clearly noticeable object degradation. On the other hand, if we represent our stego-objects as grayscale images, then the addition of white noise will be noticeable, but will not affect the secret payload.

If we subject ourselves to Kerckhoffs's law so that Wendy knows the stego-system but not the key, it is reasonable to assume that Wendy may add a specific kind of noise to the image by swapping tiles with grammatically valid alternatives. To measure the robustness of objects under this operation, we allow Wendy to change as many tiles as she wants, with the constraint that she can change a specific tile location no more than a fixed number of times in total. Under this condition, we then measure the direct impact on the secret payload in number of errors in payload after each change.

To measure the robustness of the generated stego-objects, we ran one test for each of three embedding modes (variable-, fixed-, and unit-length), and one for both fixed-length and unit-length (i.e., 1 bit/tile) embeddings with redundancy. We can add redundancy by enlarging the secret message through (redundant) repetition k times. In our tests of robustness with redundancy, the payload was an ASCII encoded version of a secret message repeated $k = 3$ times. A simple majority vote is used to decide on the final value of decoded bits.

The results of the robustness experiments are shown in Figure 4.5. In the figure, the x-axis is the number of tile changes relative to the original object and the y-axis is the bit-error rate. It is clear from the figures that fixed length embeddings exhibit a performance that is superior to that of variable length embeddings, in terms of robustness. At a tampering rate of 25%, i.e. 25% of tiles changed by Wendy,



(a) Rule set 1.

(b) Rule set 2.

(c) Rule set 3.

(d) Rule set 4.

Figure 4.5. Robustness of the tiling systems.

variable-length embeddings have a bit-error rate of close to 50%, while both fixed- and unit-length embeddings have error rates less than 20%. We also observe that redundancy increases robustness to tampering: with 25% tiles changed, fixed-length embedding with redundancy achieves a bit-error rate of less than 10% and that of unit-length is less than 5%.

Variable length embeddings perform poorly because tiles that previously encoded n bits under Alice's direction can be subjected to a neighbor change (by Wendy) that causes Bob's decoding algorithm to find the same tile now embedding $m \neq n$ bits, effectively shifting the rest of the message and causing numerous and simultaneous errors. The effect of such tampering could be mitigated by having a human in the decoding loop. In the case of ASCII encoding, for example, the point at which such an error is encountered will be clear because the payload prior to that point is semi-readable (possibly due to Wendy's partially successful tampering), while the subsequent payload is unreadable. It is possible for Bob to fix the message by adding or removing bits, continuing the recovery scheme until the next point is met for which too many or too few bits are decoded. Further, we believe it may be possible to automate this recovery process.

4.4.3 Security

To measure the steganographic security of the tiling system, the output of the system is tested against both directed steganalysis techniques presented above. The steganalysis testing process consists of two steps: a learning step and a testing step. During the learning step, Wendy computes the score of each tiled image in the training set and then finds the decision threshold which yields the equal error rate, the value at which the type I error rate is equal to the type II error rate. Then, this decision threshold is used in the testing step to obtain the values necessary to compute the accuracy, which is then recorded. Wendy is given a training set of 200 images, 100

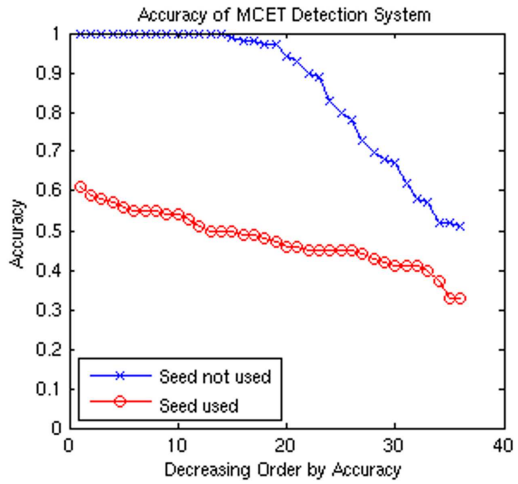
clean and 100 containing hidden information, and 200 test images, again 100 clean and 100 containing hidden information.

4.4.4 Markov Chain Equivalence Test

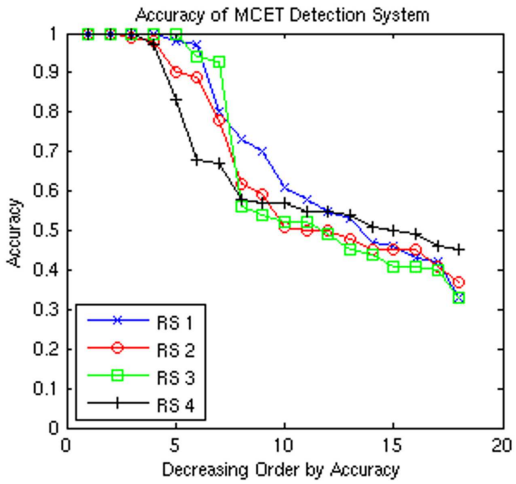
The steganalysis test was executed for all 72 possible combinations of parameters: use of seed, choice of rule set, message format, and embedding format. The results of the tests are shown in Figure 4.6. The data is shown sorted in decreasing order by accuracy, holding one parameter constant. From the results, we can see that if Alice and Bob do not use a seed, Wendy can achieve nearly perfect accuracy of detection for approximately half of the parameter combinations. However, when a seed is used, Wendy does not achieve any significant advantage in detection over pure guesswork. The difference in Wendy's accuracy between when Alice and Bob use a seed and when they do not is statistically significant. There is no significant difference between rule sets, in terms of Wendy's detection accuracy, nor is there a significant difference between message encoding types or between embedding types. Overall, Wendy's detection accuracy with the Markov chain equivalence test method averages 0.67 with a standard deviation of 0.23.

4.4.5 Chi-Square Test

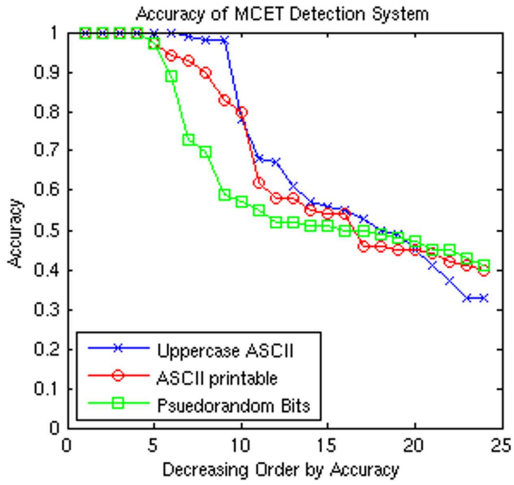
The steganalysis test was again executed for all 72 possible combinations of parameters and the results are shown in Figure 4.7. The data is shown sorted in decreasing order by accuracy, holding one parameter constant. We again see from the results that when Alice and Bob do not use a seed, Wendy achieves perfect accuracy of detection. However, now she can do it for nearly all configurations of the other parameters. When a seed is used, Wendy's accuracy is identical to pure guesswork, with an average accuracy of 0.50. As with the Markov chain equivalence test method, this difference in Wendy's accuracy is statistically significant. We find again that there is no significant difference between rule sets in terms of Wendy's detection accuracy,



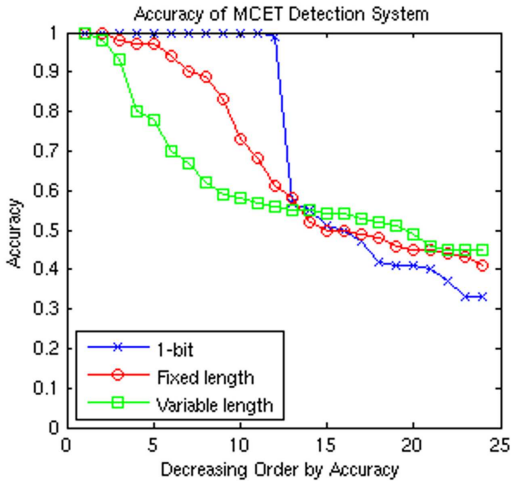
(a) Conditioned on Use of Seed



(b) Conditioned on Rule Set

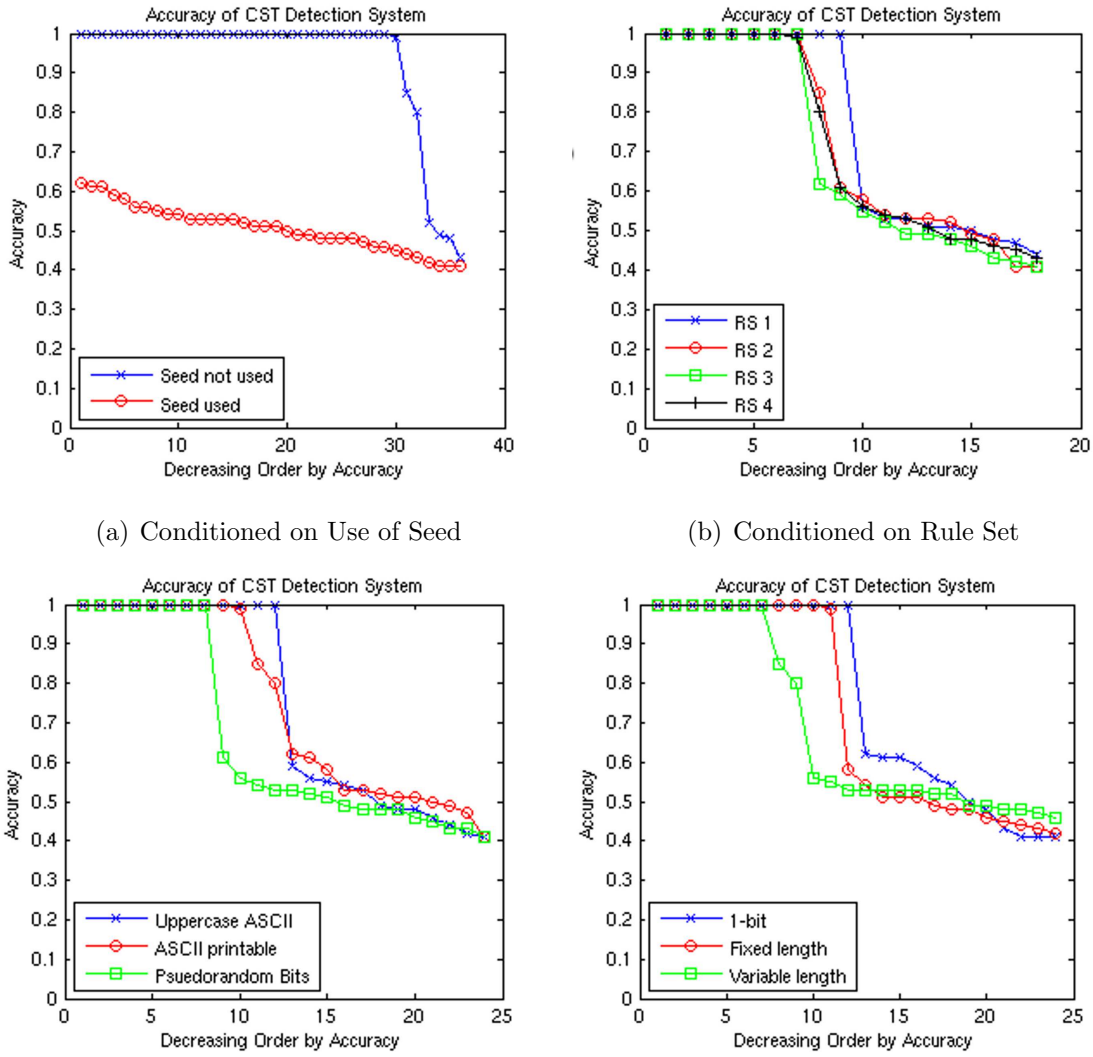


(c) Conditioned on Message Type



(d) Conditioned on Embedding Type

Figure 4.6. Detection accuracy of the Markov Chain Equivalence Test.



(a) Conditioned on Use of Seed

(b) Conditioned on Rule Set

(c) Conditioned on Embedding Type

(d) Conditioned on Embedding Type

Figure 4.7. Detection accuracy results of the Chi-Square Test.

nor is there a significant difference between message encoding types or between embedding types. Overall, Wendy's detection accuracy with the Chi-square test method averages 0.72 with a standard deviation of 0.25.

4.4.6 Analysis of Security Results

Taking into account only the 36 parameter combinations that yield the worst results for Wendy (which is roughly the set for which Alice and Bob are using a seed), her detection accuracy using the Markov chain equivalence test method has a mean of 0.47 with a standard deviation of 0.06. For the Chi-square test method, Wendy's detection accuracy has a mean of 0.49 with a standard deviation of 0.05. Thus, we can conclude that neither method performs any better than random chance when Alice and Bob choose one of the "better" parameter combinations, such as {useSeed = TRUE, RS = 1, ASCII printable character encoding, fixed-length embedding} for which Wendy achieves accuracy of 0.58 for the Markov chain equivalence test method and an accuracy of 0.51 for the Chi-square test method or {useSeed = TRUE, RS = 4, pseudo-random bit string encoding, 1-bit embedding} for which Wendy achieves accuracies of 0.51 and 0.43 using the respective test methods.

From the results obtained for the Markov chain equivalence test method and the Chi-square test method, it is found that the two methods yield quite similar results. The results also indicate that the usage of a seed significantly reduces Wendy's accuracy using these two methods. However, there is no significant difference in detection accuracy between rule sets, message encoding or embedding types.

Alice and Bob can still make Wendy's detection task even more difficult by spreading the message across several images in a series, so that only part of an image encodes information and only some of the images which are sent contain any hidden information at all. This reduction in Alice and Bob's covert channel usage directly translates into an increase in the security of their channel, in terms of Wendy's detection accuracy. Further, the additional images add to the robustness of the channel so that Wendy must do more work (change more tiles) in order to cause enough damage to destroy to hidden information.

4.5 Conclusion

We have presented a stego-system which generates stego-objects using context sensitive tiling. We have analyzed the embedding capacity and the robustness of this system under four example production rule sets and found that variable-length embeddings offer greater capacity, but that fixed-length embeddings are far more robust to tampering. To further increase the robustness to tampering, we found that repeating the message can be very effective. The security of the system was experimentally tested against several existing image steganalysis techniques and it was found that these techniques completely failed at the task of distinguishing steganographic objects from cover objects. To attempt to circumvent this problem, two directed steganalysis techniques meant specifically for tiling systems were developed and investigated. The results show that the steganalysis methods are only accurate when no seed is used. In every other case, the directed steganalysis methods performed similarly to random chance.

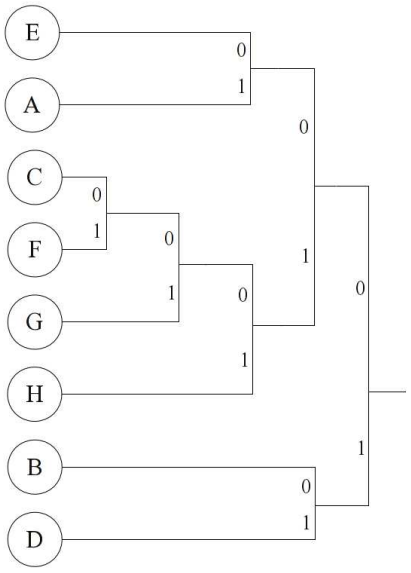
This work has led us to make some useful conclusions. In general, tampering can cause severe complications at the decoding end, but it is possible to enhance robustness through choice of embedding-type and redundancy. Second, in combination with a shared random number generator for symbol rotation, it can be shown that such synthetic steganography is virtually undetectable. Finally, the tiling framework can be generalized to many other settings. The system is not limited to creating images and can be used to create text, audio and physical compositions just as easily. It is also possible to generalize the existing stego-system implementation by including support for user-defined tiling systems.

5 HIDING SECRET MESSAGES IN HUFFMAN TREES

Huffman codes are optimal prefix-free codes for a given distribution and can be constructed according to a simple algorithm developed by Huffman [74]. The codes are optimal in that they produce the shortest expected codeword length. Huffman coding is used as a back-end to multimedia file formats, such as MP3 for audio and JPEG for images, but can also be used on text and for general purpose compression [75]. A Huffman code can be represented graphically by a tree, referred to as a Huffman tree, and the codebook and tree are interchangeable. An example of a Huffman tree and the corresponding codebook is shown in Figure 5.1.

Chen et al. present two methods for embedding hidden messages using modified Huffman trees, in which the message is hidden in a Huffman-encoded object by prefixing [76] or suffixing [77] the the codewords with bits of the secret message. These methods achieve greater capacity than the scheme presented in this chapter, but are easily detectable. In order for a third-party to correctly decode the object, they must possess the Huffman tree. However, examining the Huffman tree will immediately give away that an information hiding scheme is being used. In the first case (prepending the secret message bits to codewords), the left and right subtrees of the root node both contain the same set of symbols. In the second case (appending the secret message bits to codewords), in order for the tree to allow correct decoding, every sibling of a leaf node shares the same symbol as that node, which means that each symbol appears in the tree twice. In both cases, testing whether or not the Huffman tree contains duplicate symbols will reveal whether or not the compressed object could be hiding a secret message. The Huffman trees end up being twice as big as they should be.

The information hiding scheme presented here hides information in the structure of the Huffman tree as opposed to the content. Thus, the tree does not contain



Symbol	Prob.	Code
A	0.1889	001
B	0.2101	10
C	0.0291	01000
D	0.2123	11
E	0.1467	000
F	0.0221	01001
G	0.0643	0101
H	0.1268	011

Figure 5.1. Example of a Huffman tree and the corresponding codebook. The probabilities are shown for demonstration purposes and would not be explicitly included in the codebook or Huffman tree.

duplicate symbols which would immediately betray it's true nature. It will be shown that, under the right circumstances, the stego-tree is indistinguishable from a clean tree for the same content. Since the first priority of a covert channel is to remain hidden, the scheme presented here does not hesitate to trade capacity for increased security.

The remainder of this chapter is organized as follows. Section 5.1 defines the information hiding scheme and presents algorithms for implementing it. Section 5.2 presents analytical results on the capacity and robustness of the scheme as well as experimental results for the security of the scheme and Section 5.3 concludes the chapter. The contents of this chapter were presented and published in the Proceedings of the Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing in 2012 [78].

5.1 Hiding in Huffman Trees

Hiding information in Huffman trees exploits the property of such trees that any other tree that assigns the same codeword lengths to the symbols will also be optimal. Thus, there is redundancy in the structure of a Huffman tree which can be exploited for information hiding. Since the choice of how to order the children of a node is arbitrary, the specific choice made can encode a number of bits. Algorithm 3 presents a modified Huffman tree-building algorithm which allows information to be hidden in the tree's structure through the ordering of the children.

When building the Huffman tree, the algorithm recursively removes and merges the least probable symbols into super-symbols, whose probability is the sum of the symbols which make it up, and then adds the new super-symbol to the list until there is just one super-symbol remaining. The final super-symbol contains all the original symbols and has probability 1. After the tree is built, the children of each internal node of the tree are reordered to embed the secret information according to Algorithm 4. An example of a steganographic Huffman tree and the corresponding codebook is shown in Figure 5.1.

To extract the hidden information embedded in the Huffman tree, start at the root node and visit each internal node in breadth-first order from left to right. At each node which has all D children, convert the index of the permutation of the children to a n-bit binary number, where $n = \lfloor log_2(D!) \rfloor$, and append the bit string to an initially empty output string. For the internal node which has $(D - B)$ children (there is at most one node that does not have D children), use $n = \lfloor log_2(\frac{D!}{B!}) \rfloor$ instead. Once all internal nodes have been visited, the output string contains the bits that were hidden in the structure of the tree. The extraction algorithm is shown in Algorithm 5.

Even for non-steganographic purposes, the Huffman tree or the codebook must be sent before it can be used to decode messages. If a method for encoding and decoding the Huffman tree is publicly available, then that method can be used to encode the Huffmann tree or codebook before sending it. However, if such a method

Algorithm 3: Steganographic Huffman Tree Constructor.

Input: A set of symbols $S = \{s \in A \mid P(s) \geq 0\}$ with associated probabilities $P(s)$, the degree $D \geq 2$ of the tree, a secret message M

Output: Steganographic Huffman tree T

$T = \emptyset$, the empty set;

for $s \in S$ **do**

  $T = T \cup (\{s\}, P(s), \emptyset)$;

Sort T in increasing order by *prob*, then by *symbol*;

$B = (D-2) - ((|S|-2) \mod (D-1))$;

$R = \{T_1, \ldots, T_{(D-B)}\}$;

$T = T \setminus R$;

$t =$ new node;

$t.symbol = R_1.symbol \cup \cdots \cup R_{D-B}.symbol$;

$t.prob = R_1.prob + \cdots + R_{D-B}.prob$;

Append B NULL nodes to the end of R;

$t.children = R$;

Add t to T;

repeat

  Sort T in increasing order by *prob*, then by symbol;

  $R = \{T_1, \ldots, T_D\}$;

  $T = T \setminus R$;

  $T = T \cup (\bigcup_{i=1}^{D} R_i.symbol, \sum_{i=1}^{D} R_i.prob, R)$;

until $|T| = 1$;

Embed M into T using Algorithm 4;

return T;

Algorithm 4: Huffman Tree Information Hiding.

Input: Huffman tree T, a secret message M of sufficient length

Output: Steganographic Huffman tree T

$Q = \emptyset$;

Add T to Q **repeat**

- $t = Q_1$;
- $Q = Q \setminus t$;
- **if** *t is a leaf* **then**
  - Continue
- **if** *t has a NULL child* **then**
  - $n = \lfloor log_2(\frac{D!}{B!}) \rfloor$;
- **else**
  - $n = \lfloor log_2(D!) \rfloor$;
- $m = \{M_i\}_{i=1}^{n}$;
- $M = \{M_i\}_{i=n+1}^{|M|}$;
- Convert the bitstring m to an integer w;
- Sort $t.children$ lexicographically by symbol;
- Permute $t.children$ according to w;
- **for** *each $c \in t.children$* **do**
  - Add c to Q;

until $|Q| == 0$;

return T;

Algorithm 5: Huffman Tree Information Extraction.

Input: A Steganographic Huffman tree T

Output: A secret message M

$D = |T.children|$ is the degree of T;

n_S is the number of symbols at leaf nodes of T;

$B = (D-2) - ((|S|-2) mod (D-1))$;

$M = \epsilon$, the empty string;

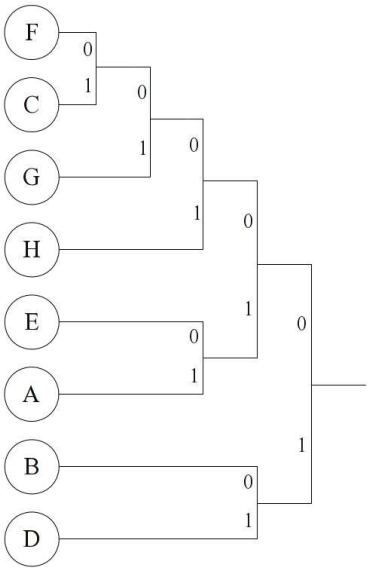
$Q = \emptyset$;

Add T to Q **repeat**

- $t = Q_1$;
- $Q = Q \setminus t$;
- **if** *t is a leaf* **then**
  - Continue;
- **if** *t has a NULL child* **then**
  - $n = \lfloor log_2(\frac{D!}{B!}) \rfloor$;
- **else**
  - $n = \lfloor log_2(D!) \rfloor$;
- Convert the ordering of $t.children$ to an integer w;
- Convert w to a n-bit binary string m;
- Append m to M;
- **for** *each $c \in t.children$* **do**
  - Add c to Q;

until $|Q| == 0$;

return M;



Symbol	Prob.	Code
A	0.1889	011
B	0.2101	10
C	0.0291	00001
D	0.2123	11
E	0.1467	010
F	0.0221	000001
G	0.0643	0001
H	0.1268	001

Figure 5.2. Example of a steganographic Huffman tree and the corresponding codebook for $M = 0100101$.

is not available, one possible method for encoding the tree and one for encoding the codebook are briefly presented here. To encode the tree, start at the root and traverse the tree in breadth-first search order from left to right, writing a '1' if the node has children and a '0' if the node is a leaf. Then, append the plaintext for each symbol in the tree in depth-first search order from left to right. The tree is recovered by first building the tree skeleton (the tree structure without any symbols) and then assigning the symbols to the leafs in depth-first search order. To encode the codebook directly, prefix each codeword with the length of the codeword and then concatenate the (length,codeword) pairs in lexicographical order by symbol. If a symbol is not used, the length of the codeword is zero. To recover the tree, the codewords are used to construct the tree on a pathwise basis and the symbol is assigned to the leaf node which is eventually created by following the path specified by the codeword.

5.2 Capacity, Robustness and Security

Three key properties of covert channels are capacity, robustness and security. The capacity of the channel is the number of bits that can be sent per transmission, usually measured in bits per object or bits per symbol. The robustness of the channel is the amount of noise, or tampering, that the channel can endure without preventing covert communication from occurring. Robustness may also refer to the types of noise and tampering that the channel can resist; for example, some image watermarking techniques are robust against resizing and compression [79] and some linguistic watermarking techniques are robust against summarization and rephrasing [80]. Robustness is only a concern in communication systems in which the adversary is active. An active adversary is one that can intercept and modify communications between Alice and Bob. If the adversary, call her Wendy, is a passive adversary, limited to observation only, there is nothing she can do to prevent the covert communication except to detect it and use that information as evidence against Alice and Bob. The steganographic security of the channel is a measure of the indistinguishability of the steganographic usages of the channel from the non-steganographic usages. Steganographic security differs from cryptographic security in that cryptography is concerned with preventing the content of messages from being read by anyone other than the intended recipient, while steganography is concerned with preventing the existence of the message from being discovered by anyone other than the intended recipient.

5.2.1 Capacity

Theorem 5.2.1 (Capacity of an Ordering of N elements) *Let L be a list of N unique elements. The number of bits of secret information that can be encoded by a permutation of L is given by*

$$n \geq \lfloor log_2(N!) \rfloor \tag{5.1}$$

Proof Enumerate the $N!$ permutations and assign the number 0 to the first permutation, the number 1 to the second, and so on. Partition the permutations into 2 sets

with the first set containing the permutations labeled 0 through $2^{\lfloor log_2(N!)\rfloor} - 1$ and the other set containing the rest. Express each label in the first set in n-bit binary notation, where

$$n = log_2(2^{\lfloor log_2(N!)\rfloor}) = \lfloor log_2(N!)\rfloor. \tag{5.2}$$

Thus, since the permutations in the first set are indexed by all possible n-bit binary numbers, any n-bit secret message can be sent by sending the permutation whose index is the bits of the message. ■

Theorem 5.2.2 (Capacity of a D-ary Huffman Tree) *The number of bits that a D-ary Huffman tree can encode in its structure is given by*

$$n = \lfloor \log_2(\frac{D!}{B!})\rfloor + \lfloor \log_2(D!)\rfloor \frac{m - (D - B)}{(D - 1)} \tag{5.3}$$

where m is the number of symbols, $D \geq 2$ is the degree of the tree and $B = (D-2) - ((m-2) mod (D-1))$ so that $(D - B)$ is the number of symbols merged at the deepest level of the tree.

Proof In the first merge step of the D-ary Huffman algorithm, $(D - B)$ symbols are merged so that D symbols can be merged during every subsequent step. But, these $(D - B)$ symbols can be spread out over D children nodes. The number of distinguishable permutations of $(D - B)$ elements over D spots is given by

$$n = \begin{pmatrix} D \\ D - B \end{pmatrix} \cdot (D - B)! = \frac{D!}{B!} \tag{5.4}$$

so that, by Theorem 5.2.1, the number of bits that can be encoded in the first merge step is

$$\lfloor \log_2(\frac{D!}{B!})\rfloor. \tag{5.5}$$

In every subsequent step, D symbols are merged and

$$\lfloor \log_2(D!)\rfloor \tag{5.6}$$

bits can be encoded in each of those steps. The number of merge steps after the first that are carried out is

$$\frac{m-(D-B)}{(D-1)} \tag{5.7}$$

since $(D-B)$ of the m symbols are merged in the first step and the degree of the tree is D. Thus, the number of bits encoded during the merge steps after the first is

$$\lfloor \log_2(D!) \rfloor \frac{m-(D-B)}{(D-1)}. \tag{5.8}$$

Therefore, the number of bits that a D-ary Huffman tree can encode in it's structure is the sum of Equation 5.5 and Equation 5.8, which is equal to Equation 5.3, . ■

5.2.2 Robustness

The channel is said to be *robust against* that which Wendy can do that does not disrupt the channel or cause information losses. Everything else, i.e. that which Wendy can do to disrupt the channel and cause information loss, is called a *countermeasure*.

Anything that Wendy does to the stego-object that does not change the tree structure or codebook will not affect the hidden message. Thus, the covert channel is robust against modifications to the content of the cover-object, such as addition and deletion. As an extreme example, suppose that Wendy intercepts a Huffman coded text document from Alice to Bob and, supposing that the secret is embedded in the text of the document, decodes the document and replaces the text with entirely new text before recoding the document and sending it on to Bob. Since Wendy did not modify the Huffman tree section of the object, the hidden message, which resides in the structure of the tree, remains intact.

On the other hand, there are a number of countermeasures that Wendy can employ to annihilate the hidden message from the stego-object without degrading the object. If Wendy correctly guesses that the covert channel is contained in the Huffman tree structure, she can change that structure and erase the secret data. For

example, Wendy can convert all Huffman trees and codebooks she observes to canonical Huffman codes, which are a type of Huffman code which allows for the codebook to be concisely encoded as the lengths of the codewords in lexicographical order, and recode the message according to the new codebook. If Wendy guesses correctly that the secret is embedded in the ordering of the children, then she can clear the message from the tree by returning the tree structure to the state in which all children are ordered lexicographically or by probability (which she can estimate from the decoded message body). She could also corrupt the message by randomly permuting the children, or she can change the message by clearing it and embedding her own (perhaps a warning for Alice and Bob to behave). To protect against an active adversary, Alice and Bob should uses digital signatures.

5.2.3 Security

Wendy may or may not know the system that Alice and Bob are using. Indeed, she must assume that they will attempt to use some system and she might as well initially suspect every one of Alice and Bob's messages as being a cover for covert communication. The security that Alice and Bob enjoy against a Wendy who does not know their system is *security through obscurity*. The security that Alice and Bob have against a Wendy who does know their system is the traditional cryptographic notion of security, where the security of the system rests on the strength and secrecy of a key [73].

Only the informed adversary needs to be considered, since security through obscurity is no security at all. Assume that Wendy knows that Alice and Bob are using Huffman trees in their covert communication. To detect the usage of the channel, Wendy must devise a steganalysis system which can determine if a given Huffman tree is hiding information. The most straightforward method of accomplishing this is to hypothesize that clean Huffman trees have a certain property that trees modified to hide information do not have and to decide whether or not a given tree contains

hidden information by testing whether or not the property holds. Three such properties might be (1) that the children of every node are in lexicographical order, (2) that the children of every node are in order by likelihood of symbol and (3) that the tree is in the canonical form. With these three properties, Wendy can construct a decision rule which classifies an object as clean if any of the three properties hold and as a stego-object otherwise. Wendy can use the content of the message to estimate the distribution of the symbol alphabet to use in testing the ordering of nodes by probability.

This decision rule was tested against a set of 1000 stego-objects and 1000 clean objects, using binary trees. The accuracy of the classification was recorded for different alphabet sizes and lengths of content. The stego-objects were generated by picking a random distribution for the symbol alphabet, building the binary Huffman tree given that probability distribution, embedding a random secret message into the tree and then generating content according to the distribution of the alphabet. The results of this security test are shown in Figure 5.3.

Another test was conducted which changed how the objects in the test set were generated to simulate a common method of compressing objects, which is to generate the Huffman tree based on the empirical distribution of the content. For this test, the stego-objects were generated by picking a random distribution for the symbol alphabet, generating content according to the distribution of the alphabet, building the binary Huffman tree given the empirical probability distribution of the generated content, and then embedding a random secret message into the tree. The results of this test are shown in Figure 5.4

Figure 5.3 shows that Wendy's detection accuracy goes to 50%, equivalent to guessing, as the size of the alphabet increases. The peak in accuracy centered around alphabets of size 6 is due to the fact that clean trees are much more likely than dirty trees to be in probabilistic order, which increases Wendy's accuracy in distinguishing between the two as the content length increases so that Wendy has a better estimate of the true symbol distribution. As the number of symbols increases, Wendy's ability

to distinguish between clean objects and stego-objects diminishes because clean trees turn out to be no more likely than stego-trees to be in lexicographical or probabilistic order nor are they more likely to happen to be in canonical form.

In Figure 5.4 it is shown that Wendy's detection accuracy is significantly improved in the case where the Huffman tree is generated based on the empirical distribution of the content. In this case, Wendy's estimate of the symbol distribution is identical to the symbol distribution used to generate the Huffman tree. The increased accuracy of her estimate increases the power of the probabilistic ordering test to distinguish between clean trees and stego-trees. When only one symbol of content is transmitted, Wendy's accuracy is equivalent to guesswork because she does not have a good estimate of the symbol distribution. The slow drop-off of accuracy as the size of the symbol alphabet increases is due to stego-trees which happen to be in order, causing false-negative errors. Wendy's detection accuracy increases for larger alphabet sizes as the length of the content increases because the increased content length gives a more fine-grained symbol distribution, which in turn reduces the likelihood of a stego-tree being mistaken for a clean tree.

A new decision rule is devised which seeks to improve upon the results of the earlier decision rule by testing whether or not the transmitted Huffman tree matches the clean Huffman tree that would have been generated given the symbol distribution. That is, Wendy knows the Huffman tree-building algorithm that Alice and Bob are supposed to be using and she tests for stego by comparing the actual Huffman tree that Alice sends to the Huffman tree that was supposed to have been generated. Wendy finds that any difference between the two is grounds for marking the object as stego. This new decision rule was also tested against a set of 1000 stego-objects and 1000 clean objects. As before, the accuracy of the classification was recorded for different alphabet sizes and lengths of content. Again, both preparations of the test set were used. The results of the security tests are shown in Figures 5.5 and 5.6.

The results shown in Figure 5.5 are very similar to the results shown in Figure 5.3 for the previous decision rule. Wendy's accuracy is a flat 50% for large alphabets and

there is a peak in accuracy for small alphabets. The peak can be attributed to Wendy obtaining an accurate estimate of the symbol distribution for small alphabets and large contents. As the size of the alphabet increases, the content length is insufficient to obtain a good estimate of the symbol distribution. This causes Wendy to classify every object as stego because the tree she builds using the standard algorithm is built using a symbol distribution which is sufficiently different from the true symbol distribution as to cause the trees to differ regardless of being clean or stego. That is, Wendy's error in estimating the symbol distribution shows up as a false-positive error in distinguishing between clean and stego-trees.

In Figure 5.6, however, Wendy's accuracy is perfect, or nearly so, for alphabets larger than 8 symbols. Her accuracy for smaller alphabets is diminished because the small secret messages that those trees can hide occasionally do not require any reordering of the tree, and thus cause Wendy to make a false-negative error. The perfect accuracy demonstrated by this decision rule is attributable to Wendy having access to the initial state of the tree before any embedding takes place because the Huffman tree is built from the empirical symbol distribution of the content to which Wendy has access. If Alice changes the structure of the tree to embed a secret message to Bob, the difference is easily discovered by Wendy and the object is marked as stego with perfect accuracy because only those trees which have been modified will differ from the standard tree that Wendy will build based on the content.

5.3 Conclusions

This chapter has presented a novel information hiding scheme which uses the structure of Huffman trees to encode secret data. The capacity, robustness, and security of the resultant covert channel were analyzed and discussed. It was shown that the capacity of the channel is related to the degree of the tree and the size of the symbol alphabet through Equation 5.3. Thus, the number of bits that a tree

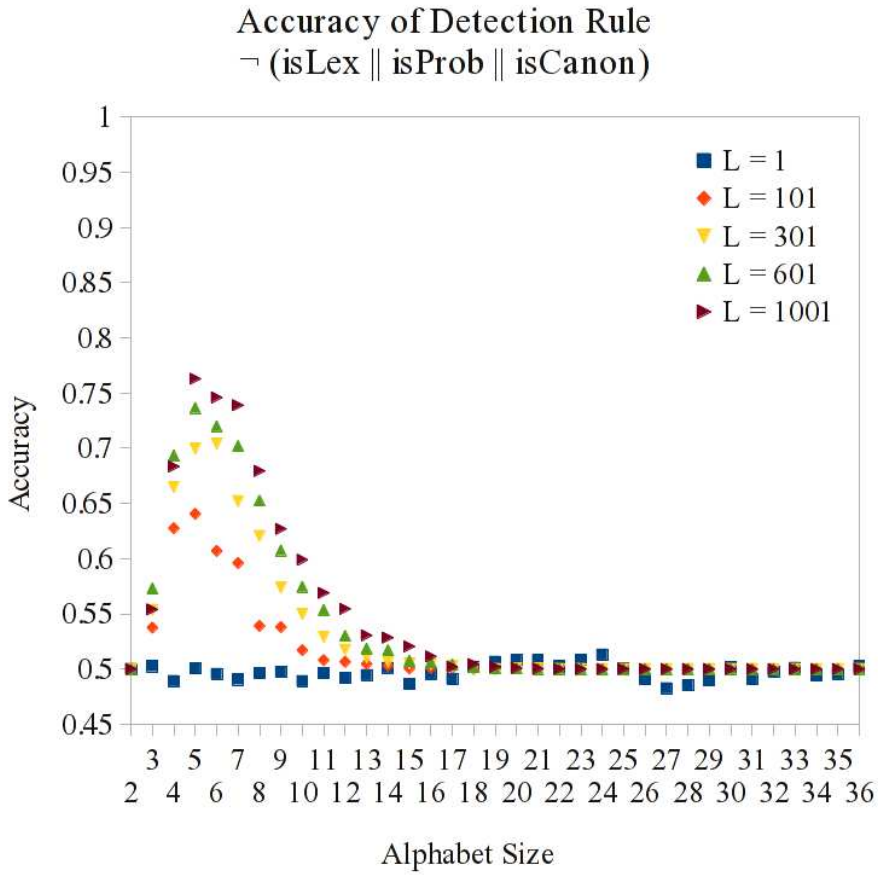


Figure 5.3. The accuracy of the detection rule that a tree which is not in some order is marked as stego. This data is from a simulation which assumes that the distribution that generated the Huffman tree is the same as that from which the content is constructed.

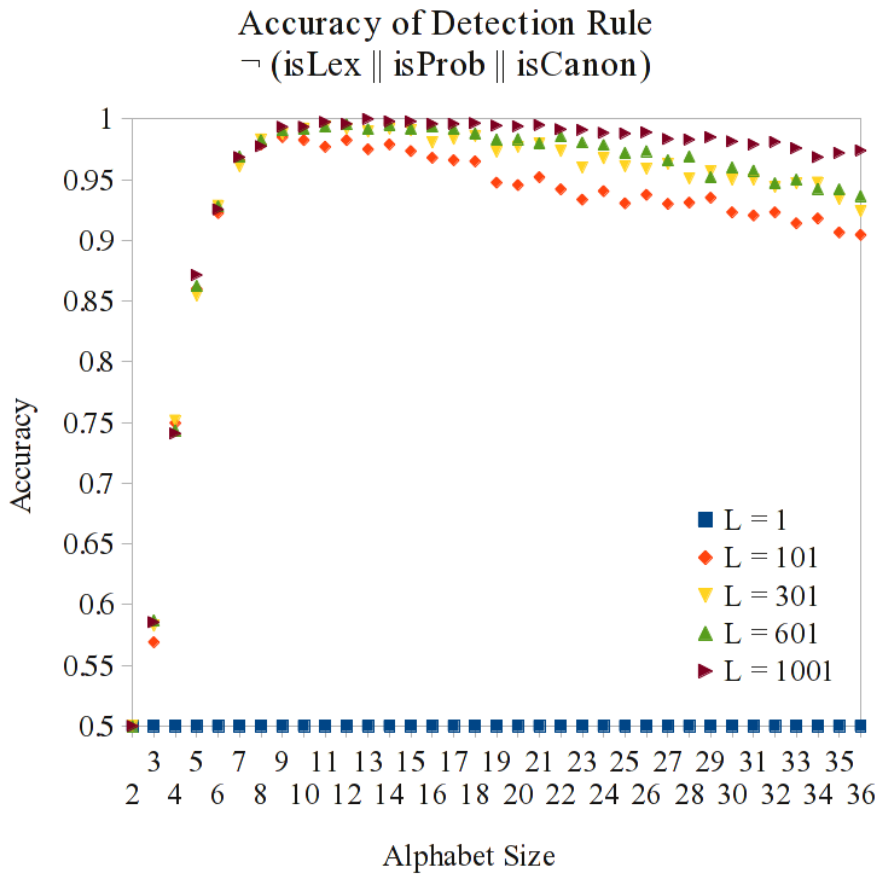


Figure 5.4. The accuracy of the detection rule that a tree which is not in some order is marked as stego. This data is from a simulation which assumes that the Huffman tree is generated using the empirical symbol distribution of the content.

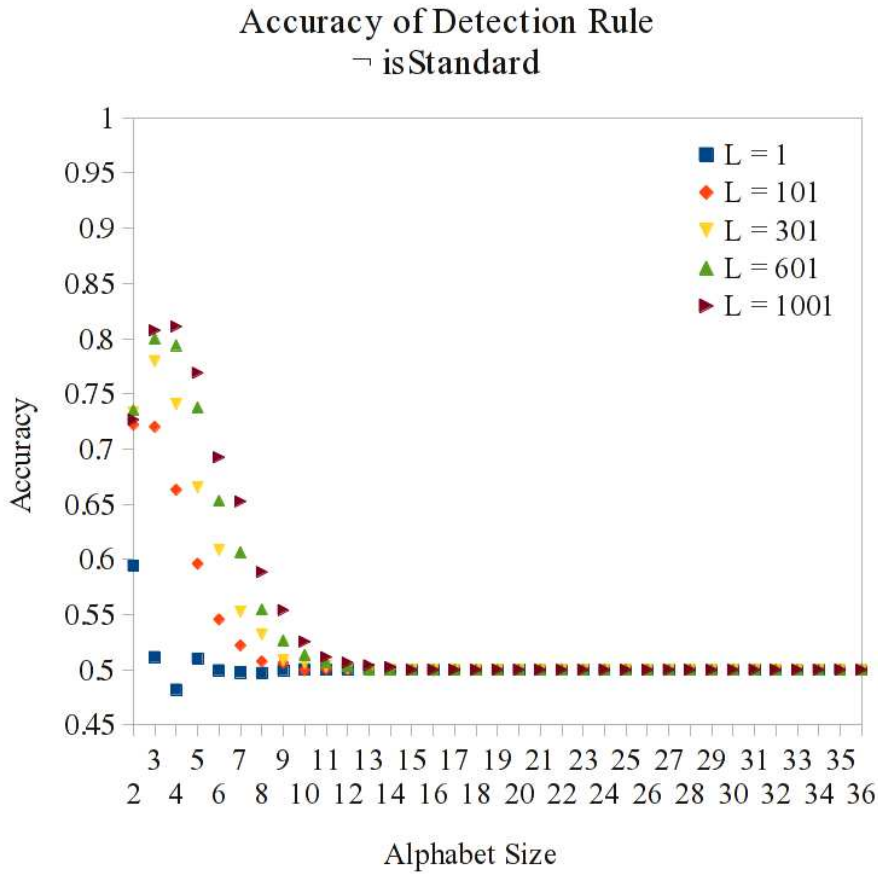


Figure 5.5. The accuracy of the detection rule that a tree which does not match the tree corresponding to the distribution of the content using the standard Huffman algorithm is marked as stego. This data is from a simulation which assumes that the distribution that generated the Huffman tree is the same as that from which the content is constructed.

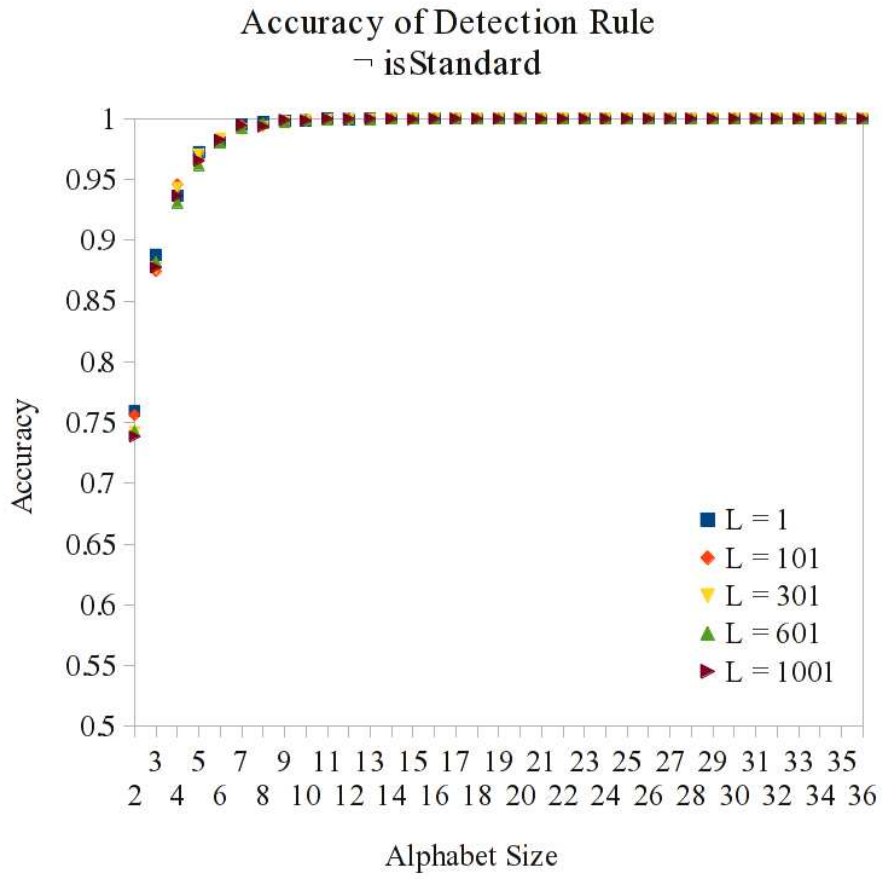


Figure 5.6. The accuracy of the detection rule that a tree which does not match the tree corresponding to the distribution of the content using the standard Huffman algorithm is marked as stego. This data is from a simulation which assumes that the Huffman tree is generated using the empirical symbol distribution of the content.

with degree $D = 2$ can embed is given by $n = m - 1$ bits, where m is the number of symbols in the alphabet.

The robustness analysis argued that the covert channel is robust against any and all modifications to the content of the cover-object. However, an active adversary with knowledge of the stego-system can employ countermeasures that reduce or eliminate the capacity of the channel, such as recoding the object using a different but equivalent tree. Since robustness is only a concern against an active adversary, the usage of a digital signature scheme would suffice to prevent Wendy from tampering with the objects in the channel.

Finally, the security analysis showed that a passive adversary with knowledge of the stego-system can achieve perfect accuracy of detection in the case where the Huffman tree is generated using the empirical symbol distribution of the content. On the other hand, if the tree is built using an *a priori* distribution that Wendy must estimate using the content, her accuracy is no better than pure guesswork. The results obtained suggest that, if the two are willing to put up with slightly sub-optimal compression, then Alice and Bob can trade a lower compression ratio for dramatically increased security, e.g. they can reduce Wendy's accuracy from 100% as shown in Figure 5.6, to 50%, equivalent to pure guesswork as shown in Figure 5.5.

6 STEGODOKU: DATA HIDING IN SUDOKU PUZZLES

Sudoku puzzles are 9×9 grids subdivided into nine 3×3 subgrids. Once solved, each row, column, and subgrid of the board will contain the numbers 1 to 9 exactly once each. When initially presented for solving, a Sudoku puzzle will have several cells already filled in which should provide sufficient information for the player to fill in the remaining cells using logical reasoning. The number of clues given is always at least 17 and usually no more than 32. A valid Sudoku puzzle allows only one solution, thus it is important that the provided clues do not permit more than one solution to the puzzle to be found. Sudoku puzzles have been popular for many years and are commonly found in print magazines and newspapers, as well as online and in apps for smartphones and other devices. An example Sudoku puzzle is shown in Figure 6.1. A StegoDoku puzzle is a steganographic Sudoku puzzle which hides information in the arrangement of numbers on the solved board as well as in the configuration of the clues provided.

	5		7	1		8		
				9				5
	6	4				1		
8				5				
		5	6		9			8
4	2		1		8		7	
		3						6
2	9							
						2	8	

Figure 6.1. This example Sudoku puzzle has 26 clues. When solved, each row, column, and 3x3 subgrid will have the numbers 1 - 9 each exactly once.

StegoDoku is an instance of steganography by cover synthesis. As outlined by Fridrich in [81], steganography by cover synthesis describes information hiding methods which generate novel objects in such a way that the object itself encodes the hidden data. A StegoDoku puzzle can therefore be thought of as an encoding of the secret bitstring. However, unlike a typical cryptographic encoding, which would generate a bitstring which appears to be random, a StegoDoku encoding generates a bitstring which can be interpreted as a valid Sudoku puzzle. Other instances of steganography by cover synthesis include mimic functions [28, 82], such as Spammimic by McKellar (`www.spammimic.com`), and the NICETEXT system by Chapman and Davida [30]. Both Spammimic and NICETEXT convert ciphertext, or any other kind of data, to natural language text in order to hide the fact that cryptography is in use. The text generated by Spammimic attempts to mimic the qualities of spam email. NICETEXT employs the use of style and context templates to generate text which mimics the style of a particular author or genre. StegoDoku can be used for the same purpose, turning ciphertext into something which appears innocuous. In this case, the ciphertext becomes a Sudoku puzzle.

In [36], Shirali-Shareza present a method for hiding approximately 18 bits of data in a single row or column of a Sudoku board. This is accomplished by converting the secret bits to a permutation of the digits 1 through 9. The solution to a previously solved puzzle is then transformed so that the chosen row or column matches the stego-permutation by replacing each instance on the board of the original digit with the new digit from the stego-permutation. After transformation, the numbers in the original clue locations are retained while the rest of the board is erased. The resulting puzzle, with the same number and location of clues as the original puzzle, can be sent via SMS text message to the recipient for decoding, which is accomplished by solving the puzzle and converting whichever row or column contains the data to a bitstring to recover the hidden bits. In order to know which row or column has the hidden data, each cover message must also include a 2-digit "Sudoku board identifier" which encodes the location of the hidden data: $R0$ means that the data is in row R, and $0C$

means the data is in column C. The first or last digit of the identifier must always be zero since the method can only hide data in a single row or column, but not both. This means that only 18 boards can be sent before an identifier has to be reused, and only approximately 22 bits of information can be encoded by each message (18 bits comes from the permutation and 4 bits come from the choice of row or column). We now present a method which aims to utilize the entire capacity of a Sudoku puzzle, and in doing so, eliminates the need for sending auxiliary information with every puzzle while allowing an average of more than 100 bits to be encoded by each puzzle. The contents of this chapter have been submitted for publication in 2015 as an article in Designs, Codes and Cryptography [83].

6.1 Generating and Decoding StegoDoku Boards

The conceptually simplest approach to encoding secret data as a Sudoku board is to map directly from the bits to the board. That is, to enumerate all Sudoku boards and use the secret bits as the index into the list. There are $6670903752021072936960 \approx 2^{72.5}$ valid Sudoku boards [84], of which $5472730538 \approx 2^{32.3496}$ are unique (not mutations of other boards) [85]. Transforming one of the more than 5 billion unique Sudoku boards into one of the more than 6 sextillion valid Sudoku boards can be done efficiently, encoding approximately 40 bits in the specific combination of rotation, mirroring, and permutation used. However, the time required to stream through a list of 5 billion boards , or the memory required to index into such a list , makes this approach impractical.

The above approach is impractical for steganography because it is impractical for board generation in general. Yet, Sudoku puzzles are published constantly in countless media all over the world, implying that Sudoku boards are generated with similar frequency. In fact, generating a Sudoku board is simple. The standard method for generating a Sudoku board is to start with a blank board and repeatedly select an empty cell (at random, or in some predefined order) and set it to a random value until

the board is uniquely solvable. If, by accident, the board ends up being unsolvable, then the previous choice is thrown out and a new choice made. Bad choices are backtracked until, eventually, the board is solvable.

To generate a StegoDoku board, we modify the standard random board generation process so that we can control which numbers are chosen while creating the solved board. Instead of choosing a number randomly, we use a secret bitstring to make the choice. If a cell has n possible numbers from which to choose, or simply *candidates*, we can hide $\lfloor \log_2(n) \rfloor$ bits of information in the choice of which candidate is placed in that cell. For example, if a cell has six candidates $\{2, 4, 5, 6, 7, 9\}$, it can hide two bits. If the next two bits of secret are `10`, then we would set that cell to 5. The choice of 5 for this cell encodes the bitstring `10`. The cells are filled in order of most-constrained to least-constrained, from bottom-right to top-left. The procedure for encoding data as a StegoDoku board is described in Algorithm 6. Once the solved StegoDoku board has been found, it can be passed along to the puzzle-maker which removes clues in order to create the final puzzle. The example Sudoku in Figure 6.1, once solved, is actually a StegoDoku board.

If, while generating the StegoDoku board, the algorithm must backtrack because previous choices caused the board to be unsolvable, some cells will have their capacity reduced and their bitstring-to-candidate mapping changed. The algorithm handles these cases by keeping track of previously set bits and backtracking the embedding appropriately. Backtracking, because it changes the bitstring-to-candidate mapping and reduces the capacity of a cell, will cause errors to appear in the extracted bits during decoding. However, we show in Section 6.3 that using a good Sudoku solver and proceeding in order of decreasing constraint on cells results in very few occurrences of backtracking.

Decoding a StegoDoku board requires first solving the puzzle to obtain the solved board. After the solved board is obtained, the process of generating the StegoDoku board is reenacted. However, instead of looking to a secret bitstring to determine which choice to make, we can look at the board itself to see which choice was actually

made and, from that information, extract the bits which were hidden in that cell by that choice. For example, if we see a cell has six candidates {2,4,5,6,7,9} and that the cell contains the number 5, we know that the bitstring hidden there is `10`. We know this because there are six candidates, meaning there are two hidden bits, and the choice from amongst those candidates was the third candidate, meaning the hidden bits are the third choice from the set {`00`,`01`,`10`,`11`}. So, if a cell had eight candidates and the second candidate was chosen, we know the hidden bits are `001`. Proceeding in the same order as is used for generating the board, all the steps taken to generate the board can be reenacted and all the hidden bits extracted. The procedure for decoding StegoDoku boards is shown in Algorithm 7.

6.1.1 Enhancements to StegoDoku Boards

We now propose and discuss two enhancements to the basic StegoDoku method to increase both the capacity and security of steganographic Sudoku boards. As presented above, the algorithm only makes use of a subset of candidates when hiding bits. For example, a cell which has six candidates can hide two bits, which means four candidates are available for embedding, but the other two will never be chosen. This has implications for the detectability of the stego-boards, but it also means that there is some capacity remaining in the selection which can be squeezed out to increase the number of bits which can be hidden in a StegoDoku board. One way of accessing this extra capacity, inspired by the ideas to use compression for steganography proposed by Anderson in [86] and Sallee in [21], is to assign a code to every candidate, e.g. by using a Huffman code. Then, the next several bits of the secret can be "decoded" into a candidate. For example, with six candidates, the codes could be {`00`,`01`,`100`,`101`,`110`,`111`}. If the next three bits of the message are `111`, the last candidate will be chosen. This allows three bits to be sent in a situation where only two bits would have been sent without using coding. We shall call the

Algorithm 6: CreateStegoDokuBoard.

Input: Sudoku board *board*, bitstring *mssg*

Data: global Sudoku board *thisBoard*

Output: True if a solved board is found

```
if board is solved then
    return True;
cell ← the next cell in board;
c ← list of candidates for cell;
b ← False;
repeat
    if c is empty then
        return False;
    cap ← ⌊log2(c.length)⌋;
    if cap > 0 then
        n ← decimal value of mssg[0..cap − 1];
    else
        n ← 0;
    num ← the n-th element of c, which is then removed;
    boardCopy ← copy of board;
    set cell in boardCopy to num;
    solve boardCopy as much as possible without guessing;
    b ← CreateStegoDoku(boardCopy,mssg[cap..end]);
    if b is True then
        set cell in thisBoard to num;
until b is True;
return True;
```

Algorithm 7: DecodeStegoDokuBoard.

Input: A solved Sudoku board *solvedBoard*

Output: The bitstring extracted from *solvedBoard*

```
board ← a blank Sudoku board;
mssg ← empty string;
while board is not completely solved do
    cell ← the next cell in board;
    c ← list of candidates for cell;
    cap ← ⌊log2(c.length)⌋;
    if cap > 0 then
        num ← value of cell in solvedBoard;
        n ← index of num in c;
        m ← n converted to a cap-bit binary string;
        append m to end of mssg;
    solve board as much as possible without guessing;
return mssg;
```

version of StegoDoku that uses Huffman coding StegoDoku-H, to differentiate it from the basic StegoDoku algorithm.

The basic StegoDoku algorithm will not choose uniformly from all candidates for each cell. Some candidates will never be chosen: the 9 in the first cell, the last three candidates in the third cell, and so on. This not only limits the capacity of the StegoDoku board, but also presents a way for an adversary to distinguish between authentic Sudoku boards and StegoDoku boards. Authentic Sudoku boards will have a 9 in the first cell with probability $\frac{1}{9}$. StegoDoku boards will never have a 9 in the first cell. In the third cell, authentic boards with have one of the last three candidates with probability $\frac{3}{7}$. A StegoDoku board will never have any of the last three candidates in the third cell. Thus, a simple technique for detecting StegoDoku would be as follows: while reenacting the construction of the board, count the number of times that an "out-of-bounds" candidate is chosen. An "out-of-bounds" candidate is one which StegoDoku could not have chosen because its index is equal to or greater than $2^{\lfloor \log_2(n_{candidates}) \rfloor}$, the largest power of two less than the number of candidates. If the number of these out-of-bounds choices is greater than zero, the board could not have been generated by StegoDoku, and is therefore clean. There will be no false negatives. On the other hand, if there are no such choices, the board is much more likely to have been generated by StegoDoku than by an authentic Sudoku board generator. The probability that an authentic board contains zero out-of-bounds choices is $2^{-10.1852 \pm 0.0248}$ with 99.999% confidence, so the probability of a false positive is very near to $\frac{1}{1131}$. Taking the detection accuracy to be the 1 minus the average of type-1 and type-2 error rates, this detection technique has an expected accuracy of over 99.9%.

This simple detection technique can be defeated by permuting the candidate set before choosing which candidate to place in each cell. If Alice and Bob share a secret random number generator seed, then they can synchronize their permutations so that the permutation Alice used for each cell during embedding can be reproduced exactly by Bob during decoding. By using permutations in this way, Alice and Bob

can prevent the adversary from being able to distinguish between StegoDoku boards and authentic Sudoku boards. We shall call the version of StegoDoku that uses permutations StegoDoku-P.

These two enhanced versions of StegoDoku, StegoDoku-H and StegoDoku-P, can be combined to make a StegoDoku generator, which we shall call StegoDoku-HP, that, for each cell, permutes the candidate set before using Huffman coding to decode the secret bits into the choice for that cell. This StegoDoku-HP should have the increased capacity of StegoDoku-H and the increased security of StegoDoku-P.

6.2 Generating and Decoding StegoDoku Puzzles

An additional source of capacity that has not been utilized anywhere else in the literature is the configuration of clues that make up the Sudoku puzzle. In this section we present methods for exploiting the steganographic capacity of the configuration of clues that make up a Sudoku puzzle.

Perhaps the most straightforward method to make a Sudoku puzzle is to start with a solved board and remove clues at random until it is not possible to remove another clue without making the puzzle non-uniquely solvable. This method results in puzzles with an average of 25.38 clues, which is solidly within the standard range of 17 to 32 clues. However, this method is not adaptable to work as a steganographic encoder by simply replacing the random numbers by secret bits. Given a puzzle, Bob cannot determine the order in which Alice removed numbers to make the puzzle. That is, the process is not reversible and so Bob cannot extract bits hidden in this way.

What is needed is for the order in which clues are removed to be made irrelevant. To do this, Alice can map her secret bits to a number and then map that number to a combination of clues. However, there is no guarantee that the set of clues generated will result in a solvable puzzle. Thus, while this method would correct the decoding problem for the clue configuration, it would also make the puzzle potentially non-

uniquely solvable. The additional solutions to the puzzle, potentially very many of them, would result in damage to the bitstring hidden in the board since Bob has no a priori method for distinguishing between the solutions.

The middle ground, where Alice can generate a clue configuration which maintains the correct unique solution to the puzzle and Bob can correctly decode the clue configuration, can be achieved in several ways. One way is for Alice to enumerate valid puzzles, starting with 17-clue puzzles and stopping after 32-clue puzzles, and to each puzzle assign an increasing index, 0 for the first valid puzzle and some n for the last puzzle. Then, to encode her secret bits as a clue configuration she need only convert her secret bits to a number between 0 and n and select the puzzle at the corresponding index. Bob can decode the puzzle by re-enumerating the puzzles, determining the index of the puzzle chosen by Alice and converting that number to its corresponding bitstring (e.g. an $\lfloor \log_2 n \rfloor$-bit string, or the Huffman code for the index). However, this method is very expensive in terms of computation time since there are an exponential number of candidate puzzles that must be checked for validity. To get an idea of how long such an enumeration would take, consider that in January of 2012 it was announced by Gary McGuire that an exhaustive search of all 16-clue Sudoku puzzle had been completed, finding that no 16-clue puzzle was uniquely solvable, therefore proving that 17 clues is the minimum number of clues required to solve a Sudoku board. This exhaustive search took 7.1 million core hours, from January to December of 2011, to complete [85]. To date, only 49151 17-clue puzzles are known, curated by Gordon Royle at The University of Western Australia. But, this number is the total over all boards. The record for a single board is 29 17-clue puzzles. Verifying this count for that one board took nearly a week with McGuire's state-of-the-art software running on a supercomputer.

Since the enumeration approach is so computationally expensive, an alternative technique is required in order for exploitation of the clue configuration to be feasible. We now propose a method for encoding bits in the clue configuration of a Sudoku puzzle which is much faster than the enumeration approach outlined above.

We start by constructing a valid puzzle quickly using a deterministic greedy heuristic approach. We then encode secret bits as a combination of extra clues. Since these clues are not necessary for solving the puzzle, they are redundant information. To offset this addition of unnecessary clues (at least from the perspective of solving the puzzle), we remove as many of the initial clues as possible. Since the information is hidden by the extra clues, removing initial clues does not affect the secret bits. At the receiver's end of the channel, recovering the secret bits from the puzzle starts with solving the puzzle. From a solved board, the same initial clue set can be computed. The extra clues are then identified and decoded to a numerical value representing the combination of extra clues added to the puzzle. This number is then converted to a bitstring, which completes the extraction of the secret from the puzzle. The board can be processed independently to extract the bits hidden in the board itself.

6.2.1 Deterministic Greedy Heuristic Puzzle Making

In making the initial clue set, Alice's goal is to find the smallest set of clues that makes the puzzle solvable. This is because the capacity of the puzzle comes from the choice of combination of clues from those not included in the initial clue set. The more clues there are to choose from, the higher the capacity. Finding an exact minimal initial clue set requires enumeration of hitting sets, which we have ruled out as being too computationally expensive for practical use in a stego-system. The straightforward puzzle-making algorithm described above, where clues are removed until no more can be safely removed, is one method to find an approximate minimal initial clue set efficiently. However, we have found that the following approach results in a smaller initial clue set.

Our method for constructing the initial clue set uses a heuristic greedy approach. Starting from a blank puzzle, each candidate clue receives a score which is computed as the sum of the number of candidates for each cell of the board given that the candidate cell is provided as a clue, as shown Equation 6.1.

$$s_c = \sum_r \sum_c |candidates(r, c)| \tag{6.1}$$

The cell which minimizes the number of candidates remaining on the board is selected as the next clue to add to the puzzle. Ties between candidates which have the same score are broken in a deterministic manner so that the receiver can reconstruct perfectly the exact initial clue set that the sender created. This process is repeated, adding a single clue to the puzzle each time, until the puzzle is uniquely solvable. This method generates puzzles with an average of 23.55 clues.

Once the puzzle is uniquely solvable, the secret bits are added by mapping them to a number and then mapping that number to a combination of the cells which were not selected for the initial clue set. For example, if 24 clues were chosen for the initial clue set, there are 57 clues remaining which can be added to the puzzle. The particular combination of which clues are added will encode the secret bits. Let M be the maximum number of clues that the final puzzle is allowed to have and N be the number of clues in the initial clue set. Then, the total number of combinations from which the extra clue set can be chosen is given by the sum shown in Equation 6.2. The capacity of the choice of which combination of extra clues to provide is then $log_2(T)$.

$$T = \sum_{i=0}^{M-N} \binom{81 - N}{i} \tag{6.2}$$

Mapping a number to a configuration of clues is accomplished by using the set of combinatorial number systems of degree less than or equal to $M - N$. The reason we can uniquely map numbers to combinations, and back again, is that every number can be expressed uniquely as the sum of k combinations [87]. That is, every number n can be written as

$$n = \binom{c_k}{k} + \cdots + \binom{c_2}{2} + \binom{c_1}{1}$$

The numbers less than $\binom{n}{k}$ corresponds to all k-combinations of the numbers $\{0, 1, \ldots n-1\}$. Since the correspondence does not depend on the size of n, the mapping is a bijection from $\mathbb{N}$ to the k-combinations from $\mathbb{N}$ and is therefore invertible.

To map a number to its corresponding k-combination, we use the algorithm shown in Algorithm 8. For each element of the combination, c_k is chosen to be the maximal value such that $\binom{c_k}{k} \leq n$. Then, n is decremented by $\binom{c_k}{k}$, the number of k-combinations of c_k objects, and k is decremented by one. This process is repeated until $k = 0$, at which time the values c_k which have been computed are returned as the desired combination: $\{c_k, c_{k-1}, \ldots, c_1\}$.

Algorithm 8: MapNumberToCombination.

Input: natural numbers n and k

Output: k-combination corresponding to n

```
while k ≥ 1 do
    c_k ← k;
    while (c_k choose k) ≤ n do
        c_k ← c_k + 1;
    c_k ← c_k − 1;
    n ← n − (c_k choose k);
    k ← k − 1;
return {c_k, ..., c_2, c_1};
```

As an example, take the value 12 and convert it to its corresponding 4-combination. To find the correct value of c_4, we look for the largest value such that $\binom{c_4}{4} \leq 12$. From the table below, we can see that this value is $c_4 = 5$.

c_4	4	5	6
$\binom{c_4}{4}$	1	5	15

We decrement n by $\binom{5}{4}$ and k by 1 to get $n = 12 - 5 = 7$ and $k = 4 - 1 = 3$. Then we find the largest c_3 such that $\binom{c_3}{3} \leq 7$. We can see from the table below that this value is $c_3 = 4$.

c_3	3	4	5
$\binom{c_3}{3}$	1	4	10

We decrement n by $\binom{4}{3}$ and k by 1 to get $n = 7 - 4 = 3$ and $k = 3 - 1 = 2$. Then we find the largest c_2 such that $\binom{c_2}{2} \leq 3$. We can see from the table below that this value is $c_2 = 3$.

c_2	2	3	4
$\binom{c_2}{2}$	1	3	6

We decrement n by $\binom{3}{2}$ and k by 1 to get $n = 3 - 3 = 0$ and $k = 2 - 1 = 1$. Since $n = 0$, we know that c_1 must be $c_1 = 1 - 1 = 0$.

Thus, the 4-combination at index 12 is $\{5, 4, 3, 0\}$. Indeed, this can be verified by enumerating the 4-combinations up to index 12, starting at index 0.

The above process applies once we know a specific k to use. The value to use is the largest value of k such that the inequality in Equation 6.3 holds. This allows us to use not only the capacity of the k-combinations, but also that of all combinations of less than k elements.

$$\sum_{i=0}^{k} \binom{81 - N}{i} \leq n \tag{6.3}$$

Mapping from a particular k-combination back to the corresponding number n is even simpler than mapping from n to the corresponding k-combination. Given a particular k-combination c, we can recompute n by using Equation 6.4. The final value for n is just the index of the k-combination as computed directly from the elements chosen plus the offset provided by all the combinations of less than k elements.

$$n = \sum_{i=1}^{k} \binom{c_i}{i} + \sum_{i=0}^{k-1} \binom{81 - N_c}{i} \tag{6.4}$$

6.3 Analysis and Discussion

In this section, we conduct experiments to analyze and discuss the proposed StegoDoku system in terms of the decoding error caused by backtracking, embedding capacity, and security against a passive adversary. Two-tailed p-values are reported for comparisons between results obtained from different algorithms and parameters. A p-value of less than 0.005 is required in order for the difference to be considered statistically significant. The p-value indicates the probability, under the null hypothesis that the means are equal, of observing a larger difference than the one actually observed.

6.3.1 Backtracking Error

To determine which ordering cells results in the fewest occurrences of backtracking, we present the results of an experiment which measured the number of times the generation process backtracked when using each of several possible cell orderings. For each cell ordering scheme, a sample set of 10000 boards was generated and the total number of times the generator backtracked for each board was recorded. The observed means and standard deviations of the backtracking count for each scheme are shown in Table 6.1. The schemes with an asterisk ($*$) denote that at least one of the trials for that scheme had a board which backtracked more than 100 times, an upper limit set so that trials would terminate after not too long. The difference in the resulting amount of backtracking between "first", meaning the first satisfying cell found in top-left to bottom-right order, and "last", meaning the last satisfying cell found in top-left to bottom-right order (equivalently: the first satisfying cell found in bottom-right to top-left order), is statistically significant only in the case where the amount of constraint is ignored. The difference between "inner", meaning the

Table 6.1.
Number of backtracks during generation ($N = 10000$).

Ordering	μ	σ	p-value
First (Top-Left to Bottom-Right)	0.7563	2.5168	7.4896×10^{-12}
Last (Bottom-Right to Top-Left)	0.5562	1.4803	
First Most Constrained	0.0055	0.0740	0.7772
Last Most Constrained	0.0058	0.0759	
First Least Constrained∗	49.6141	48.1968	0.2124
Last Least Constrained∗	50.4647	48.2672	
Inner (Outward Spiral from Center)∗	7.1325	22.8575	≈ 0
Outer (Inward Spiral from Bottom-Right)	0.0842	0.3486	
Inner Most Constrained	0.0142	0.1470	≈ 0
Outer Most Constrained	0.0373	0.2152	
Inner Least Constrained∗	46.9062	47.9748	0.2496
Outer Least Constrained∗	46.1262	47.8396	
Random∗	12.9683	31.6098	–

first satisfying cell found in an anticlockwise spiral originating in the center of the board, and "outer", meaning the last satisfying cell found in an anticlockwise spiral originating in the center of the board (equivalently: the first satisfying cell found in a clockwise spiral originating in the bottom-right corner), is statistically significant except in the case of least constrained. It is clear from the table that proceeding in order of decreasing constraint results in the least amount of backtracking, the best results being obtained by proceeding in order of last most constrained.

The motivation behind minimizing the amount of backtracking is to minimize the decoding error caused by backtracking. To determine the impact of backtracking on decoding accuracy, a sample set of 10000 StegoDoku boards was generated and the decoding error measured for each board. The decoding error is computed as the

Table 6.2.
Backtracking error ($N = 10000$).

Algorithm	n	μ_n (%)	μ_{10000} (%)	σ_{10000} (%)
Random	39	–	–	–
StegoDoku	42	6.2398	0.0262	0.4376
StegoDoku-H	44	5.8010	0.0255	0.4118
StegoDoku-P	43	6.5435	0.0281	0.4736
StegoDoku-HP	39	6.3569	0.0245	0.4477

Hamming distance between the original secret bitstring and the extracted bitstring, trimming the longer of the two so that both have equal length. The results of this experiment are shown in Table 6.2. The "n" column shows how many boards, out of the 10000, had an error during decoding (for "Random", the authentic Sudoku generator, this is just the number of boards which required backtracking). The "μ_n" column shows the mean decoding error for the n boards which had errors during decoding, as a percentage of the length of the bitstring. The "μ_{10000}" column shows the mean decoding error, as a percentage of the length of the bitstring, over all 10000 boards, and the "σ_{10000}" column shows the standard deviation of the decoding error. Because these error levels correspond to less than a single incorrect bit out of 3559 bits, we claim that these levels of error are all acceptable. There is no statistically significant difference between any of the error rates ($p > 0.5806$).

6.3.2 Puzzle Generation

Since the StegoDoku system proposes to hide secret data in Sudoku puzzles, it is reasonable to find out how long it takes StegoDoku to generate a puzzle which encodes some arbitrary secret bitstring. Making a StegoDoku is a 2-step process: generate a board, then generate a clue configuration. The total time required to

Table 6.3.
Sudoku board generation time ($N = 10000$).

Algorithm	μ (ms)	σ (ms)
Random	8.8378	0.9484
StegoDoku	8.9346	0.7554
StegoDoku-H	8.8440	0.8442
StegoDoku-P	8.7438	0.7490
StegoDoku-HP	9.0560	0.6718

make a StegoDoku puzzle is therefore equal to the time required to generate the board plus the time required to generate the clue configuration.

To determine the time required to generate a board, 10000 boards were generated by each algorithm using random secret bits and the elapsed time to generate each board was recorded. The results are shown in Table 6.3. The difference in generation time between the authentic generator and all four StegoDoku algorithms is not statistically significant ($p \approx 0.43$).

To determine the time required to generate a clue configuration, 10000 random boards were generated and the elapsed time to generate a clue configuration, which encoded some random secret bits, was recorded. Each generated clue configuration may have between 17 and 32 clues, and so, in addition to data about the time required to generate clue configurations, data about the final number of clues provided was also collected. The results are shown in Figure 6.2, where the mean observed puzzle generation time has been plotted as a function of the number of clues provided in the final puzzle. Puzzles that end up with more than 25 clues are generated quickly. Puzzles that end up with fewer clues, on the other hand, take much more time to generate because the puzzle maker must search a very large space of sets of clues that can be removed to find the largest set. A 99.999% confidence interval for the mean puzzle generation time overall is 30.7368 ± 0.065 seconds.

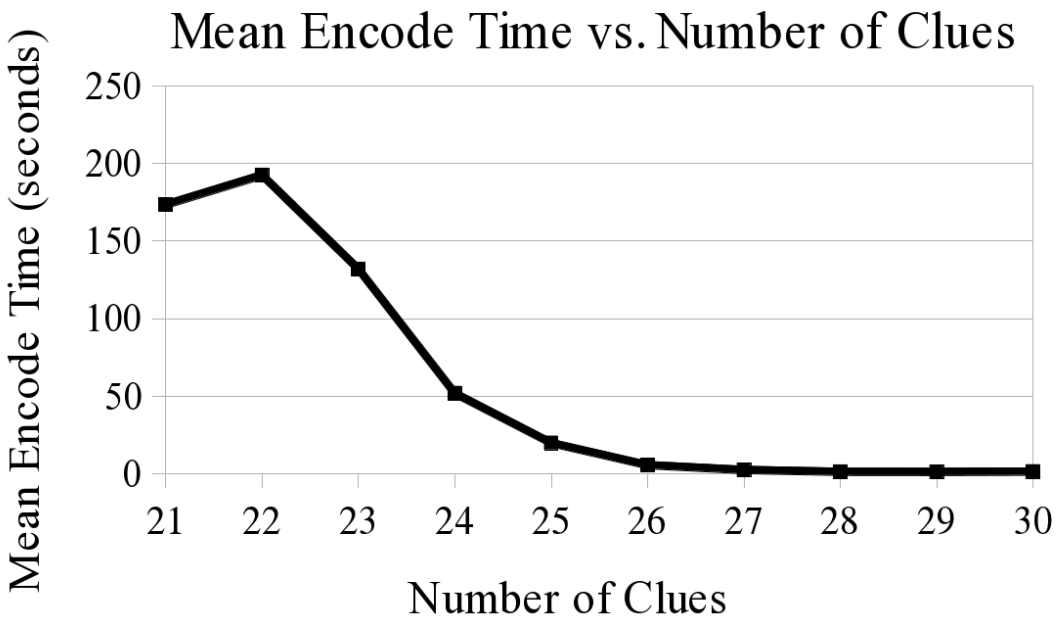


Figure 6.2. The mean observed puzzle generation time as a function of the number of clues provided in the final puzzle. No puzzles with fewer than 21 clues were found.

6.3.3 Capacity

Since boards and puzzles are treated independently in StegoDoku, the total capacity of StegoDoku is the sum of the capacity of the board and the capacity of the puzzle. There are $6670903752021072936960 \approx 6.6710 \times 10^{21}$ valid Sudoku boards [84]. Assuming all boards are equally likely, the theoretical expected embedding capacity for a Sudoku board is therefore $\log_2(6670903752021072936960) \approx 72.5$ bits. To estimate the actual embedding capacity of StegoDoku boards, 10000 boards were generated, using random bitstrings as the secret data, and the number of bits hidden in each board was recorded. Generating 10^4 boards, while only a small fraction of the more than 10^21 possible boards, is nonetheless sufficient to provide an extremely tight confidence interval for the embedding capacity. Figure 6.3 shows the distribution of the observed embedding capacities. It can be seen in the figure that using permutations has a negligible effect on embedding capacity, but using Huffman coding results in a significant increase in embedding capacity. The mean and standard deviation of the embedding capacity for each algorithm is reported in Table 6.4. There is no statistically significant difference between the two versions which use permutations and the two which do not. However, the difference when using Huffman coding, an increase of 8.722 and 8.69 bits for StegoDoku and StegoDoku-P, respectively, is statistically significant ($p \approx 0$). StegoDoku and StegoDoku-P both have an average embedding capacity of slightly more than 61 bits, which is approximately 85.6% of the theoretical expected embedding capacity. StegoDoku-H and StegoDoku-HP both have an average embedding capacity of just over 70 bits, which is approximately 96.7% of the theoretical expected embedding capacity.

To estimate the actual embedding capacity of StegoDoku clue configurations, 10000 clue configurations were generated for random Sudoku boards, using random bitstrings as the secret data, and the number of bits hidden in each clue configuration was recorded. The observed mean and standard deviation of the embedding capacity of the clue configurations generated by the puzzle maker are shown in Table

Table 6.4.
StegoDoku embedding capacity ($N = 10000$).

Algorithm	μ (bits)	σ (bits)	99.999% CI
StegoDoku	61.3034	1.6856	$\mu \pm 0.0745$
StegoDoku-H	70.0254	2.7015	$\mu \pm 0.1193$
StegoDoku-P	61.4029	1.6774	$\mu \pm 0.0741$
StegoDoku-HP	70.0929	2.6539	$\mu \pm 0.1172$
PuzzleMaker	32.4327	3.0920	$\mu \pm 0.1366$

6.4. From these results, we can see that the expected capacity of a StegoDoku puzzle (board plus clue configuration) will be approximately 92.7 bits, and for StegoDoku-H the expected capacity will be approximately 102.5 bits. Permutations do not effect the capacity so StegoDoku-P and StegoDoku-HP will have the same total capacities as StegoDoku and StegoDoku-H, respectively. To verify these expectations, 10000 StegoDoku-H boards were generated and a clue configuration was generated for each. The observed total capacacity of the puzzle was recorded. The observed distribution of total embedding capacities is shown in Figure 6.4.

6.3.4 Security

It must be assumed, by Kerckhoffs's principle, that the adversary knows the system. Therefore, hiding plaintext with a known structure will be detectable since the adversary can decode each Sudoku board and test whether the result contains data with the known structure. Authentic Sudoku boards, which are not hiding data, should produce random bits, which can be distinguished from plaintext data.

An experiment was conducted to test whether Sudoku boards which are hiding structured data can be distinguished from clean Sudoku boards by looking at the decoded message. A test set was generated containing 500 random clean Sudoku boards and 500 random stego-boards, each hiding a random printable ASCII encoded

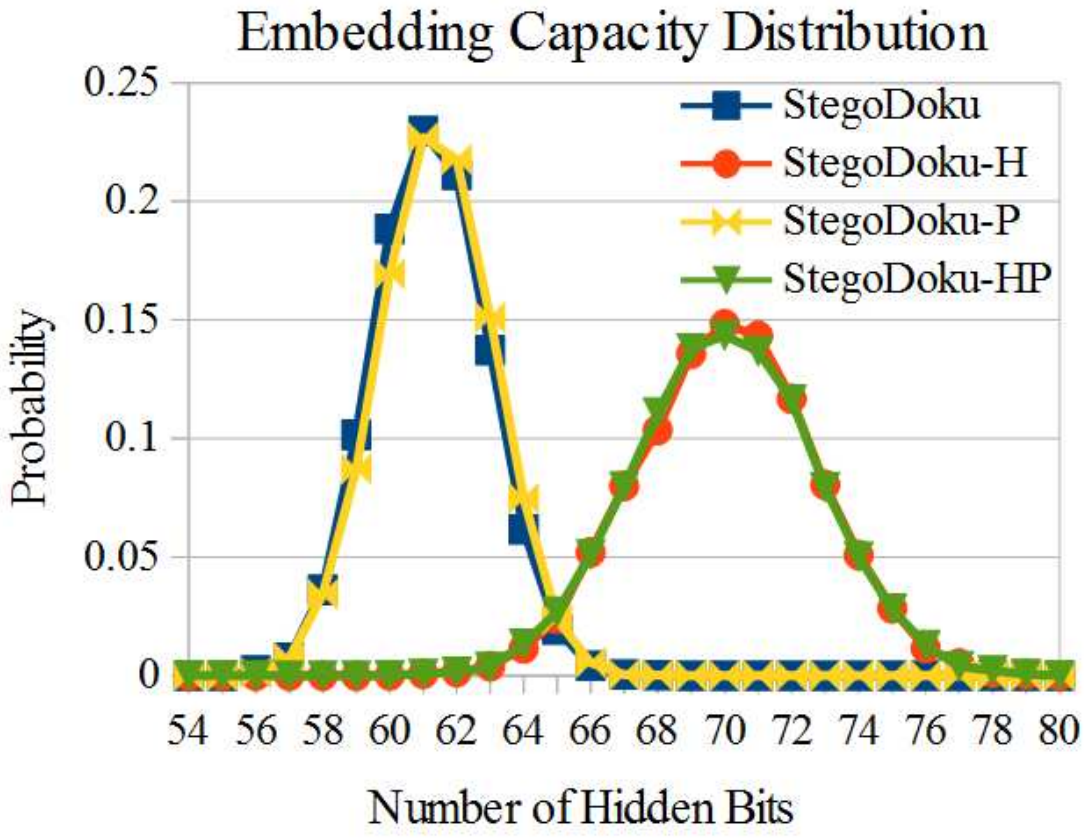


Figure 6.3. The distribution of embedding capacities for the four proposed StegoDoku board generators. The effect on embedding capacity of using permutations during embedding is negligible. The effect of using Huffman coding with StegoDoku and StegoDoku-P, however, is a statistically significant ($p \approx 0$) increase of 8.722 and 8.69 bits, respectively.

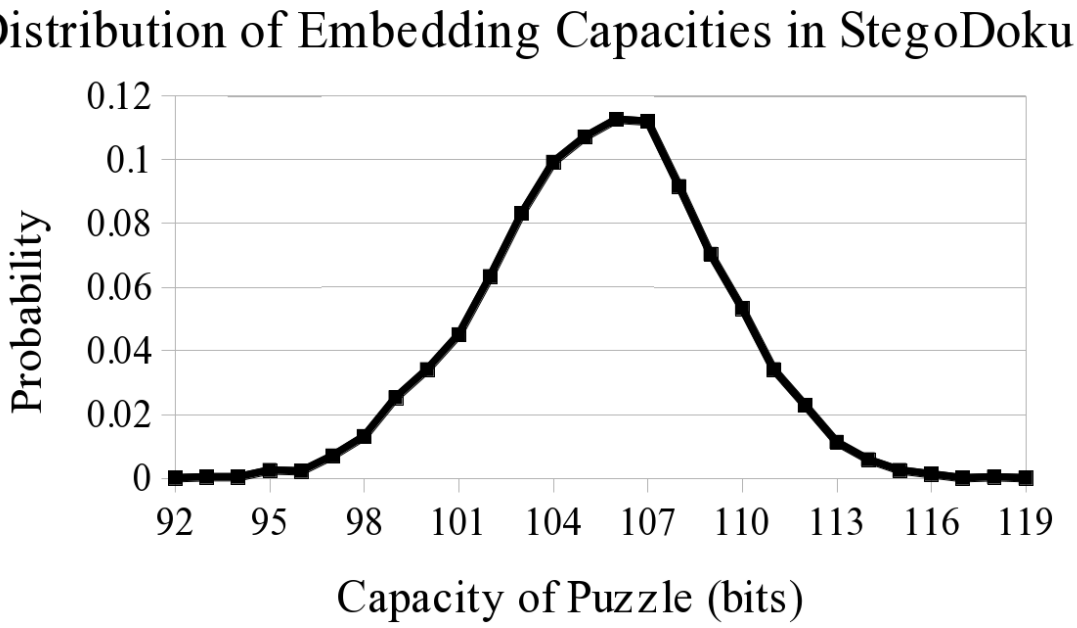


Figure 6.4. The distribution of embedding capacities for the StegoDoku puzzle maker, using the StegoDoku-H board generator. A 99.999% confidence interval for the mean embedding capacity is 105.5425 ± 0.0016 bits. The minimum observed capacity was 93 bits, the maximum was 118 bits.

message. Then, each board was passed through the decoder and the result analyzed to determine if all the output bytes fell into the range of printable ASCII characters, between 32 and 126, inclusive. If more than some threshold number were found to be outside that range, the board was classified as clean, otherwise it was classified as StegoDoku. For StegoDoku and StegoDoku-H, the results show that the detector, using a threshold of zero, can distinguish between authentic Sudoku boards and stego-boards with 100% accuracy, where accuracy is computed as 1 minus the average of type-1 and type-2 error rates.

Encrypting the secret bits, or otherwise obfuscating them, before hiding should preclude this attack since ciphertext should generally be indistinguishable from random bits. We demonstrate this by conducting the above experiment using StegoDoku-P and StegoDoku-HP, both of which use permutations to shuffle the candidate sets during embedding. The permutations act as a cipher and scramble the data as it is being hidden. The results of the experiment show that the detector has 50% accuracy against both StegoDoku-P and StegoDoku-HP. The decoder requires a secret key, which is used as the seed for the random number generator which controls the permutations. Without this secret key, the decoder can only extract garbage. Thus, this detector is unable to distinguish between a StegoDoku-P or StegoDoku-HP board and an authentic Sudoku board. Similarly, when random bits, representing ciphertext, are hidden in a Sudoku board using StegoDoku or StegoDoku-H, the detector can only achieve 50% accuracy using the test for structured data. This is because random bits are very unlikely to appear to be structured data, so the detector will classify all boards as clean when only 50% of them are actually clean, resulting in 50% accuracy.

As mentioned previously in the motivation for enhancements to StegoDoku, the basic StegoDoku algorithm cannot use all candidates to hide data. By observing the choices made during the generation process, a detector could determine if the source is likely to be authentic (truly random boards) or steganographic (the StegoDoku generator). Because both Huffman coding and permutations make it so that any

candidate is able to be selected, this approach will not work when Huffman coding or permutations are being used. An experiment was conducted to test whether StegoDoku boards which are hiding random bits, as an analogue for ciphertext, can be distinguished from clean Sudoku boards by looking at the choices made when making the board. If the detector finds that no out-of-bounds candidates were selected for the board, then it will be classified as dirty. A board with at least one out-of-bounds candidate will be classified as clean. The test set consisted of 500 clean boards and 500 stego-boards. The detector was 99.9% accurate at distinguishing between authentic boards and dirty boards generated by StegoDoku. However, because Huffman coding and permutations both allow all candidates to be potentially chosen, the detector's accuracy on StegoDoku-H, StegoDoku-P, and StegoDoku-HP was 50%, or pure guesswork.

While Huffman coding does allow for all candidates to be potentially chosen, thus defeating the out-of-bounds candidate-counter detector, it does so with predictable probabilities for each candidate. For example, in a cell with nine candidates, such as the first cell, all nine candidates are possible, but, due to the Huffman coding, the 1 and the 2 will only be chosen with probability 1/16. The other seven candidates will each be chosen with probability 1/8. On the other hand, in an authentic Sudoku board, a cell with all nine candidates will have each candidate in equal proportion, each will be chosen with probability 1/9. A similar analysis for cells with other numbers of candidates shows that Huffman coding results in some candidates being chosen less, or more, often in a stego-board than they would be in an authentic board. By comparing the likelihood that a given board was generated by the authentic generator to the likelihood that the same board was generated by the StegoDoku-H generator, a detector can determine whether the board is more or less likely to be authentic, and therefore clean, than to have been generated by StegoDoku-H, and therefore dirty. The likelihood $P(board \mid A)$ that an observed board was generated by the authentic generator A is computed as

$$P(board \mid A) = \prod_{c \in board} \frac{1}{|C(c)|} \tag{6.5}$$

where $C(c)$ is the set of candidates for cell c at the time when that cell was set. Likewise, the likelihood $P(Board \mid S)$ of the observed board having been generated by the StegoDoku-H algorithm is:

$$P(board \mid S) = \prod_{c \in board} \frac{1}{2^{l_c}} \tag{6.6}$$

where l_c is the length of the Huffman code for the value chosen at cell c, given the candidate set $C(c)$. Taking the ratio of the log-likelihoods yields

$$\rho = \frac{\log(P(board \mid S))}{\log(P(board \mid A))} = \frac{\sum_{c \in board} l}{\sum_{c \in board} \log\left(|C(c)|\right)} \tag{6.7}$$

Because $P(board \mid S)$ and $P(board \mid A)$ are always less than 1, when $\rho < 1$, $P(board \mid A) < P(board \mid S)$ and thus it is more likely that the board was generated by the StegoDoku-H generator. If $\rho > 1$, $P(board \mid A) > P(board \mid S)$ and thus the board is more likely to have been generated by the authentic generator. When $\rho = 1$, the two generators are equally likely, but the detector will classify the board as dirty, opting to err on the side of over-detection.

A test set of 500 authentic Sudoku boards and 500 stego-boards was generated and used to measure the detection accuracy of the detector. For each board, if $\rho \leq 1$, the board is classified as dirty, otherwise the board is classified as clean. This detector achieves accuracies of 39.3% on StegoDoku, 78% on StegoDoku-H, 51.5% on StegoDoku-P, and 52.5% on StegoDoku-HP. The accuracy against basic StegoDoku is due to the fact the basic StegoDoku does not use Huffman coding, and so the likelihood $P(Board \mid S)$ is low, causing an increase of false negatives and a decrease of true positives (the false positives and true negatives are unchanged). If the adversary knows for a fact that Alice and Bob are using plain StegoDoku, she

can flip her decision to obtain an accuracy of 60.7%. Stego-boards generated using permutations remain indistinguishable from authentic boards.

By increasing the number of boards seen by the detector before making a decision, the detection accuracy of the detector is improved for both StegoDoku and StegoDoku-H. The results of this experiment are shown in Figure 6.5. As the number of boards increases, the accuracy against StegoDoku improves to 50%, while the accuracy against StegoDoku-H improves to greater than 99%. However, stego-boards generated using permutations still remain indistinguishable from authentic boards regardless of the number of boards observed.

If the secret data is too short to generate an entire Sudoku board, then the StegoDoku algorithms will pad the secret with 0 bits until it is long enough. For short secrets, less than 50-60 bits or approximately 15% of board capacity, the message extracted from the board will have several padding bits. These bits can be used to distinguish between stego-boards and authentic Sudoku boards, which will very likely not have several identical bits at the end of the garbage extracted from them. By looking at the last ten bits of the extracted message and classifying the board as dirty if they are all the same and clean otherwise, a detection accuracy of more than 99% is achieved against both StegoDoku and StegoDoku-H. However, when the hidden data is long enough to take up the full capacity of the board, this method achieves an accuracy of approximately 50% against all four StegoDoku variants. Padding with random bits will also decrease this method's detection accuracy to 50%.

The detection accuracy results for all the board-based detectors described in this section are shown in Table 6.5. From the table, it can be seen clearly that StegoDoku and StegoDoku-H are distinguishable from authentic Sudoku boards with very high accuracy. It can also be seen that StegoDoku-P and StegoDoku-HP, because they use a secret key, are indistinguishable from authentic Sudoku boards. If the key used for doing the permutations and padding is known by the adversary, then she can distinguish authentic boards from boards generated by both StegoDoku-P and StegoDoku-HP with that same accuracy as if permutations had not been used.

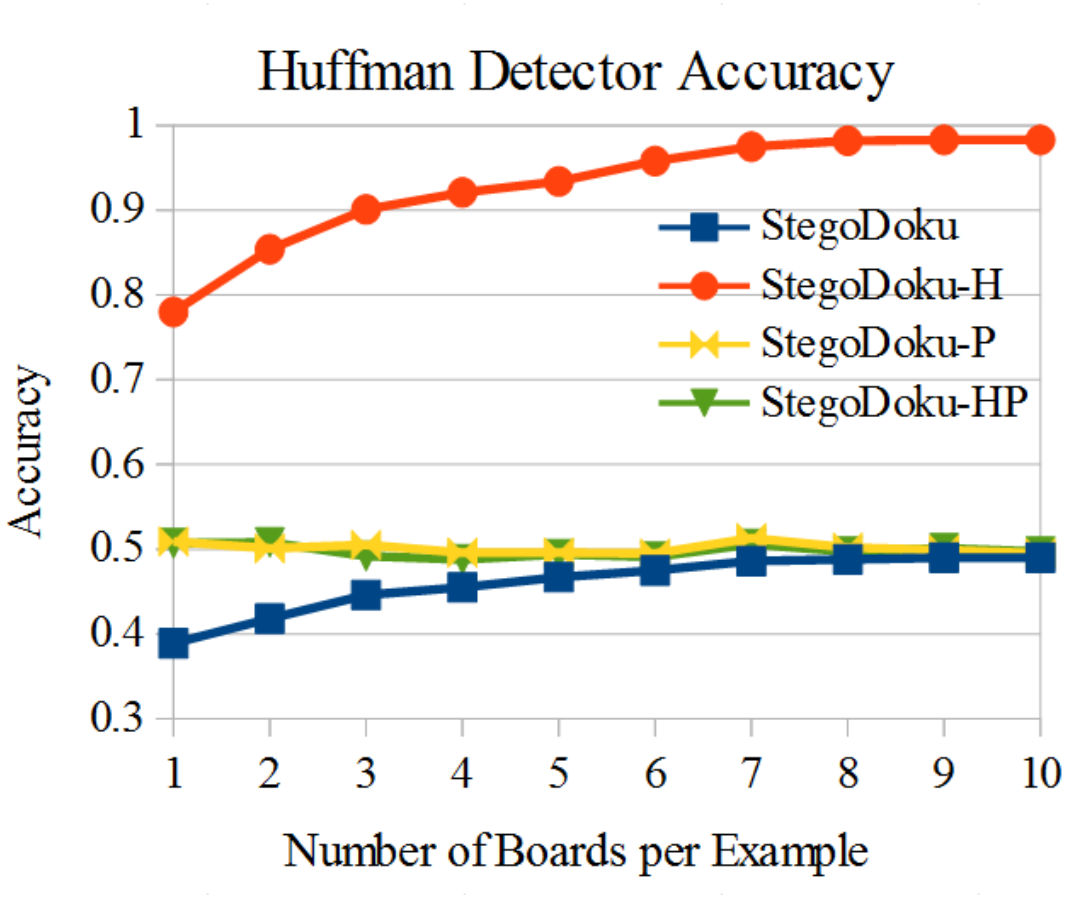


Figure 6.5. The detection accuracy of the Huffman-likelihood detector. As the number of boards per example increases, the accuracy against the systems which do not use permutations increases. Near perfect accuracy is achieved against StegoDoku-H around 10 boards.

Table 6.5.
StegoDoku detection accuracy.

	Stego-System			
Detector	**SD**	**SD-H**	**SD-P**	**SD-HP**
Structure	100.0%	100.0%	50.0%	50.0%
encrypted	50.0%	50.0%	50.0%	50.0%
OoB Candidates	99.9%	49.9%	50.0%	49.9%
Huffman ($n = 1$)	38.9%	78.0%	50.9%	50.7%
($n = 2$)	41.8%	85.4%	50.1%	50.8%
($n = 5$)	46.7%	93.4%	49.6%	49.4%
($n = 7$)	48.6%	97.5%	51.3%	50.6%
($n = 10$)	49.0%	98.3%	49.5%	49.7%
Padding (short)	99.9%	99.7%	49.9%	50.1%
(long)	50.1%	50.0%	50.2%	49.9%
(random)	50.0%	50.0%	50.0%	50.0%

The clue configurations may also give information that can be used to distinguish authentic Sudoku puzzles from steganographic stego-puzzles. For example, authentic puzzles are very likely symmetric in their clue configuration while stego-puzzles are very likely not. Therefore, if a detector were to classify symmetric puzzles as clean and asymmetric puzzles as dirty, it would have near perfect accuracy. That is, until the stego-puzzle maker was modified to generate symmetric puzzles. This modification would reduce the capacity of the puzzles somewhat, but would completely defeat the symmetry-based detector. Another potential avenue for detection based on clue configuration is to use the number of clues provided. The stego-puzzle maker adds more clues than are necessary for the puzzle to be solvable and therefore stego-puzzles may have a greater expected number of clues than an authentic puzzle, even though effort is made to reduce the number of clues actually provided. The detection utility of this feature can be reduced by modifying the stego-puzzle maker to generate puzzles with fewer clues. In the extreme, any clue configuration-based detector will be defeated by a stego-puzzle which is not actually hiding any data in the clue configuration, but only in the board. Finally, given that the StegoDoku system is known to the adversary, it is possible for the adversary to learn a signature for its generated clue configurations, both clean and dirty, if the clue configurations are generated deterministically. Therefore, if the clue configuration will not be used for data hiding, then it should be randomly generated. The efficacy of these approaches and the effect on the embedding capacity of the countermeasures designed to defeat them require further study that we propose to conduct in future work.

6.4 Conclusion

StegoDoku, a method for steganography by cover synthesis which uses Sudoku puzzles, has been presented and empirically analyzed. Recognizing that the basic algorithm did not fully exploit the capacity of Sudoku boards and that it resulted in a detectable signature, two enhancements to the basic StegoDoku algorithm were

proposed to improve both capacity and security: a Huffman code to map secret bits to candidates and a secret key to control permutations of candidates during embedding.

The StegoDoku system was analyzed in terms of the decoding error caused by backtracking, efficiency of generation, embedding capacity, and security against a passive adversary. In the analysis, it was shown that using Huffman coding during embedding increases capacity by an average of 8.722 bits over the average capacity obtained when not using Huffman coding, and does so without increasing the backtracking error. It was also shown that permuting the candidate set has a negligible effect on both capacity and backtracking error. Four steganalytic methods were proposed and analyzed. In accordance with Kerckhoffs's principle, it was assumed that the adversary knows the StegoDoku system, including the Huffman codes and permutation method. However, the adversary does not know the key. The use of the secret key to permute the candidate sets during the embedding process renders the resulting stego-boards indistinguishable from randomly generated authentic Sudoku boards. If the key value ever becomes known to the adversary, then she can achieve the same level of accuracy as if permutations had not been used.

For the methods without a secret key, StegoDoku and StegoDoku-H, testing the extracted message for a known encoding scheme, such as ASCII, resulted in 100% classification accuracy. When the secret was encrypted before being hidden, this detection method fails and can do no better than 50% accuracy, equivalent to guessing. By taking advantage of the way the basic StegoDoku algorithm selects candidates for cells, it was shown that boards generated by the StegoDoku algorithm can be distinguished from authentic Sudoku boards with 99.9% accuracy by checking for the use of out-of-bounds candidates during generation. However, the same method is unable to distinguish between authentic boards and boards generated by StegoDoku-H. Accurate detection of boards generated by StegoDoku-H was achieved by a method which compared the likelihood that a set of boards were all generated by StegoDoku-H with the likelihood that they were all generated by an authentic generator and classified the set based on which likelihood was greater. For a single board, this

technique achieves 78% accuracy. As the number of boards observed before making a decision increases, so too does the classification accuracy, becoming nearly perfect after observing 8 or 9 boards. Finally, stego-boards which are hiding messages which take up less than 85% of the available capacity can be detected with greater than 99% accuracy by checking for padding at the end of the extracted message. This detector is defeated, however, by using random padding or by sending a long enough message to take up the full capacity of the board so that padding is not required.

7 TIC-TAC-STEGO: COVERT CHANNELS IN COMBINATORIAL GAMES

A combinatorial game is a two-player game with perfect information and no chance moves [88]. Chess and Go are two examples of combinatorial games which have already been identified by other authors as being capable of supporting covert channels. Desoky and Younis [27] and Lange [26] present schemes for covert communcation via chess. Hernandez-Castro *et al.* present a general methodology for steganography in games and apply it to the game of Go [41]. However, these papers do not address the possibility that the games can be played naturally (as if by a human) and still transmit information.

The motivation for researching covert channels in games comes not only from the possibility of such channels, but also from the advantages which can be gained by exploiting them. For instance, Murdoch and Zielinski conducted a study which proved that a covert channel in the game of Connect Four could be used to collude with other players for the purposes of winning a league-style tournament [40]. However, there is no reason to believe that the aim of colluding through a game should be limited to winning a tournament of that game.

In this chapter, the general task of exploiting covert channels in combinatorial games is approached by examining the game of Tic-tac-toe. For ease of exposition, a standard framework for studying covert channels, known as the Prisoners' Problem [89] is drawn upon. In Section 7.1, the initial conditions of Alice, Bob, and Wendy's situations are presented along with Tic-Tac-Stego, a general methodology of exploiting covert channels in combinatorial games. The methodology is demonstrated in Section 7.2 by applying it to the game of Tic-tac-toe. Experimental results on the capacity and security of the Tic-tac-toe stego-system are presented in Section 7.4. Section 7.5 mentions other games which have not been covered in the existing literature to which the general theory can be easily applied. Section 7.6 concludes the

chapter with a summary. A portion of this chapter was presented and published in the Proceedings of the International Workshop on Distributed Simulation and Online Gaming in 2012 [90].

7.1 Tic-Tac-Stego Methodology

Alice and Bob are engaged in a series of communications in which they are playing a two-player game, likely more than once. The channel is public but trusted. Wendy is a passive adversary who observes all communication between Alice and Bob looking for messages which contain secret information. Alice and Bob do not want Wendy to learn their secret, but messages which appear to be encrypted are prohibited on public channels. So, Alice and Bob decide to use steganography to hide their encrypted communication. Messages containing the game data may not be the only communication they send and receive, but such messages are sufficient for Alice and Bob to exchange information covertly. Wendy's task is to distinguish between when Alice or Bob are hiding information (covert channel usages) and when they are not (authentic channel usages).

In [41], Hernandez-Castro *et al.* describe an approach to hiding data in games which is built upon in this work. Each player has a set of strategies which specify the move that player should make, given the action sequence leading up to that player's turn. There may be multiple strategies defined for any given action sequence, allowing a player to choose between a set of moves. The set of moves is assumed to be ordered according to the expected payoff of the move.

Under typical playing conditions, each player is expected to respond to the given action sequence with the best move from their set of strategies. However, the selection of a specific move from the set of all possible moves can be used to send information to the other player or an observer. Assuming candidate moves are equally likely, a player could send $log_2(N)$ bits of data per move, where N is the number of options available to the player for that move.

Tic-Tac-Stego is a general methodology for covert communication in games and is a special case of a more general technique for generating steganographic objects, which Fridrich calls steganography by cover synthesis in [81, section 4.2]. The idea behind it is actually very simple: whenever there is a choice between two or more options, there is capacity to send information using that decision. In Tic-Tac-Stego, the cover being generated is a sequence of moves in a combinatorial game. The maximum capacity for each turn is achieved when every legal move is included in the set of candidate moves. However, not every legal move is a good move, and some legal moves are positively bad. The distinction between good moves and bad moves is made using a strategy. Formally, a strategy is a mapping from board states (or histories) to a subset of the legal moves for that board state. This subset is the candidate move set, but might also be referred to as the set of good moves. Figure 7.1 depicts the Tic-Tac-Stego encoding and decoding methodologies.

Encoding bits of secret information works as follows. Alice starts with some board state and uses her strategy to determine the set of good moves. To each of these moves she assigns a bit sequence. She uses some bits of her secret information to select a move from the candidate move set. Specifically, she selects the move whose bit sequence matches the bits she took from her secret. She applies the move to the board and sends the updated board to Bob, her opponent and communication partner. At the receiving end, Bob can decode the bits as follows: Bob receives a board from Alice and can determine which move Alice made by comparing it to the board state before Alice made her move. Bob also knows the strategy that Alice is using, so Bob can also compute the set of candidate moves from which Alice chose her move. Bob assigns to each candidate move the same bit strings as Alice did. Because Bob knows not only which move Alice made but also the bit sequence for that move, Bob therefore knows some bits of Alice's secret. Bob can then make his move, either sending some secret bits of his own or just playing to allow Alice to continue sending her secret, and send the updated board back to Alice. If Alice expects Bob to also send information to her, she can extract the secret bits using the same method as

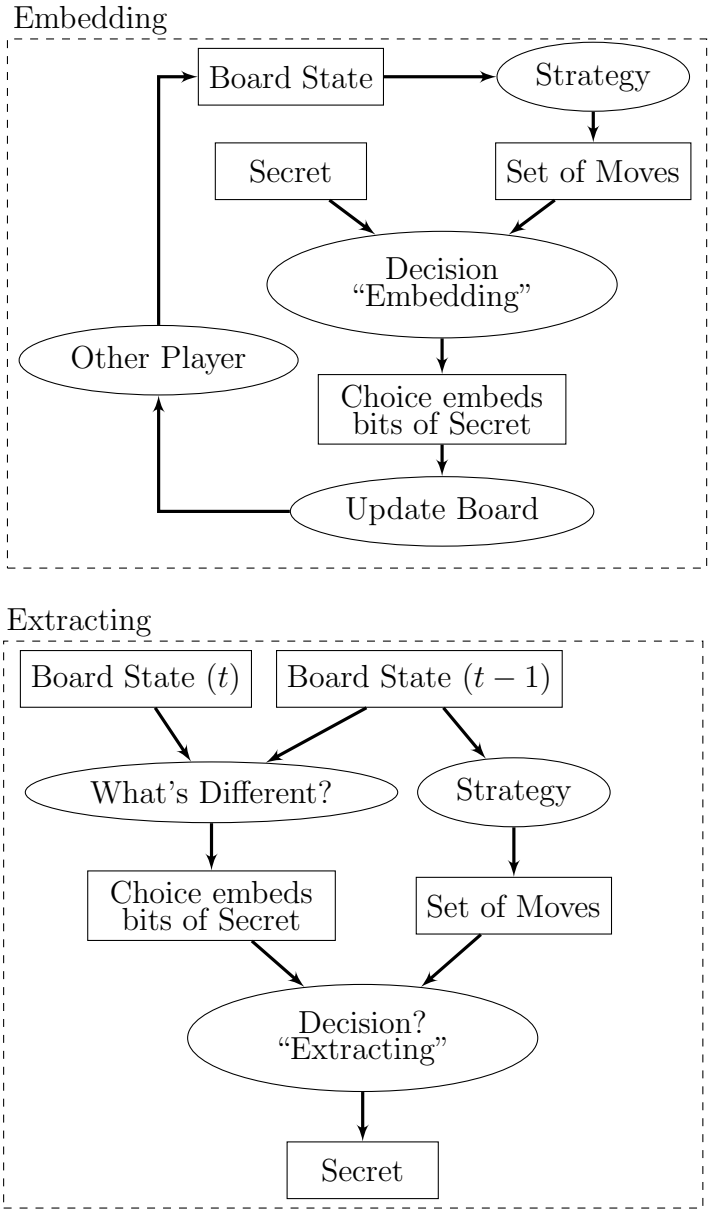


Figure 7.1. The Tic-Tac-Stego methodology's processes for embedding and extracting hidden data. In order to be able to extract the correct bits, the receiver must know the sender's strategy.

Bob. Alice sends the next few bits the same way as before. This process continues until all of Alice's and Bob's secret bits have been sent.

There are several techniques that Alice and Bob could use to protect their hidden data and help the cover generation process to hide the data. One such technique is to compress and encrypt the data, making it smaller and more random. We can view the cover generation process, as Sallee does in [21], as being a mapping from random bit sequences to valid elements of some cover-space. The hypothesis is that the more random the message appears, the more convincingly diverse the generated objects will be. A second technique is to use a shared random number generator and a seed to randomly permute the move labels at each step. This provides an additional layer of security by acting as a variable length substitution cipher so that any given move does not always encode the same data. Another technique is to use the channel only occasionally, either on a game-by-game or move-by-move basis, which can be accomplished by the usage of the same or another shared RNG and seed. This forces Wendy to decide when individual games or moves are part of the covert channel and when they are not and is similar to the technique of Chaffing and Winnowing [91].

7.2 Tic-tac-toe

Tic-tac-toe is a two-player turn-based pencil-and-paper game played on a 3x3 grid, wherein players attempt to place three marks in a row in order to win. Each player uses marks of a single kind, player-1 uses `X` and player-2 uses `O`, and player-1 usually goes first.

If Alice and Bob wanted to play Tic-tac-toe, they could do so by sending a piece of paper back and forth (always through Wendy) and take turns making their marks. They could also use computers and play the game through a digital medium, such as images or text. The specific communication medium is inconsequential, as Alice and Bob can use the game of Tic-tac-toe to communicate covertly in any medium. This section will detail how the game of Tic-tac-toe can be used as a covert channel and

will explore ways in which Wendy can attempt to detect and neutralize the usage of the channel.

7.2.1 Tic-tac-toe as a Covert Channel

The essence of a covert channel in a combinatorial game is that Alice must make a choice of which move she will make during her turn and that her choices can be used to embed hidden information in her gameplay. When using Tic-tac-toe, Alice will encode her secret information as mark locations in the Tic-tac-toe grid.

Assume that Alice goes first and that her strategy simply tells her that all moves are equally good so that, during her turn, she can choose to place her mark at any open location. On her first turn, Alice has nine choices of where to place her mark, thus her placement encodes three bits. Bob then makes his move, leaving Alice with seven locations for her second mark. So, with her second move, Alice can send two bits . Again, Bob makes his move and Alice is left with five locations for her mark, allowing her to send two more bits. At this point, Alice has sent seven bits. If she did not win with this move, it is possible that Bob will win with the next move. However, if Bob does not make a winning move during his turn, Alice gets to place another mark in one of the three places Bob has left for her and she can encode one more bit with her final choice. This is the last bit Alice can send since, even if Bob cannot win during his turn and she gets to place her last mark, there will only be one location in which she can place it and thus she cannot encode any additional information.

Alice is guaranteed to be able to send seven bits under this strategy, since neither Alice nor Bob can win the game until Alice has placed her third mark. Furthermore, the exchange between Alice and Bob can be used to simultaneously transfer information in both directions. Assuming that Bob's strategy also lets him play anywhere that is open, his first move has eight choices and thus encodes three bits. His second choice is from six options, encoding two bits. If he gets a third turn, he will choose from four options and encode another two bits. If he gets yet another turn, his fi-

nal choice between the two remaining spaces will encode one additional bit. Bob is guaranteed to be able to send five bits, with high likelihood of being able to send seven or eight bits. When not using his channel capacity, Bob can allow Alice to send an eighth bit by forcing a draw since it is always possible for either player, playing perfectly, to force a draw in Tic-tac-toe [92].

7.2.2 Properties of the Channel

The most important properties of an information hiding system are steganographic security, robustness and capacity. The security of an information hiding system is a measure of the indistinguishability of its stego-objects from clean cover-objects. The robustness of the system is a measure of the stego-objects' resistance to tampering. And the capacity of the system is a measure of the amount of secret information the stego-objects can hold.

Three strategies of play are identified and their capacity and security against Wendy, a passive adversary, is analyzed. The first strategy is Random, which allows the player to place their mark in any open grid cell. The next is Greedy, which limits the player to winning or blocking when possible, but is otherwise equivalent to Random. The last is Optimal, which limits the player to those moves which lead to the best outcomes, winning, if possible, or else drawing.

Since the public channel is trusted and Wendy is a passive adversary, Alice and Bob know that what they send to the other arrives unchanged. Therefore, the possibility of Wendy tampering with the stego-objects directly and dropping or forging messages can be ruled out. Thus, the issue of robustness can be safely skipped, for now.

Security

For Tic-tac-toe, steganographic security means that Alice and Bob want Wendy to have as much difficulty as possible distinguishing between games of Tic-tac-toe which are hiding information and games which are not. Since, by Shannon's maxim,

the enemy knows the system, one way for Wendy to test for hidden information is to assume that all games contain stego and attempt to extract the hidden data. If the extracted data is not garbage, Wendy will mark the games as stego, otherwise, they will be marked as clean. However, if Alice and Bob have compressed or encrypted their data, Wendy will have difficulty determining whether the data she extracted is truly garbage without secret key information (e.g. the encryption key and RNG seeds). Additionally, Wendy may also need to determine which games or moves she should use when extracting data, which would again require knowing secret key information. If Alice and Bob can keep their keys secret, then Wendy has no hope of using this method for detection.

Another way Wendy can test for hidden information is to assume that the presence of hidden information will cause the game or move to differ from the expectation for that move or game. That is, Wendy will test whether the moves Alice and Bob make agree with her model of how the game should be played. In this case, Wendy's model is a set of strategies for playing the game which dictate which moves are expected and which are not. If Wendy sees a move which she did not expect she will classify the game as stego. The simplest such model for Wendy is to check for obviously bad moves. To avoid this, Alice and Bob can update their strategy from Random to Greedy, so that they no longer make any obviously bad moves. In response, Wendy can update her model to reflect an expectation of perfect gameplay, possibly on the assumption that perfect gameplay does not leave room for covert communication. Under Greedy, Alice and Bob will occasionally be forced to make a suboptimal move in order to send some data, which Wendy will catch and use as evidence of the existence of a covert channel. As with Random, Greedy will occasionally cause Alice and Bob to play sub-optimally even when not sending data. However, Alice and Bob can further upgrade their strategy to Optimal in order to avoid making the mistakes for which Wendy is looking. Additionally, it turns out that Wendy's assumption that perfect play precludes covert communication is false; Alice and Bob can play perfectly and still have covert channel capacity available.

Table 7.1.
Analytical Tic-tac-toe channel capacity.

	Capacity				
	Total (bits)		Mean (bits)		Mean Game
Strategy	**Min**	**Max**	**Alice**	**Bob**	**Length (moves)**
Random	12	16	7.974	7.774	8.255
Greedy	8	16	7.455	6.650	8.7398
Optimal	6	15	7.489	4.326	9

Capacity

The capacity of the covert channel depends on the play-strategy that Alice and Bob use. Their play-strategy, as shown in the experimental results of Section 7.4 affects the security that Alice and Bob have against Wendy's analysis. Analytical results on the channel capacity and game length under different strategies are given in Table 7.1. The means are taken over all possible action sequences (game paths, or histories) for games played under each strategy.

7.3 Algorithms

There are two general functions, Embed and Extract, which are necessary to carry out communication over covert channels in combinatorial games. Algorithm 9 contains the embedding process which is carried out on each turn to send data through the channel and Algorithm 10 shows the extraction process which is carried out on each turn to receive data from the channel.

To embed data in a game, the set of best moves are computed for the given board. The size of the returned set is used to compute the number of bits of information which will be consumed in this turn. The set of best moves is randomly permuted using a shared random number generator and seed. The embedding is accomplished

by extracting and consuming the correct number of bits from the secret and using them as the index value into the permuted set of best moves. The chosen move is made and a new updated board is returned, along with the updated seed and message.

The first several steps of the extraction process are identical to those of the embedding process: determining the best moves given the board your opponent saw, permuting the set of best moves and determining the number of bits to expect. Comparing the old board to the new reveals which move the opponent selected, the index of which is converted to a binary number containing the correct number of bits. The result is returned as the next fragment of the message being received.

In practice, each turn would consist of an attempt to read data from the channel followed by an attempt to write data to the channel. The determination of when reads and writes contain data or garbage is made external to the algorithms presented here.

Algorithm 9: Embedding process for combinatorial games.

Input: Board B, Message M, Seed S

Output: Board newB, Message newM, Seed newS

$bm = GetBestMoves(B)$

$n = \lfloor \log_2(|bm|) \rfloor$

$bm = RandomPermute(bm, S)$

$newS = RNG(S)$

$m = M[1 : n]$

$newM = M[n + 1 : end]$

$md = toDecimal(m)$

$move = bm[md]$

$newB = applyMove(move, B)$

return $newB$, $newM$, $newS$

Algorithm 10: Extraction process for combinatorial games.

Input: Board oldB, Board newB, Seed S

Output: Output: MessageFragment m

$bm = GetBestMoves(oldB)$

$n = \lfloor \log_2(|bm|) \rfloor$

$bm = RandomPermute(bm, S)$

$newS = RNG(S)$

$move = extractMove(oldB, newB)$

$md = find(bm == move)$

$m = toBinary(md)$

return m

7.4 Experimental Results

Nine parameter sets for Alice and Bob are made up of a choice between three game-playing strategies and a choice between three embedding techniques. The three game-playing strategies Alice and Bob can use are Random, Greedy and Optimal. Under the Greedy strategy, it is still possible for Alice or Bob to send data when forced to play "smart", such as in the case of Tic-tac-toe when a player can win or block in more than one location, providing a choice and thus capacity for covert communication. The Optimal strategy uses the Negamax algorithm [93] to compute the set of best moves as those which maximize the current players pay-off, which is the negation of the opposing players pay-off since Negamax assumes that the game is zero-sum. Whenever there is more than one best move, a player has the opportunity to send hidden information through the channel. The three embedding techniques that Alice and Bob use are Every Game, Random Games and Random Moves. The Every Game technique causes Alice and Bob to attempt to send data on every move of every game, in order to maximize the utilization of the channel. Using the Random Games technique, Alice and Bob will only send (and expect to receive) data during games chosen at random by a random number generator and seed shared by both Alice and Bob. Every move of a chosen game is used for data transfer. When using the Random Moves technique, Alice and Bob only send and receive data on moves chosen at random by their shared random number generator.

7.4.1 Capacity Results

Figure 7.2 contains the mean embedding capacity results for different values of $P(embed)$, the likelihood that a game or move is chosen to be used for transmitting data, for each of the different embedding techniques when applied to Tic-tac-toe. Only the Random Games and Random Moves techniques are shown because the Every Game technique does not depend on $P(embed)$ and is equivalent to the capacity measured when $P(embed) = 1$, which is shown on both plots.

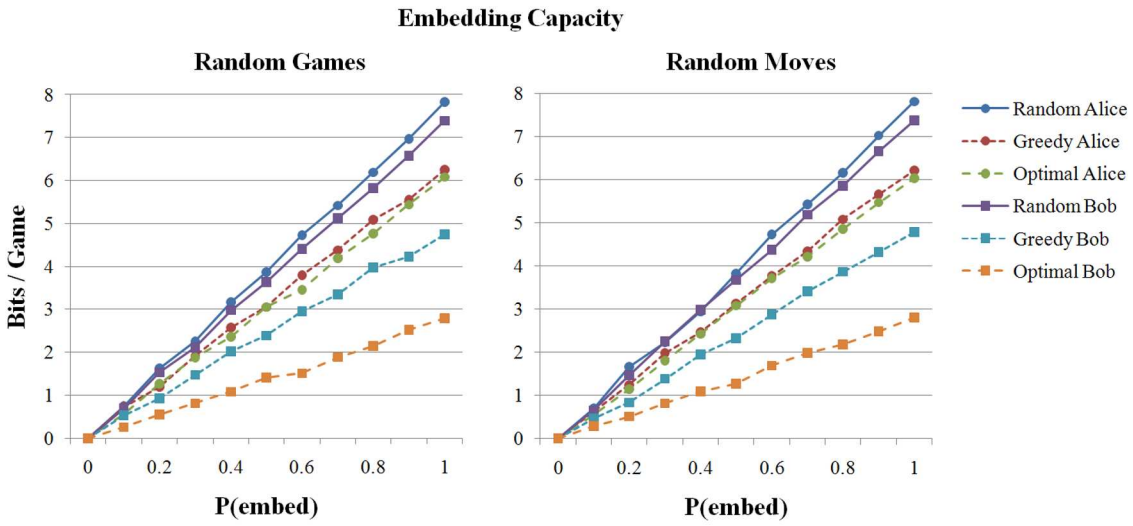


Figure 7.2. Experimental results for the embedding capacity of Tic-tac-toe.

In the results for the Random Games technique we can see that Alice, playing randomly for full utilization of the channel, has the highest capacity followed by Bob playing randomly, both being between seven and eight bits per game when $P(emebed) = 1$. Next, Alice playing Greedy and Alice playing Optimal have very similar capacities, just above six bits per game at most. Greedy Bob sends just under five bits per game while Optimal Bob sends just under three bits per game. The embedding capacity for each player increases linearly with the value of $P(embed)$. The results for the Random Moves technique, which are very similar to those of the Random Games technique, shows that the embedding capacity is not much affected by the choice between Random Games and Random Moves. One reason for the lack of effect is that embedding in $p\%$ of games is, on average, equivalent to embedding in $p\%$ of moves.

7.4.2 Security Results

Figure 7.3 displays the results of the security tests for the two embedding techniques, Random Games and Random Moves. In these tests, the different parameter

sets for Alice and Bob's stego-system are pitted against two Wendys, each with a strategy for determining which games are suspected of containing hidden information, and Wendy's mean error rate is recorded. Wendy makes an error whenever she classifies a clean game as dirty (false positive) or a dirty game as clean (false negative). Alice and Bob's security is inversely related to Wendy's error rate; the more errors Wendy makes, the more secure Alice and Bob's covert channel is because Wendy has more difficulty distinguishing clean exchanges from those containing hidden information. The Greedy Wendy makes her decision by checking for moves in which Alice or Bob failed to make an obviously good move when they had the chance. If a game contains such a move, it is classified as suspicious. The Optimal Wendy makes her decision by checking for non-optimal play, that is any move which is not in the set of best moves given the state of the game.

The cases of Greedy Wendy against Random Alice and Bob, Optimal Wendy against Random Alice and Bob and Optimal Wendy against Greedy Alice and Bob all have error rates which decrease linearly with $P(embed)$. When no embedding is taking place, at $P(embed) = 0$, Wendy has a high error rate because she is mistaking Alice and Bob's poor play for covert communication, resulting in mostly false positives with a few true negatives whenever Alice and Bob get lucky to stumble into what appears to be a clean game. When $P(embed)$ is near 1, Wendy has a low error rate because her propensity for classifying games as dirty is matched by the proportion of games which actually contain hidden information. The opposite trend occurs for the cases of Greedy Wendy against Greedy Alice and Bob, Greedy Wendy against Optimal Alice and Bob and Optimal Wendy against Optimal Alice and Bob. In these cases, a low value of $P(embed)$ results in Wendy having a low error rate because Alice and Bob's games are well-played and do not confuse Wendy. As the embedding frequency increases, so too does Wendy's error rate. This is because the games are not looking any less well-played despite the increasing frequency of covert transmission. When Alice and Bob are sending bits on every move of every game, Wendy's error rate

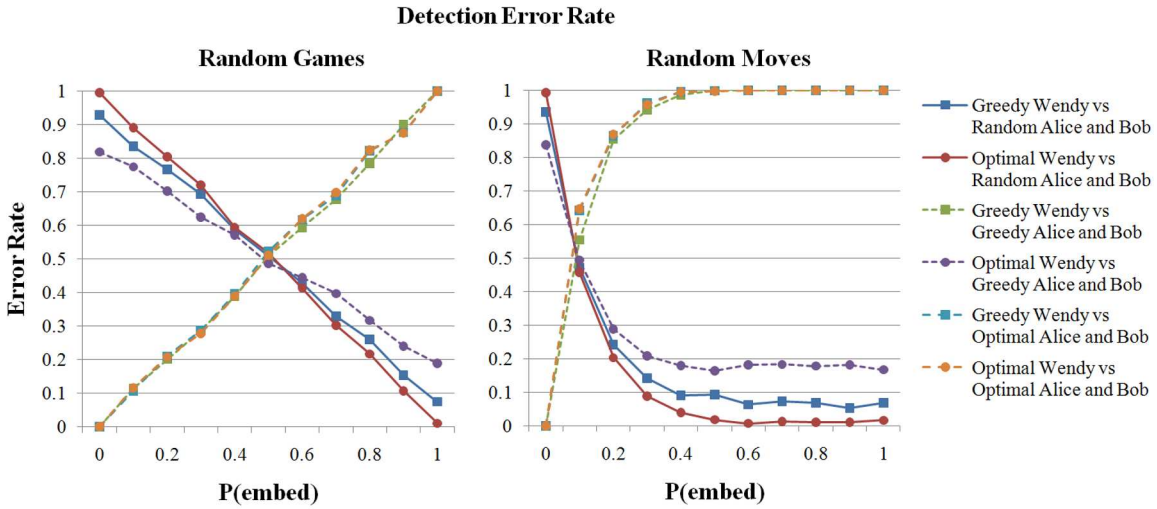


Figure 7.3. Experimental results for the detection error rate of Tic-Tac-Stego.

is 100% because the games still look well-played according to her decision making process.

In Figure 7.3, the Random Moves technique has interesting results for security. We see the same general trend of Wendy's error rate decreasing as the rate of transmission increases when Wendy has the better model and increasing when Alice and Bob have an equivalent or better model. However, the relationship is no longer linear. This is because Wendy is making her classification based on games and not on a move-by-move basis. At $P(embed) = 0.5$, only half of the moves are used to send data but they occur in almost every game, which is equivalent to every game being used to send data at a lower rate by half. This means that when Wendy classifies a game as being suspected of transmitting data, she does not know which or how many moves were used to transmit data. Thus, Wendy's correct classifications are useful for detecting *that* covert communication has occurred, but not *what* was sent since Wendy cannot extract the hidden message.

The security tests were revised to allow Wendy to make her classification on a move-by-move basis and the results are shown in Figure 7.4. The results for both the Random Games technique and the Random Moves technique are very similar to

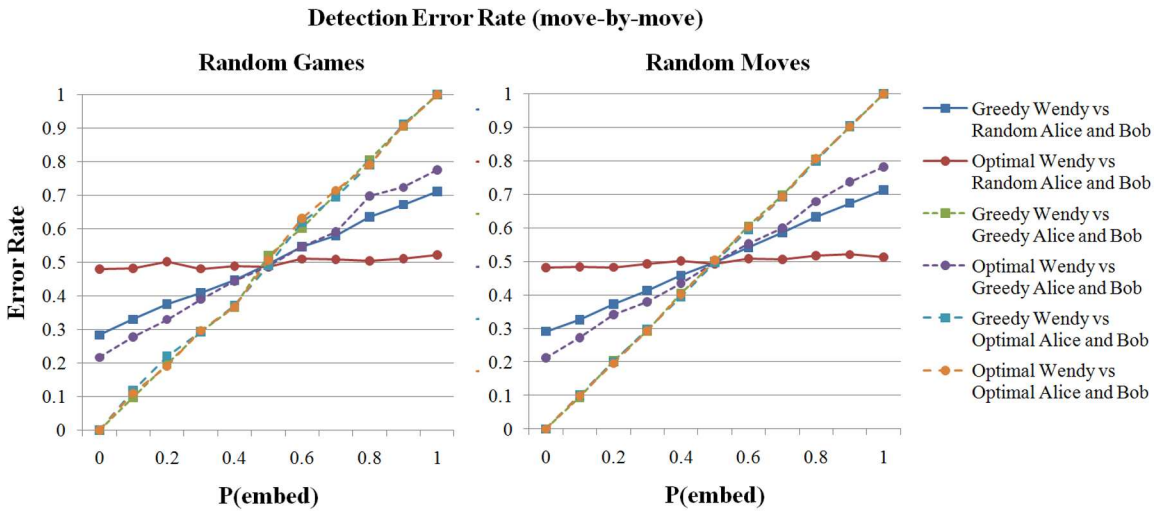


Figure 7.4. Experimental results for the move-by-move detection error rate of Tic-Tac-Stego.

each other but differ greatly from the game-by-game analysis in the three cases where Wendy has a better model of gameplay than do Alice and Bob. The other three cases, where Alice and Bob have a model which is at least as good as Wendy's, produce much the same results as they did in game-by-game analysis. Under move-by-move analysis, Wendy's error rate increases with $P(embed)$, except in the case of Optimal Wendy against Random Alice and Bob which is just about 50% regardless of the value of $P(embed)$. This is because most of the moves Alice and Bob make seem completely normal to Wendy, except when Alice and Bob are playing Random, in which case half of their moves are inadvertently played perfectly.

The likelihood that Wendy considers a given move to be suspicious depends only on the two models of gameplay being pitted against each other. From the plots, we can see that Greedy considers about 30% of Random's moves to be suspicious and Optimal considers nearly 50% of Random's moves, and approximately 20% of Greedy's moves, to be suspicious. Greedy does not consider its own or Optimal's moves to be suspicious, and similarly neither does Optimal consider its own moves to be suspicious.

These results show that Wendy's advantage over Alice and Bob is dependent entirely on the relative quality of her model of the cover-object space compared to that of her opponents. When Alice and Bob have a model of the cover-object space which is as good as or better than Wendy's model, then they can sneak hidden information past her without any problems. When Wendy has the better model, Alice and Bob are easily defeated.

7.5 Other Games

The framework presented in this paper is applicable to other games as well and supports schemes which are substantially similar to those presented in the previous work on this topic. The framework can also be generalized to be applicable to multi-player games and games without perfect information. This section will briefly explain the exploitation of covert channels in two other games which are popular games to play online.

The first game is Dots and Boxes, which, like Tic-tac-toe, is a turn-based pencil-and-paper game played on a grid. However, it can be played by more than 2 players and the grid for Dots and Boxes is very much larger than the grid for Tic-tac-toe. A typical Dots and Boxes grid may be larger than 9x9, where Tic-tac-toe is only 3x3. Also, instead of making moves in the cells, moves are made at cell boundaries, the goal being to completely surround a grid cell in order to claim it for oneself. The winner of the game is the player with the most boxes of their own at the end of the game. The large board size results in a much larger state-space for this game, which makes computation of globally optimal choices expensive in terms of time and memory. Such a large initial state-space also means that seemingly random play early in the game is not out of the ordinary and a good-enough endgame strategy is often enough to secure a win against one's opponent or opponents. The large board coupled with the nearly complete freedom of the first several turns means that the capacity of the covert channel is much larger than for Tic-tac-toe. For instance, an $n \times n$ grid

for a game of Dots and Boxes contains $(n+1)^2$ dots and $2n(n-1)$ borders linking the dots. Thus, under the null strategy, Alice can send approximately $\frac{1}{N}log_2(2n(n-1)!)$ bits of information per game, as can each of the other $N-1$ players. Using a strategy which aims to win, or, equivalently, to not lose, will reduce this capacity but will allow the bits to be sent less detectably.

The second game to mention is Battleship. While still a 2-player turn-based pencil-and-paper game, Battleship is not actually a combinatorial games since the players do not enjoy full information while the game is being played. Before the first move is made, players hide a set of ships of varying length in a 10x10 grid with the hopes of having their well-placed fleet survive bombardment by the opposing player. The players take turns bombing a grid location belonging to their opponent and are told only if they hit or missed. The player who sinks all of their opponent's ships first wins. There are actually two independent covert channels in the game, one using ship placement and one using bombing locations. Using ship locations, the hidden message is encoded as a permutation of ship orientation and location. Once a player has won, she will know the location and orientation of all her opponent's ships, from which she can extract the hidden message. Using bombing targets to implement the covert channel is very much similar to the process used in Tic-tac-toe and Dots and Boxes: each choice encodes a bit sequence and the choices are driven by the bits of the secret information being transmitted. Since a player's ship placement and choice of targets are independent, the total capacity of the covert channel in Battleship is the sum of the capacities of using each individually. The capacity of the ship-placement channel is approximately 34 bits and that of the bombing target channel is 21 bits, thus the capacity of both used together is 55 bits per game.

7.6 Conclusion

The Tic-Tac-Stego methodology is applicable to any combinatorial game and only requires that the communication endpoints know each other's strategy and move-

labeling scheme. The security of the channel against a passive adversary (one which can observe but not modify the contents of the channel) relies upon the secrecy of the move labeling scheme and the characteristics of play produced by the strategies. The move labeling scheme (actually, just the seed that controls it) must be secret because, if Wendy knows how moves are labeled, then she can extract the secret bits that Alice sends just as easily as Bob can. If the strategies the players use produce play which is distinguishable from normal play, then Wendy can detect the usage of the channel (prompting her to disconnect Alice from Bob permanently, per the rules of Simmons' Prisoners' Problem). Therefore, it is important to Alice and Bob that they have access to strategies that produce gameplaying behavior which appears to be human-generated. In general, Wendy can only detect the usage of the covert channel when she has a model which is better, or more accurate at predicting or describing authentic behavior, than Alice and Bob's (c.f. [11, 21, 86]). Therefore, it is also important to Wendy that she has access to a model of gameplay which is better than that of Alice and Bob.

The application of this framework to the game of Tic-tac-toe was demonstrated and it was shown that, even in the case of such a simple game, Wendy cannot accurately distinguish stego-games from clean games and nor can she, through application of the strict requirement of forcing perfect play, reduce the capacity of the channel to zero and thereby prevent covert communication. Experimental and analytical results on the capacity of the channel showed that playing optimally reduces, but does not eliminate the capacity of the channel. Experimental results on the steganographic security of the channel showed that Wendy's ability to accurately detect usage of the covert channel comes down to whether or not her model of play is better than Alice and Bob's. We also described how to exploit covert channels in multiplayer games and games without perfect information, such as Dots and Boxes and Battleship, which are more complex and have much larger covert channel capacity than Tic-tac-toe.

Multiplayer games are abundant on the Internet and websites exist which provide the means for users to play these anonymously games with others. The difficulty

that Wendy has in detecting and preventing covert communication between Alice and Bob in isolation is compounded many times when Alice and Bob have access to anonymizing services which they alone can circumvent using covert authentication and communication through online games.

8 ANOMALY DETECTION FOR STEGANALYSIS OF GAMES

As the Warden in Simmons' Prisoner's Problem, Wendy must suspect that Alice and Bob will attempt to use a stego-system to establish a covert channel, and so will be on the lookout for suspicious messages that might indicate the usage of such a channel. In this regard, Wendy's job is very much like that of an anomaly detector which attempts to identify activity which is out of the ordinary, and therefore possibly indicative of "bad things" happening. On the other hand, Alice and Bob's goal is to generate stego-objects that will not seem anomalous.

In this chapter experiments will be conducted to determine how well, if at all, Wendy can distinguish between games played by humans and games played by not-humans (i.e. computer agents). The implication made by using this distinction for steganalysis is that human-played games do not contain hidden information but that games which are not played by humans could contain hidden information. Also, simply by virtue of having not been played by a human, such computer-generated games are suspicious. Therefore, the task of detecting the usage of a covert channel in a game is equivalent to determining whether or not the agent playing the game is a human. If a human is playing the game, the game is clean. But, if a computer is playing the game, it can be assumed to be dirty. In the experiments which follow, all dirty examples have been generated by the Tic-Tac-Stego system using the three rules-based strategies: Random, Greedy, or Optimal.

8.1 Human Gameplay Data Collection

Human gameplay data for Tic-tac-toe is collected through 2 web applications which differ in the manner in which they collect the data. The first web application shows the user a randomly selected board state from the set of all games still in

progress (i.e. those games which have not yet ended, either by one player winning or the game ending in a tie) and asks the user to select the next move in the game. If the {game, move} pair already exists in the database, the count of the number of times that move has been made in that game is incremented. Otherwise, the new {game, move} pair is added to the database and its count initialized to one. Each entry in the database also records the parent game from which it was generated. Figure 8.1 shows an example of the online interface through which gameplay data is collected.

The other web application engages the user in complete games of Tic-tac-toe. This second data collection system more closely simulates the usual experience that humans have with games, which is to play them from start to finish rather than in a random order. The user is first shown either a blank board or a board with a single move already made. The user's response to this board is recorded in the database. In order to continue playing the game the human started, the application must select a move of its own. To retain some of the flavor of the first data collection application, which only shows the user boards generated by humans, the application attempts to make a move which it has seen a human make in the past. If it cannot find such a move, it plays at random. If more than one human move has been observed for the current board state, the application will select one of the candidate moves with probability proportional to its frequency. We refer to this second type of data collection as *sequential*, indicating that gamplay examples are collected in sequence, in order from the start of the game to the end of the game.

For both applications, the database initially contains only the game which has no moves and is represented by an empty game history. In total, the two web applications collected 18990 examples of human Tic-tac-toe gameplay, 10011 from the non-sequential collector and 8979 from the sequential collector. The exact number of human contributors to the dataset is unknown, since data collection was completely anonymous, but the authors know of six subjects who chose to notify us of their participation.

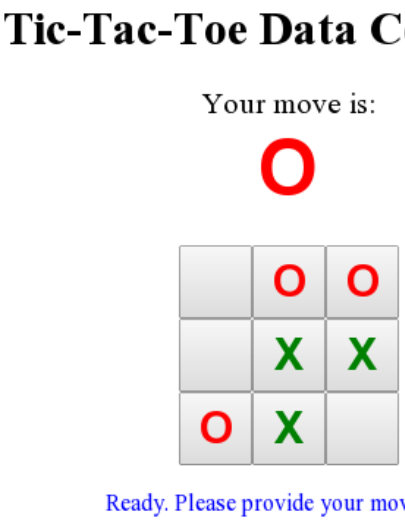


Figure 8.1. The user interface of the online Tic-tac-toe data collection application. To avoid confusion over which player (X, or O) the human player is playing, the symbols are color-coded. The user is always red, and the opponent is always green.

Human gameplay data for the game of Connect4 was collected through a web application very much like that used for data collection for Tic-tac-toe. Figure 8.2 shows an example of the online interface through which gameplay data was collected.

For Connect4, only sequential data is collected. That is, the user plays as either first or second player for an entire game, from start to finish. The computer plays using moves that it has observed a human make previously. Once the game goes "off book", meaning that the board state is completely new, the computer will play using a greedy strategy, attempting to win if it can, or blocking the human's win if it can, or otherwise playing at random. Initially, the computer's database of observed gameplay contains only the game which has no moves and is represented by an empty game history. So far, 1172 examples of Connect4 gameplay have been collected.

After collection, the moves are processed to obtain a weighted directed acyclic graph of boards starting from the root empty board to terminal end-of-game boards, where the weight of each edge e from node u to node v is proportional to the number of times a human shown the board at node u chose the move corresponding to edge e resulting in the board at node v. Random walks of this graph generate samples from the observed set of human-played games.

8.2 Experimental Results

Three kinds of detectors were implemented and tested. The first type of detector was a rule-based detector, which classifies gameplay using a set of simple rules. The second type is a feature-based detector which uses machine learning on a set of computable features to classify gameplay. The third is a probabilistic model-based detector which uses Bayesian hypothesis testing to classify gameplay. The metric used to assess the performance of each detector is the classification accuracy on a test set containing an equal number of clean and dirty examples. Accuracy is defined as the proportion of correct classifications:

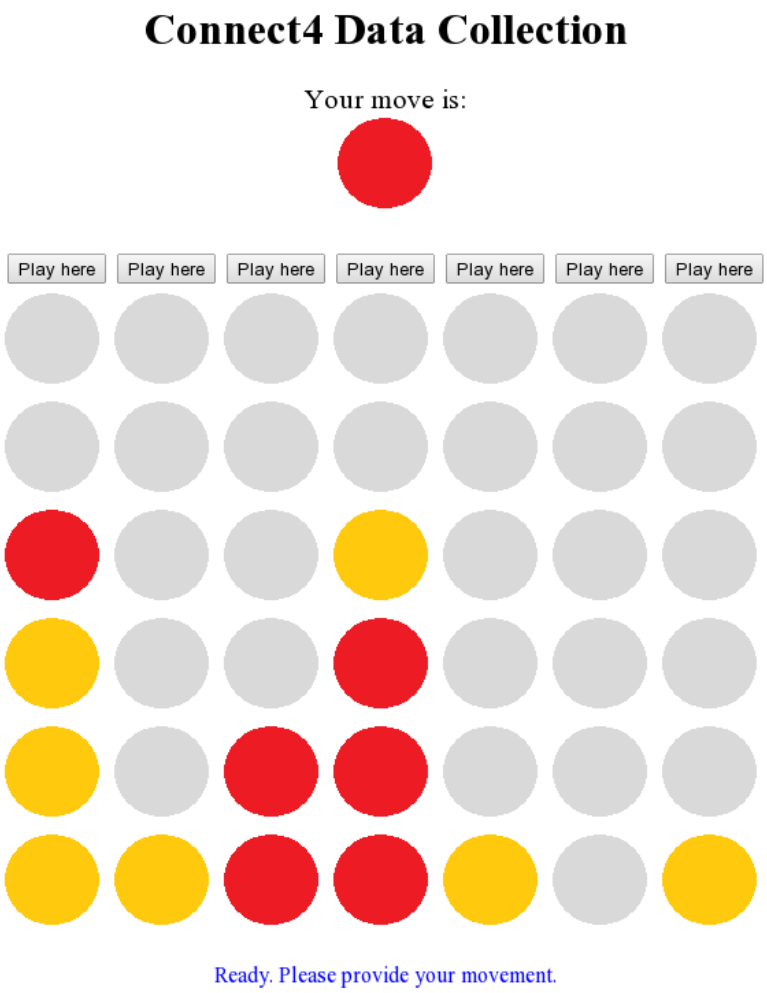


Figure 8.2. The user interface of the online Connect4 data collection application. The user is always red, and the opponent is always yellow.

$$ACC = \frac{1}{N}\sum_{i=1}^{N} 1_{\{c_i\ =\ t_i\}} \tag{8.1}$$

where c_i is the detector's classification of example i, t_i is the true class of example i, and $1_{\{c_i\ =\ t_i\}} = 1$ only when $c_i = t_i$ and is 0 otherwise.

8.2.1 Rules-Based Detection

The first type of detector evaluated was a rules-based detector. Rules-based detectors classify objects based on a static set of rules which the cover media is assumed to obey. Two rules-based detectors were tested. One used a rule set which defines greedy play. Greedy play means that the player must win if they can, and must prevent the other player from winning if they can. If the player can neither win nor block their opponent from winning this round, then they are free to make any legal move. The other rules-based detector used a rule set which defines optimal play. Optimal play means that the player must always make a globally optimal move which will guarantee the player the best possible outcome reachable from the current board state.

Using a rules-based detector a threshold value on the allowed deviation from the rule set can be defined. A threshold value of t means that no more than t moves which violate the rule set are allowed without triggering an automatic positive classification as suspicious. A threshold value of $t = 0$ means that no deviation is allowed. So long as the number of rule violations stays beneath the threshold, the play will be considered innocuous. However, in the cases where $Detector \subseteq Computer$, that is where the detector uses a rule set which is a subset or equal to the rule-set used by the computer agent, the decision must be flipped. In such cases, deviation from the detector's rule set is actually evidence of human gameplay and so conformance to the detector's rule set is actually evidence of computer-generated gameplay. The cases where this occurs are the Greedy detector against the Greedy and Optimal agents, since $Greedy = Greedy \subset Optimal$, and the Optimal detector against the Optimal

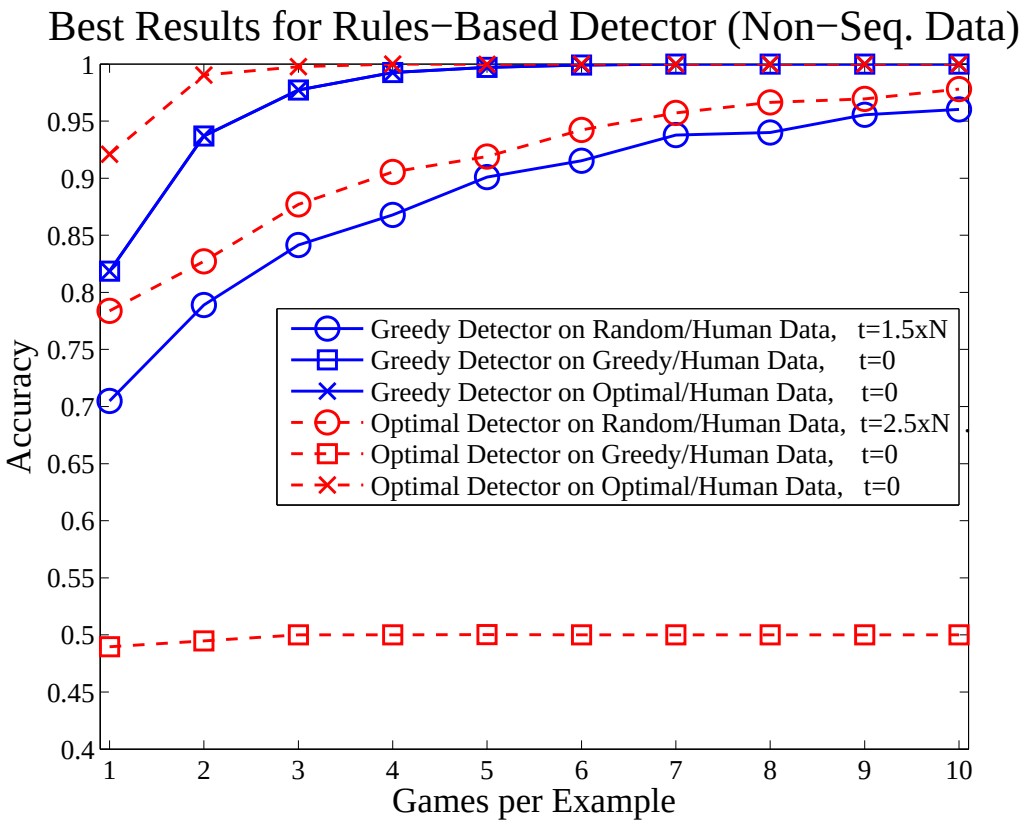


Figure 8.3. Detection accuracy results for Greedy rule-based detector using the best threshold for each case on test sets containing 100 examples of computer-generated gameplay and 100 examples of non-sequential human gameplay.

agent, since $Optimal = Optimal$. In the experiments, the threshold value is varied from a value of $t = 0$ to a value of $t = 4 \times N$, where N is the number of games per example. Because the gameplay under consideration happens to be from Tic-tac-toe, the value $t = 4 \times N$ indicates that the decision threshold is placed at the point where half the chosen moves in the example violate the rule set.

Figure 8.3 shows the detection accuracy results of the experiment using the Greedy rules-based and Optimal rules-based detectors using the best threshold for each case. Each test set contains 100 examples of human play and 100 examples of computer play, where each example contains $1 \leq N \leq 10$ games. The human-generated games are sampled from the database containing non-sequential human games. From the fig-

ure, it can be seen that the Greedy rules-based detector can achieve greater than 95% accuracy in all three cases. The Optimal rules-based detector can also achieve greater than 95% accuracy in distinguishing between Human and Random, and between Human and Optimal. However, it can do no better than guessing at distinguishing between Human and Greedy. The poor performance of the Optimal rules-based detector against the Greedy agent indicates that non-sequential human gameplay is very close to Greedy, but not close to Optimal. The perfect overlap of the Greedy detector's results against Greedy and Optimal agents is a consequence of the fact that $Optimal \subset Greedy$, and so the Greedy detector cannot distinguish Optimal gameplay from Greedy gameplay because both always make a greedy move.

Figure 8.4 shows the detection accuracy results of the experiment using the Greedy rules-based and Optimal rules-based detectors using the best threshold for each case on the same three test sets as above, except that now the human data is from the database of human games played sequentially. From the figure, it can be seen that the Greedy rules-based detector can achieve greater than 95% accuracy in distinguishing between human and computer in all three cases, just as with the non-sequential human data. However, unlike with the non-sequential data, the Optimal rules-based detector can now achieve greater than 95% accuracy in distinguishing between human and computer in all three cases.

The improvement in detector performance between non-sequential data and sequential data suggests that humans play more optimally, but not perfectly optimal, when they have seen the gameplay history leading to the current board state. Lucky for Wendy, then, since sequentially is the most natural way to play a game, and so sequential gameplay data should be easy to collect and will, at the same time, allow Wendy to achieve high accuracy against all three computer agents. One important point to note, however, is that no one threshold value is able to accurately distinguish between human gameplay and all three computer agents. This is nothing to worry about if Wendy knows what computer agent Alice and Bob are using when they use the covert channel. If Wendy does not initially know which agent they are using, she

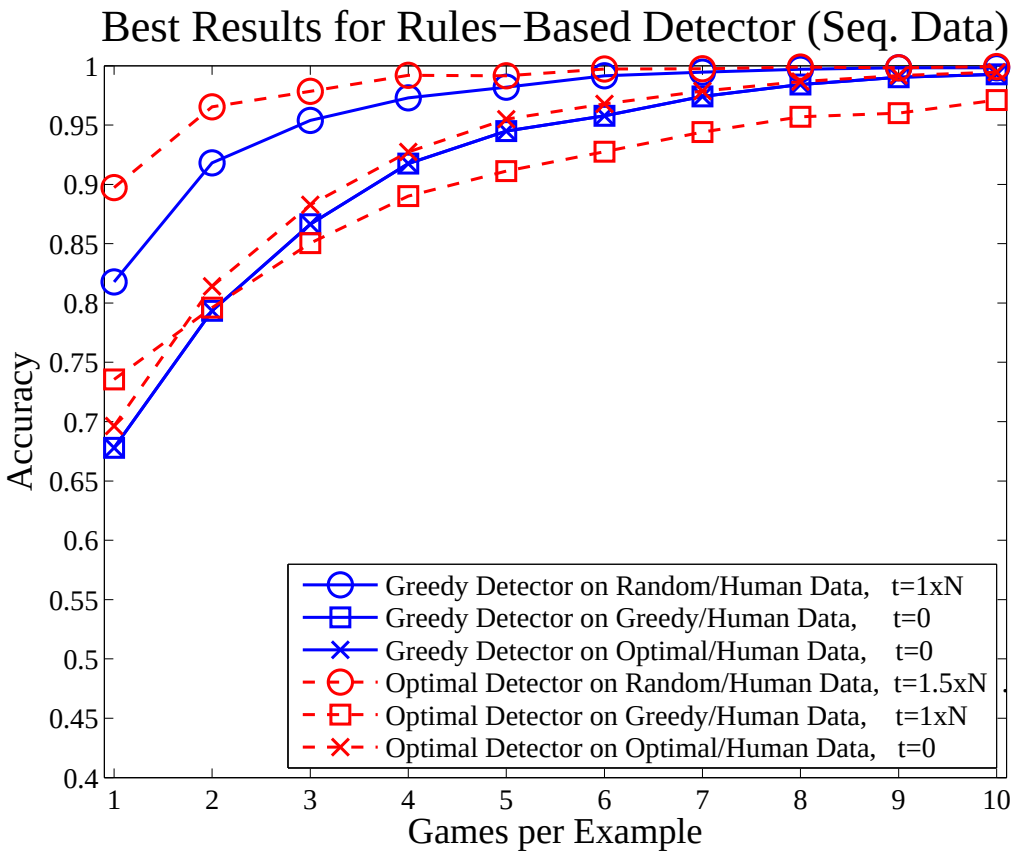


Figure 8.4. Detection accuracy results for Greedy rule-based detector using the best threshold for each case on test sets containing 100 examples of computer-generated gameplay and 100 examples of sequential human gameplay.

may be able to determine this by observing their play for a while. However, during this time, they could be using the channel, or they could be purposefully playing a certain way in order to sabotage Wendy's surveillance data, e.g. they could play with a Random strategy initially to convince Wendy that their computer agent is Random, and then switch to Greedy after a while.

8.2.2 Feature-Based Detection

The next detector evaluated was a feature-based detector. Feature-based detectors utilize attributes of the cover-media objects and machine learning algorithms to classify objects as clean or dirty. In these tests, two classifiers are analyzed. Both classifiers use two features: the Greedyness and the Optimalness of the play. Greedyness measures the proportion of observed moves that abide by the Greedy strategy. Optimalness is the same idea, but for the Optimal strategy. The general notion of Strategyness, of which Greedyness and Optimalness are special cases, can be defined as:

$$\eta_S = \frac{1}{|G|} \sum_{\{m,b\} \in G} 1_{\{\{m,b\} \in S\}} \tag{8.2}$$

where G is the sequence of {move, board} pairs that make up the gameplay example, and $\{m, b\} \in S$ means that the move m is in the set of good moves on board b identified by the strategy S. The choice of these two features, Greedyness and Optimalness, stems from our review of the cognitive psychology of human gameplaying. There, it was learned that humans do not play optimally, but rather with maximizing, or greedy, strategies. By measuring these two features of gameplay, it is hoped that human gameplay will be found to exhibit a unique balance between the two which can be distinguished from gameplay generated by pure Greedy or pure Optimal strategies.

The first classifier tested was a 1-nearest neighbor classifier. During training, the algorithm learns a representative for each of the four agents (Human, Random, Greedy, and Optimal) by finding the mean Greedyness and mean Optimalness of

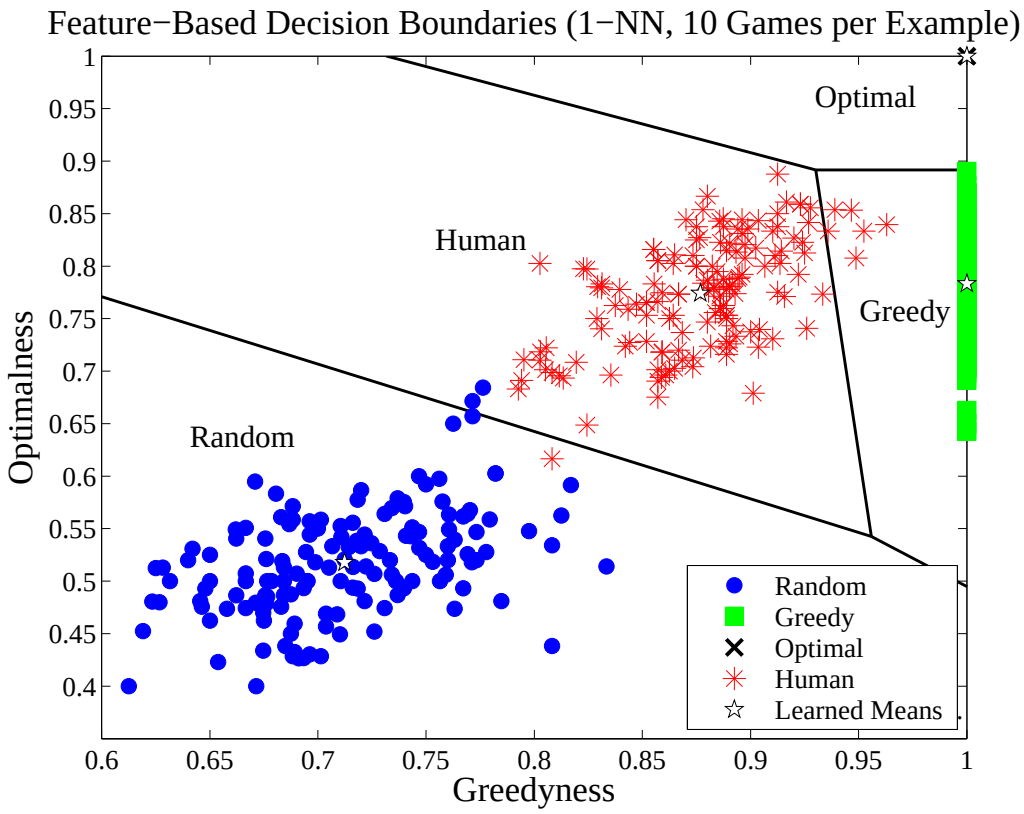


Figure 8.5. Learned decision boundaries for {Greedyness, Optimalness} feature-based 1-NN classifier using ten games per example. Example data points of gameplay from humans and computer agents are shown. The human data is from the non-sequential human dataset.

their play. Classification is carried out by finding the nearest representative to a given example. If the nearest representative is Human, then the example is classified as clean, otherwise it is classified as dirty. The learned classifier is tested using data which was held out from the training set. Ten-fold cross validation is used to obtain a more smooth result curve.

Figure 8.5 shows example learned decision boundaries for the 1-NN classifier when using ten games per example. Human gameplay is approximately 87% Greedy and 77% Optimal. Random gameplay is approximately 72% Greedy and 52% Optimal. Greedy gameplay is 100% Greedy and 80% Optimal. Optimal gameplay is 100% Greedy and 100% Optimal. The accuracy of the feature-based 1-NN detector on the

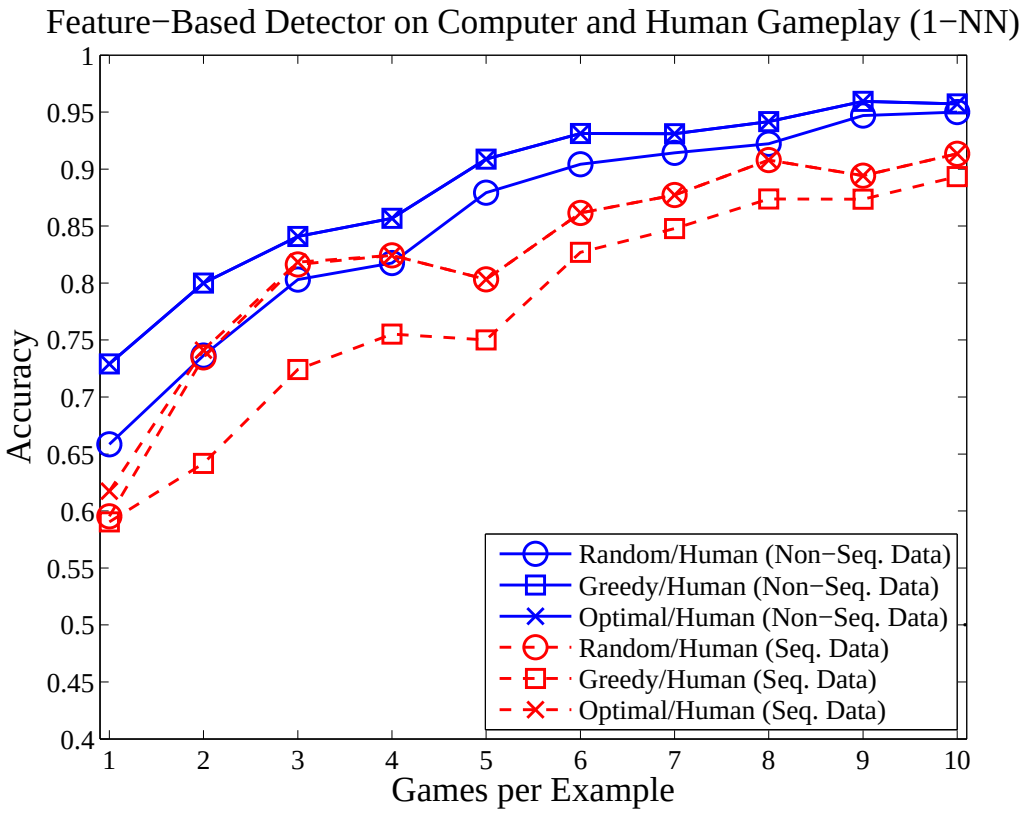


Figure 8.6. Detection accuracy results for {Greedyness, Optimalness} feature-based 1-NN detector on test sets containing an equal number of examples of human and computer gameplay.

task of distinguishing between human-generated gameplay and computer-generated gameplay for both non-sequential and sequential gameplay is shown in Figure 8.6. As the number of games per example increases, the accuracy of the detector increases for all cases. Once there are ten games per example, the detector's accuracy is approximately 95% on the non-sequential gameplay data, and close to 90% on the sequential gameplay data.

The second classifier tested was a binary decision tree classifier. During training, the algorithm learns a set of decision nodes, arranged in a tree structure. Beginning with a single node which contains all the training data, nodes are split in such a way that the Gini's Diversity Index (gdi) of the children is minimal. Nodes are split until both children have $gdi = 0$. The gdi is defined as

$$gdi = 1 - \sum_{c} p(c)^2 \tag{8.3}$$

where the sum is over the classes c at the node and $p(c)$ is the observed fraction of examples with class c present at that node. When all the examples at a node have the same class, gdi will be 0, otherwise, $gdi > 0$. Nodes with $gdi = 0$ are called *pure* nodes. Thus, gdi is a measure of impurity.

To classify an example, the algorithm follows the branches of the tree until a leaf node is reached. The label of that leaf node is selected as the predicted label of the example. The learned decision tree is tested using data which was held out from the training set. As before, ten-fold cross validation is used to obtain a more smooth result curve.

Figure 8.7 shows example learned decision boundaries for the binary decision tree classifier when using ten games per example. The first split is on the Greedyness value, greater than 98% is classified as dirty, less than 98% goes to the second decision node. The second decision node splits on the Optimalness value, less than 61% is classified as dirty, greater than 61% goes to the third decision node. The third, and final, decision node splits on the Greedyness value, greater than 77% is classified as clean, less than 77% is classified as dirty. The accuracy of the feature-based decision

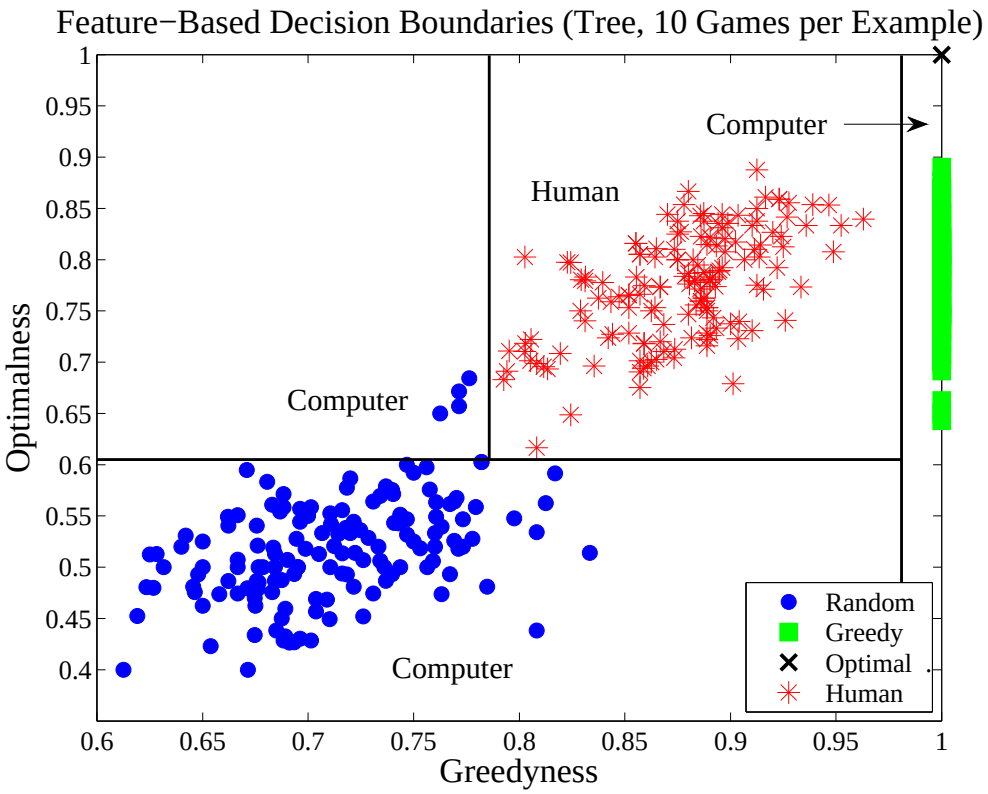


Figure 8.7. Learned decision boundaries for {Greedyness, Optimalness} feature-based tree classifier using ten games per example. Example data points of gameplay from humans and computer agents are shown. The human data is from the non-sequential human dataset.

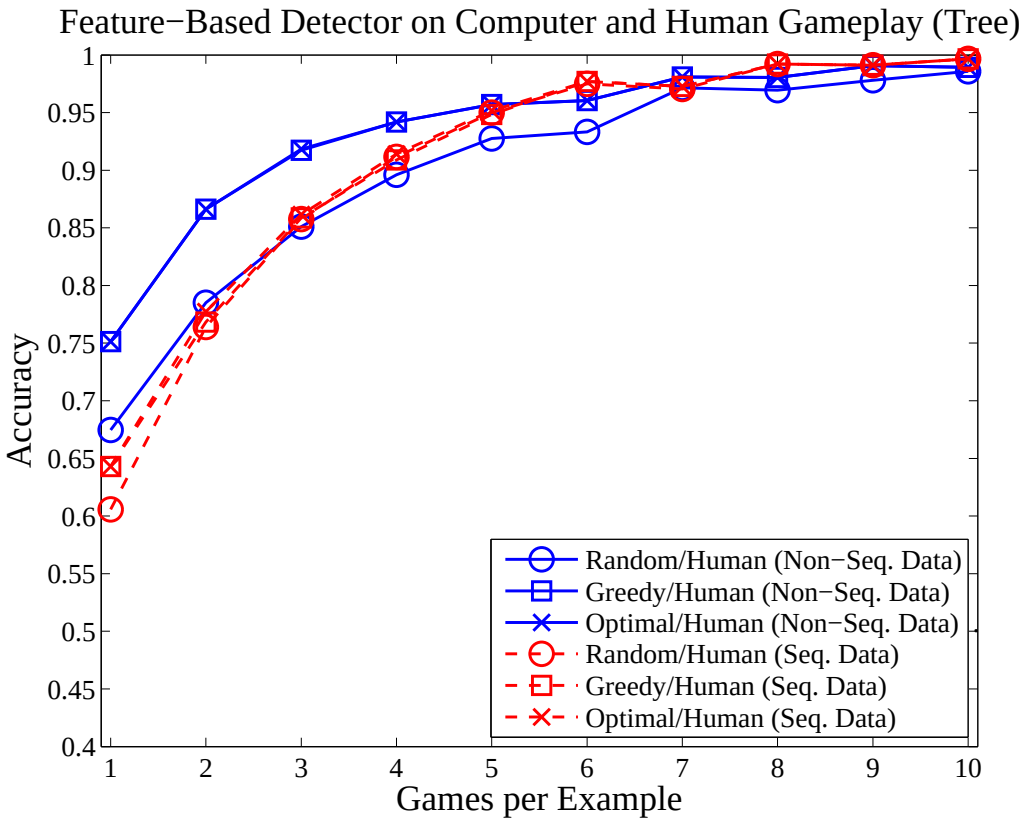


Figure 8.8. Detection accuracy results for {Greedyness, Optimalness} feature-based detector on test sets containing an equal number of examples of human and computer gameplay.

tree detector on the task of distinguishing between human-generated gameplay and computer-generated gameplay for both non-sequential and sequential gameplay is shown in Figure 8.8. As the number of games per example increases, the accuracy of the detector increases for all cases. Once there are ten games per example, the detector's accuracy is greater than 97% on both the non-sequential and sequential gameplay data.

Using the feature-based detector, Wendy does not need to know which computer agent Alice and Bob are using for their covert communication in order to achieve high accuracy in detecting their channel usage. This is an improvement over the rules-based detector which required Wendy to know which computer agent is being used in order to achieve high accuracy. However, if Wendy does know which agent they are using, then the rules-based approach can achieve higher accuracy than the feature-based approach with fewer games per example. This means Wendy can make her decision sooner and therefore limits the size of the secrets that Alice and Bob can communicate.

8.2.3 Probabilistic Model-Based Detection

The third detector evaluated was a probabilistic model-based detector. A probabilistic model-based detector uses probability distributions to do classification. To classify an example, the detector uses the observed gameplay in the example to compute the likelihood of the generator being human and the likelihood of the generator being a computer agent. The detector classifies each example based on the more likely generator, gameplay which is more likely to be human-generated is clean while gameplay which is more likely to be computer-generated is dirty. That is, the detector computes

$$\rho_i = \frac{P(Human \mid Data_i)}{P(Computer \mid Data_i)} \tag{8.4}$$

$$\rho_i = \frac{\left(\frac{P(Data_i \mid Human)P(Human)}{P(Data_i)}\right)}{\left(\frac{P(Data_i \mid Computer)P(Computer)}{P(Data_i)}\right)} \tag{8.5}$$

$$\rho_i = \frac{P(Data_i \mid Human)}{P(Data_i \mid Computer)} \tag{8.6}$$

where the $P(Data_i)$ values cancel each other out and $P(Human) = P(Computer) = 0.5$ also cancel each other out. The reason $P(Human) = P(Computer) = 0.5$ is that the test set contains half human-generated examples and half computer-generated examples, thus the prior for both classes is 0.5. The classifier which the detector uses is then simply

$$c_i = \begin{cases} 0 & : \rho_i \geq \tau \\ 1 & : \rho_i < \tau \end{cases} \tag{8.7}$$

where τ is the learned decision threshold, $c_i = 0$ indicates that the example is classified as clean and $c_i = 1$ indicates the example is classified as dirty.

The likelihood $P(Data_i \mid Source)$ that an example of gameplay $Data_i$ was generated by a source $Source$ is computed as the product over all moves in the example of the relative frequency of the move chosen by the source, given the current boardstate. For a human source, this relative frequency is the number of times a human, when presented with the current boardstate, selected that particular move divided by the total number of times that boardstate was presented:

$$P(Data_i \mid Human) = \prod_{\{m,b\} \in Data_i} \frac{N_{m,b}}{N_{board}} \tag{8.8}$$

where $N_{m,b}$ is the number of times a human selected move m when presented with the board b and N_b is the total number of times board b was presented.

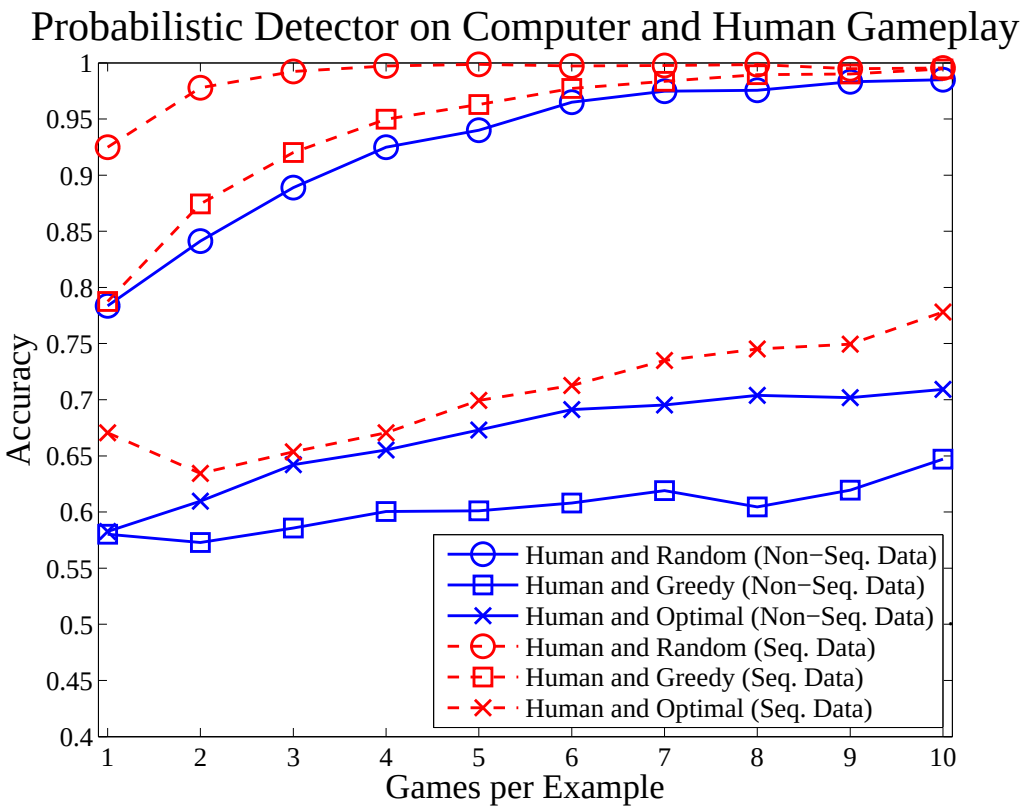


Figure 8.9. Detection accuracy results for probabilistic model-based detector on test sets containing an equal number of examples of human and computer gameplay.

For a computer agent, the required relative frequency is just the reciprocal of the size of the set of good moves for the current boardstate because each good move is equally likely to be chosen:

$$P(Data_i \mid Computer) = \prod_{\{m,b\} \in Data_i} \frac{1}{|\{x : \{x, b\} \in Computer\}|} \tag{8.9}$$

where $\{x, b\} \in Comp.$ means that move x is in the set of good moves on board b identified by the computer agent, $Computer \in \{Random, Greedy, Optimal\}$.

Figure 8.9 shows the results of the experiment using the probabilistic model-based detector to distinguish between human-generated gameplay and computer-generated gameplay for both sources of human data. Ten-fold cross validation is used to smooth

the result curve. The decision threshold τ is learned during the training phase on the nine folds of data set aside for training. The learned value for τ is the threshold which maximizes the classification accuracy on the training data. The classifier is then tested on the fold which was not used during training using the learned decision threshold τ. The results from the ten folds are averaged together to obtain the data shown in the figure. As was seen in the analysis of the rules-based detector, the use of sequentially-collected gameplay data improves Wendy's decision accuracy over the use of non-sequential data. Wendy's highest accuracy is achieved when distinguishing between Random and sequential Human data, which is greater than 95% for two games per example and nearly 100% for at lest four games per example. She also achieves high accuracy when distinguishing between Greedy and sequential Human data, as well as between Random and non-sequential Human data. In both cases, her accuracy is greater than 95% when she observes at least 6 games per example.

The probabilistic-based detector is not as accurate as the feature-based detector. In addition to decreased accuracy against Greedy and Optimal agents and a larger disparity in accuracy between sequential and non-sequential Human data, the probabilistic-based detector requires Wendy to make an assumption about which computer agent Alice and Bob are using. If Wendy guesses wrong, or Alice and Bob manage to trick her, Wendy will not be able to accurately detect the usage of the covert channel. On the other hand, if Wendy does know which computer agent Alice and Bob are using, she can achieve the same or better accuracy, with fewer games per example, by using the rules-based detector. Overall, of the three detectors tested, the probabilistic detector performed the worst against the rules-based stego-agents.

8.3 Conclusion

Experiments were conducted to analyze three types of detectors: rules-based, feature-based, and probabilistic-based. The probabilistic-based detector performed the worst. In order to use it, Wendy must know which computer agent Alice and

Bob will use to covertly communicate. However, if she knows this, then she can use the rules-based detector to achieve the same or higher accuracy. The rules-based detector was accurate in specific cases, but was not able to distinguish between human gameplay and computer gameplay using any single set of parameters. If Wendy does not know which computer agent Alice and Bob are using, then the feature-based detector performs best. Using the Greedyness and Optimalness features, it was able to take advantage of both dimensions to distinguish between human and computer gameplay with greater than 95% accuracy when shown at least six games per example, and greater than 97% accuracy when shown ten games per example, regardless of the computer agent or data collection style. However, if Wendy does know which computer agent Alice and Bob are using, then she can achieve the highest accuracy, using the fewest games per example, by using the rules-based detector. Requiring fewer games to make an accurate decision allows Wendy to limit the amount of secret data that Alice and Bob can share.

The analysis of the feature-based detector suggests that some improvements could be made to the steganographic method so that the computer agents produce play which more closely mimics that of humans along the given dimensions. For example, a mostly-optimal agent which makes a non-optimal move 30% of the time might be able to generate gameplay which is inseparable from authentic human gameplay. Also, the analysis used rules-based agents, against which the rules-based detector was the best. It is possible the probabilistic-based detector will perform better against probabilistic-based agents, which might use a Markov chain to mimic a specific player's style. The question is then whether Wendy's Markov chain can be more accurate than Alice's. The effect of the data collection method on detection accuracy suggests that there is a difference is gameplay between the two methods. Lucky for Wendy, the most natural way to collect data, sequentially, is also the method which yields the best accuracy results. While non-sequential gameplay data collection may not be as useful for detection, it may still be useful for understanding how humans play games. It is our hope that future work in the area of steganography in games will take into

account the cognitive psychology of human gameplaying when designing techniques for generating and detecting steganography in games. This back and forth progress (steganalysts achieve better detection, steganographers use the advance to defeat the steganalysis) is ultimately good for the field as it continually pushes both sides to develop better techniques.

9 STASI: A SOFTWARE TESTBED FOR ANALYZING STEGANOGRAPHIC INTERACTIONS

While cryptography and steganography are both broadly applicable to security and privacy, the field of steganography is not nearly as well studied as cryptography. This is due, in part, to the mathematical foundations of cryptography which allow researchers to analyze cryptographic algorithms *in vitro*, or on paper, as it were. In the case of steganography, because the goal is to conceal the existence of the message, rather than the contents, analysis can not generally be done on paper but must be done *in vivo*. That is, the analysis depends on factors external to the algorithm, such as properties of the communication medium and the cover-objects. For this reason, the study of steganography is more experimental in nature than that of cryptography.

While there are many pieces of software that implement steganographic methods, there are very few software systems for doing experiments with steganographic and steganalytic methods. Despite the enormous amount of work on image steganography, the authors are aware of only one software system for testing image steganography and steganalysis techniques: Virtual Steganographic Laboratory developed by Forczmanski and Wegrzyn. Virtual Steganographic Laboratory (VSL) is a graphical block diagramming tool that provides a framework for using and testing methods for image steganography and steganalysis. VSL utilizes a graphical user interface and a modular, plug-in architecture [67,94]. Aside from VSL, researchers working on information hiding in images do not have a standardized framework in which to conduct their research into new steganographic methods or techniques for analysis and detection. Moreover, information hiding in images is only a small glimpse of what is possible with steganography.

In this chapter, we present STASI, the Software Testbed for Analyzing Steganographic Interactions. STASI provides a framework for implementing and analyzing

steganographic systems that exploit structured interactions for covert communication. In Section 9.1, we discuss the design specifications for STASI. In Section architecture, we present the software architecture of STASI and discuss the implementation of the testbed. Section 9.3 presents the results of some experimental analysis of STASI, along with example results of analyses conducted using STASI. Finally, we conclude in Section 9.4 with a summary of STASI.

9.1 Design Specifications for STASI

STASI is built to meet the following design specifications for a steganographic testbed:

Extensible. Users should be able to add new interactions and new agents for interactions without needing to see or touch the core code, but they should be able to do so if they want, see *open source*.

Easy to use. The user experience should be simple and clean. Menus should be clear, documentation should be extensive but concise. Tools and operations should work as expected, without strange methods that are peculiar to this particular piece of software. Errors should be handled gracefully and informatively, prescribing the solution whenever possible.

Customizable. The testbed should be able to take advantage of local preferences and data to provide the user with a custom experience. Data, agents, and rules for interactions can be stored and accessed both locally and remotely, as files or as records in a database.

Efficient. Testbed operations should be time and space efficient. If the operation will take much time or space, the user should be warned, given an estimate of the runtime of the operation, and provided the opportunity to cancel or augment the operation. Operations should be multi-threaded whenever possible.

Open Source. The testbed source code should be open and free. However, a canonical public repository should be maintained that includes significant useful

additions to the core functionality. Repositories of data, agents, and interactions should be maintained and made freely available to all users of the testbed.

9.2 Architecture

The software architecture of STASI provides a highly flexible, customizable, and extensible framework for studying steganography in interactions through a modular interface with a central core of functionality. In addition to the core, the testbed consists of three types of components, each of which can be extended and adapted to suit user needs: interaction specifications, data-generating agents, and analysis tools. All of these components are brought together into one, easy-to-use, graphical user interface. In this section, we discuss the design and implementation of each of these components.

9.2.1 Core Functions

In the testbed, structured interactions use a metaphor of turn-based multiplayer online gaming. An interaction is modeled as a game between one or more player agents, which is managed by a mediator between the agents. The mediator, or game manager, maintains and provides to the players information about the state of the interaction. The game manager also logs gameplay data to be used later for analysis. The game manager is one of the many core functions the testbed provides. The testbed also provides MySQL database connectivity, game configuration management, gameplay data logging and parsing, and a machine learning library as part of the core functionality of the testbed.

9.2.2 Interaction Specifications

An interaction specification formally defines the rules and environment of the game. It may also, optionally, provide a graphical interface to display the state of the game to the testbed user and enable human-in-the-loop agents to receive input.

The rules and environment of the game are codified by a game board which maintains the current state of the game, handles updates to the state of the game from moves made by players, provides information to the players about legal next moves, and detects when some agent has met the win condition of the game or when the game has ended in a draw or loss for all agents. The game manager maintains the game board for each game and makes it available to the agents playing the game.

As an example of an interaction specification, take a simple paper-and-pencil game such as Tic-tac-toe. Tic-tac-toe is a two-player game played on a three-by-three grid onto which the players take turns placing a mark, usually 'X' for the first player and 'O' for the second player, until a player succeeds in placing three of their marks in a row, column, or diagonal. The first player to do so is the winner. If, after the ninth move, neither player has succeeded in getting three of their marks in a line, the game ends in a draw since there are no more available moves.

In STASI, the interaction specification for Tic-tac-toe consists of a game board which stores the location of each mark on three-by-three grid of cells as well as the last move made. The game board can compute, for the current state of the grid, the set of all legal next moves. A legal move in Tic-tac-toe is the location of an empty grid cell. Given a legal move, the game board can update the state of the grid to include the player's mark ('X' or 'O') in the cell chosen by the player. The game board can check the win condition for the current state of the board by looking for three of the same symbol in a row column or diagonal. Using the game board for Tic-tac-toe, the game manager can step two agents through the game until it ends. The agents can get the list of all legal moves from the board and choose one to make. The game manager will apply the chosen move to the board, which will update the state of the

game. If the board is a win condition, then the game manager can communicate to the agents that the game has ended and which agent was the winner.

Consider another example of an interaction which can be represented in STASI as a game: the Transmission Control Protocol (TCP) handshake. Before data can be sent over the network, the endpoints must establish a connection so that the data can be reliably transmitted. The TCP handshake is a 3-step protocol that establishes a network connection between two programs. In the TCP handshake game, the players represent programs connected to a network. The game board is a finite state diagram which maintains the current state of the connection. Initially, the connection is closed and both players are listening for incoming SYN packets. At some point, one of them will choose the move which corresponds to sending a SYN packet and start the handshake. Upon receiving the SYN packet (learning from the game manager or game board that the last move was a SYN move), the other player should select the move which corresponds to sending back a SYN-ACK, acknowledging the SYN from the other player and setting up a connection in the other direction. When the SYN-ACK is received, an ACK should be sent back to finalize the establishment of the connection. On any of their turns in the game, each player must choose to take any legal action. The rules of the game can be relaxed to allow players to choose do-nothing operations, simulating transmission delays. Also, players may be allowed to resend packets, or send packets out of order. The game ends after a correct sequence of SYN / SYN-ACK / ACK messages are sent. The game manager alternates between players, updating each in turn about the state of the game and asking for their next move. When the game board enters a winning state (the connection established state), the game manager communicates to the agents that the game has ended (perhaps in a draw, or with both agents winning).

By implementing interaction specifications as game boards and providing a general game manager, STASI can be used to study any interaction which can be effectively described as a game between one or more players. The game manager is agnostic to the rules of the game and needs only to maintain the game board and communicate

with the agents to get each agent's next move. The game board can make games which are not turn-based seem to the game manager as if they were by not updating the state of the game (except the current player) until all players have selected a move. This mechanism can also handle skipping players, or letting players move more than once.

9.2.3 Data Generating Agents

One of the main applications of STASI is to generate synthetic gameplay data to use for analysis. Gameplay data is generated by players in a game. Starting from the initial state of the board, each move made by the players is recorded. The sequence of moves made by the players in the game is called the game's history. Given a game's history, the entire game can be replayed and the moves of each player analyzed individually or as a whole.

In STASI, the players in a game are modeled as decision-making agents. An agent receives from the game manager a copy of the game board and is asked to respond with the agent's next move in the game. Aside from physical constraints (such as memory or time), there is no restriction on the method by which an agent may make its decision. The simplest agent is one which makes a uniformly random selection of next move from the list of all legal next moves. A more complex agent may attempt to find a set of best moves from which to make its choice, such as by executing a minimax search of the game tree rooted at the current game state. Each move generated by an agent is recorded in the history of the game. The histories are stored in memory and can be used for analysis immediately or can be written to a file or database to be reloaded and analyzed later.

When generating data with STASI, the user first loads the game specification, then loads and selects the agents which will play the game, and then specifies the number of times the agents will play the game. The testbed will automatically divide that number of games into several smaller sets and have multiple threads simulate

the games in parallel. After the games have completed, the results are combined and the gameplay data added to the existing set of data already loaded in memory, all of which is made immediately available for analysis. When written to a file, the data is stored in CSV format. The data can also be written to a remote or local MySQL database.

9.2.4 Adding New Interactions to STASI

Due to STASI's modular design, it is easy to add new specifications for structured interactions without needing to change any of the testbed core code. As mentioned above, an interaction specification is a game board and an optional user interface.

In STASI, game boards for interactions are subclasses of an abstract `GameBoard` class. The abstract `GameBoard` class defines a set of methods that every game board must implement. These methods are described below.

Game Board Methods

- `availableMoves()` returns an array of available next moves for the current player given the current state of the interaction. The array should never be empty. If the game should continue even if a player has no available moves, that player should actually have only the single move *NOP*, signifying that the agent does nothing except wait until its next turn.

- `getMoveHistory()` returns the list of moves made so far in the interaction. Moves made in interactions are visible to all agents, so this method allows agents to access that history without needing to explicitly store it themselves.

- `lastMove()` returns the last move made on the board. This information is also available as the last element of the move history, however providing direct access to the last move allows agents to see it without having to get the entire move history, which may be large.

- `moveToString(Object move)` converts a move object into a String so that the move can be logged easily. This method, in part, allows users to define the specific encoding to use when representing moves as Strings.

- `playMove(Object move, int player)` applies the specified move object to the game board on behalf of the specified player. Specifying the player allows agents to more easily use game boards to explore the space of potential future states. Only the game manager will have direct access to the actual game board, the agents should will only have access to copies.

- `playMoveAsString(String move, int player)` applies the specified move, given as a String, to the game board on behalf on the specified player. This method should behave exactly as `playMove`. Typically, a game board will implement this method so that it parses the String into a valid move object and then calls the `playMove` method on that object.

- `queryBoard(Object location)` returns the contents of the specified location on the game board. Locations are not necessarily spatial locations, but rather some coordinates that signify a particular area in which play is possible. For example, an interaction may have a common area where moves may be made (such as paying money to a central bank) but which has no spatial location.

- `reset()` resets the board to the initial state, before any moves have been made, ready for the first player to make their first move.

- `win()` determines whether the board is in a winning state and returns the identity of the winner, if any. The return value 0 is used to signify that a game is incomplete, having not yet reached wither a win or draw condition. The return values 1 through n are used to indicate that player i, for $i \in [1, n]$, is the winner. The return value $n + 1$ indicates that the game has ended in a draw. Since games are user-definable, the particular meanings of return values will be specified by the author of the game board.

The optional user interface for an interaction is a subclass of an abstract `GamePanel` class, which defines a set of methods that a user interface for an interaction must implement. These methods are described below.

Game Panel Methods

- `getButtonID(JComponent button)` determines which button in the game panel was clicked. When an agent is using a human as part of it's decision-making process, this method allows the testbed to translate the user's action of clicking a button into a move selected for the interaction.

- `has(JComponent comp)` determines whether this game panel has the specified component. While the user is interacting with the testbed, any events generated must be traced to determine from which particular componenent the event was generated. This method allows the testbed to determine if the event was generated by the game panel. Such events can then be directed to the game panel for further handling.

- `refresh(GameBoard gameBoard)` updates the graphical representation of the game state using the provided game board. After a move is made, the panel can display this, and other effects of the move, by using the updated game board to redraw the game panel. It is also possible that components of the game panel may be added, removed, or modified as part of that update.

- `setEnable(boolean state)` is used to enable an disable the game panel. An enabled game panel is one which the user can interact with, such as by clicking buttons or entering text into fields. A disabled game panel does not permit user interaction. A game panel may be disabled during the opponent's turns or when the game ends.

If a user interface is not provided, the testbed will not attempt to render it and will instead display a blank panel where there user interface would have been. Game panels, as these user interfaces for interactions are called in STASI, allow game state

information to be displayed as well as providing a way to receive information from the user (e.g. a human-in-the-loop agent would need input from a human in order to play the game, the game panel could provide the human player with information about the board and supply a convenient and intuitive way to choose a next move). Since the game panel is graphical, rendering it takes time, and for that reason it is optional. The testbed even includes an option to disable rendering of the game panel. Without the game panel to render, the testbed can run experiments much faster.

The first time a new interaction is loaded in STASI, in addition to the game board and the game panel, the testbed will need to know where the gameplay data for the interaction is to be stored. The data can be stored in a file or in a MySQL database, hosted locally or remotely. All of this information will be saved in a configuration file, which the user can name. The configuration will be available the next time an interaction is loaded, so the user need only provide this information once. Multiple configurations can be created, allowing the user to have flexibility in how they use the testbed. For example, a user may wish to have data for the same game stored in two places, one for each of two different game panels. The configurations make this possible by allowing the user to define two configuration which both use the same game board class, but have two different game panel classes and store data in two different places. As another example, the user may wish to have one configuration which stores data locally, and one which stores data remotely, so that the testbed can be used even when a network connection is not available. The testbed's core functionality provides a tool for loading data so that the user can load data from local storage and save it to a remote database, or vice versa.

This framework was designed to achieve maximum compatibility and flexibility without the need to change the testbed itself. An interaction's game board is used to retain the state of the game board and to communicate between game panel and the game-playing agents. The Testbed itself does not need to know what kind of game board it is and expects that the game panel and the agents can correctly handle that kind of game board. This is a reasonable assumption, because when the user adds

a new game, the implementation of the game components should be consistent with each other. Such consistency provides the benefit that should the game panel or agents need a specific feature which is not provided through the methods prescribed by the abstract game board class, the game board can be downcasted to that specific type of game board. In other words, agents are only expected to know how to use the methods defined in the abstract game board class, but may optionally use methods provided by the specific implementation of the game board for the interaction in which they will participate. The Testbed does not use any of these specific features which are defined by specific game boards and is able to handle all game boards as if they were all the same. In this way, the testbed achieves both good compatibility, allowing a single game manager to handle any properly implemented interaction, and good flexibility, allowing users to define and implement extra functions for game boards to provide to agents.

9.2.5 Adding New Agents to STASI

As with adding new interactions to STASI, adding new agents for interactions does not require any code to be changed in the testbed core. Once the game board for the interaction has been defined, implementing an agent is very simple. Following the game metaphor, an agent for an interaction is called a game agent. Similar to game boards, game agents are subclasses of an abstract `GameAgent` class which defines the methods which every game agent must implement. These methods are described below.

- `finalize()` is called when a game ends and the game manager is about to dispose of the agent. This method will be the last one called on an agent externally. Any operations that an agent needs to do before the interaction completely ends can be completed in the finalize method. For example, an agent which uses a database to record information about the games it plays should do any final updates to the database and close the database connection

in the `finalize` method. Any initialization operations should be done in the class constructor.

- `nextStep(GameBoard gameBoard, int player)` is called by the game manager when it is the agent's turn to select a move. The game manager gives to the agent a copy of the game board and an integer representing the player order for this agent (usually this number is constant, however in some games it may change). The agent is expected to an array of two objects: the move selected and the set of moves from which the agent chose its move. The first element, the selected move, allows the game manager to update the game board using the agent's chosen move. The second element, the candidate set, is logged by the game manager for analysis.

- `toString()` returns the name of the agent as a String. It is used by the game manager to record the names of the agents playing the game. The agent's name is expected to be globally unique. For this reason, the canonical name is the suggested return value, however the user may assign any name to the agent class.

Internally, a game agent may have any other methods it needs in order to make its decision. Game agents may keep state, or they may be stateless. The particular design of an agent is immaterial to the game manager, which only uses the above three methods, plus the constructor, to interact with the agent. Since every agent has an identical interface, as does every game board, the game manager never needs to know which interaction is it managing and can use a standard algorithm to run every interaction.

Game agents can be loaded dynamically in STASI. One agent may be loaded to generate some data, and then a second agent can be loaded to generate even more, including data generated one one type of agent playing against another. Agents are also used for steganalysis, and can be loaded for that purpose at anytime. When an agent is loaded in a configured interaction, that agent is added to the configuration

file and will be preloaded the next time the configuration is used, saving the user time in the future.

9.2.6 Analysis Tools

In addition to generating data, another of STASI's main applications is analyzing gameplay data to measure the properties of steganography and steganalysis techniques for structured interactions. One of the most important properties of a steganographic system is the capacity.

The capacity of a stego-system is the number of bits of secret data that it can hide in a stego-object. For structured interactions, secret data is hidden in the choice of move at each step of the interaction so, the stego-objects are the histories of the interactions. Since every agent can see the actions of all other agents at every step, the covert channel created in the interaction by encoding bits as moves transmits bits in two directions for every agent: every move an agent, say Alice, makes can send bits to all the other agents, and the other agents can send bits to Alice through their choice of move. Since one agent's capacity is dependent on the types of agents against which she is playing, as well as on player order, the capacity is measured for all observed combinations of agents. For example, the capacity of a two-player game played by an agent of type A as first player and an agent of type B as second player will be computed separately from the capacity of the same game with a type B first player and a type A second player.

Definition 9.2.1 *The capacity $C_p(g)$ for a player p in a game g is the amount of information, measured in bits, that the player was able to transmit in the game.*

Let g be a game played by n agents and $m_i^p \in \mathbb{N}$ be the i-th move of player p in that game. Let $h_g = (m_1^1, m_1^2, \ldots, m_1^n, m_2^1, \ldots)$ be the history of game g and denote the sequence of moves leading from the beginning of the game up to and including the move before player p's i-th move as $h_g(p, i) = (m_1^1, m_1^2, \ldots, m_i^{p-1})$. Let $c : \mathbb{N}^* \to (\mathbb{N}^*)$

be the candidate function, which maps a sequence of moves to a list of available next moves.

Then, the capacity $C_p(g)$ for player p in game g is given by Equation 9.1.

$$C_p(g) = \sum_i \log_2(|c(h_g(p,i))|) \tag{9.1}$$

The testbed computes the mean and standard deviation of the capacity of the observed gameplay for each player type on both a per-game and per-move basis. On a per-game basis, the statistics are computed over a given set G of games (see Equation 9.2). On a per-move basis, the statistics are computed over all moves in the given set of games (see Equation 9.3). The results are reported in CSV format.

$$C_g(p,G) = \frac{1}{|G|} \sum_{g \in G} C_p(g) \tag{9.2}$$

$$C_m(p,G) = \frac{1}{\sum_{g \in G} |\{m_i^p \in h_g\}|} \sum_{g \in G} C_p(g) \tag{9.3}$$

Steganographic Security

The security of a stego-system is the error rate that a passive adversary has when attempting to distinguish between authentic objects which are not hiding data (e.g. games played by humans) and objects which have been modified or synthesized to hide data (e.g. games played by computer agents). The success of a stego-system to evade detection is the failure of steganalysis techniques to detect the stego-system in use. Many steganalysis techniques may be brought to bear on the problem of detecting stego-objects. In order to be secure, a stego-system's stego-objects must be able to evade all the steganalysis techniques that may be used against them. On the other hand, in order to be useful, a steganalysis technique need only have a low error rate against some stego-system.

The testbed will use an agent-based system to classify observed gameplay, or more generally the history of any given interaction, as either authentic or generated. The same agents which may be used for generating data may also be used to analyze data. Given a history of an interaction, each agent type can be queried to discover for each move whether that agent could have made that move. The ratio of the number of moves in the interaction which could have been made by agent A to the total number of moves made in the interaction is the *score* assigned to the history by agent A. The higher the score, the more likely it is that agent A was the one which generated the data. These scores may also be computed for each player, giving an indication of the likelihood that a particular player is agent A. Depending on the set of agents chosen to be used for classification, the scores will have different meanings. An agent which is a model of human gameplay will score highly those games which are likely to have been played by a human (or good model of a human), whereas an agent which takes a purely computational approach to gameplay (such as attempting to play optimally) will score highly those games which are likely to have been generated by software agent and thus not by a human.

To evaluate the security of a stego-system, the testbed will take two equally sized sets of games: one which has been played by actual humans and one which has been generated by software agents. The human-played games are clean games, since they cannot have any secret information hidden in them. The computer-generated games are dirty, since they could have been generated in such a way as to encode secret information. The equal number of games in both sets will mean that an accuracy of near 50% is equivalent to guesswork and an accuracy near 100% will be indicative of the detector having a true advantage over the stego-system. In addition to specifying the number of games in the sets, the testbed user may also specify the number of games per example. Grouping games together into examples, sets of games played by the same agents, allows the detector to get more fine grained scores for classification. However, it also means that the detector will need to let a certain number of games complete before it can make its decision. Since each move can be encoding data, the

delay in decision making will potentially allow many bits to be transferred before the channel can even be detected. Once the dataset is compiled and grouped into examples, the scores are computed, one for each agent type loaded for this purpose. Thus, each example will make up a single row in a table where the columns are the class label and the attribute values, each attribute being the score assigned to the gameplay in that example by the corresponding agent. For example, if an agent of type G and an agent of type O are loaded for analysis, the table will have three columns: class label (0 for clean, 1 for dirty), G score ($x \in [0, 1]$ for the proportion of moves which could have been made by a G-type agent), and O score ($x \in [0, 1]$ for the proportion of moves which could have been made by an O-type agent). This table may then be used as input for any standard machine learning algorithm.

The testbed will use JavaML to provide basic machine learning tools for classifying data. The standard algorithm used by STASI for this purpose will be decision trees with 10-fold cross validation. The data set is shuffled and divided into 10 smaller sets. Over the course of 10 folds, one subset is held out as the test set and the other nine sets are combined and used as the training set for the algorithm to learn the parameters of the model (for decision tree, these are the decisions at each node of the tree which iteratively split the data until the examples at each leaf node are all of the same class). Once the model is learned, it is applied to the test set and the accuracy achieved is recorded. The accuracies from each of the 10 folds are average together to give an estimate of the accuracy of the detector.

The agents used for detection need not be intended for use as agents for generating data. That is, when used for detection, an agent is just a model of some behavioral feature of gameplay. For example, a Tic-tac-toe agent may have a simple rule of always selecting the middle cell if it is available and playing randomly otherwise. This agent would be very bad at playing Tic-tac-toe and would not be a good model of human gameplay in general. However, it may capture a particular aspect of human gameplay, which is that humans may prefer the middle cell over other cells. Together with other agents that model particular aspects of human gameplaying behavior, such

agents can form part of a group of agents that, as a whole, are useful for distinguishing authentic gameplay from synthetic gameplay.

9.2.7 User Interface

The testbed has four information panels on a tabbed pane: a gameplay data panel, an agent list panel, a game panel and an output panel. The gameplay data panel displays the list of gameplay data records being processed in the current session. This includes games which have been loaded as well as games which have been generated. The agent list panel displays the name of every agent which has been loaded and is available for use in the currently loaded interaction. The game panel displays a graphical representation of the game board, according the the user-implemented game panel class for the interaction. The output panel displays messages to the user about the progress of various operations and is used to present the results of data analysis. The testbed supports multiple tabs so that the user can load data separately into different tabs, analyze them independently and compare them easily.

9.3 Experimental Results

Since STASI is the first of its kind, there are no other systems with which it can be compared to determine the level of improvement STASI provides. Therefore, in this section, some basic results about STASI's efficiency are presented to provide a benchmark against which to measure future advances, either by future versions of STASI or by other implementations of testbeds for structured interactions. Also presented here are some example analyses of the steganographic properties of structured interactions conducted using STASI. The experiments in this section were conducted on a machine with an Intel Core i7-2630QM processor running at 2.00 GHz with 6.00 GB of RAM.

9.3.1 Basic Timing Results

The first experiment measured the time it takes STASI to generate gameplay data. For each agent, the time in seconds spent simulating 10000 games was recorded and is shown in Table 9.1. The amount of time required to simulate a single game is affected by the complexity of the agent, as well as the complexity of the game board. For example, the complexity of a random agent is constant for all agents, since it does exactly the same thing for any game: get the list of available moves and pick one at random. The games of Chess and Go are of similar complexity in terms of state space, however random Chess games take longer to simulate than random Go games. This is due in part to the greater complexity of the Chess game board which must do a great deal more move validation than the Go game board. Other factors affecting the simulation time are the complexity of the moves and the length of the games. The DB agent for Tic-tac-toe is so slow because it is querying a database for every move and the time required for the round trips to and from the database dominate the time required to make and validate the moves.

The next experiment measured the time it takes STASI to analyze the steganographic capacity of a game. For each agent, the time in milliseconds spent estimating the capacity from an observed set of 10000 games was recorded and is shown in Table 9.2. STASI uses a time-memory trade-off to speed up analysis preferentially to generating data. When data is generated, STASI records the number of available moves at each step of the interaction. This extra information is stored along with the gameplay histories and can be used during analysis to directly compute the capacity of each move without having to query the agent. The time required to analyze the capacity of an interaction depends mainly on the length of the interaction. Chess and Go are long games and so take a few hundred milliseconds more to add up all the per-move capacities compared to short games like Tic-tac-toe and Sim.

Table 9.1.
Time required to simulate 10000 games.

Game	**Time** (seconds)
Chess (Random)	322.015
Connect4 (Random)	20.288
Connect4 (Greedy)	23.022
Go (Random)	24.591
Quarto (Random)	21.746
Rock-Paper-Scissors	17.379
Sim (Random)	17.746
Sim (Greedy)	18.719
Tic-tac-toe (Random)	16.039
Tic-tac-toe (Greedy)	18.070
Tic-tac-toe (Optimal)	21.543
Tic-tac-toe (DB)	49547.730

Table 9.2.
Time required to measure the capacity of 10000 games.

Game	**Time** (milliseconds)
Chess (Random)	1579
Connect4 (Greedy)	446
Connect4 (Random)	505
Go (Random)	975
Quarto (Random)	350
Rock-Paper-Scissors (Random)	106
Sim (Greedy)	158
Sim (Random)	147
Tic-tac-toe (Greedy)	201
Tic-tac-toe (Optimal)	206
Tic-tac-toe (Random)	167
Tic-tac-toe (DB)	275

9.3.2 Example of Analysis Using STASI

The first experiment conducted was to measure the steganographic capacity of several structured interactions for various agent types. When measuring the maximum expected capacity, the agent type should be Random. Random agents are free to choose any available move at each step and so have access to the maximum capacity for each move. Agents which limit the number of moves available at each step, such as those trying to produce better play by only selecting from moves which meet a certain quality requirement, will result in reduced capacity. Greedy agents try to win with their next move, or block their opponent's win, in that order. If they cannot win and their opponent is not about to win, they are free to move randomly. Optimal agents always make a globally optimal move.

In this experiment, each agent played 10000 games and the observed capacity per game was recorded. Table 9.3 gives a 99.999% confidence interval around the observed mean for each agent. Long and complex games, like Chess and Go, have very large capacity because each move is taken from a large candidate set and many moves are made over the course of a single game. Short and simple games, like Rock-Paper-Scissors and Tic-tac-toe, have small capacity because the number of available moves at each step is small and the game ends after only a few moves. It is interesting to note that optimal play (such as in Tic-tac-toe) does not eliminate steganographic capacity.

Another experiment was conducted to measure the steganographic security Tic-tac-toe by determining how well a feature-based detector can distinguish between authentic human-generated data and synthetic computer agent-generated data. In this experiment, 5000 examples of authentic human data and 5000 examples of agent-generated data are used as input to STASI's feature-based decision tree classifier and the accuracy achieved is reported in Table 9.4. The features used were the Greedyness and Optimalness of the observed gameplay. In agreement with our results from Chapter 8, the decision tree classifier is able to detect with high accuracy the differ-

Table 9.3.
Steganographic capacity of interactions.

Game	**Capacity** (bits per game)
Chess (Random)	869.7631 ± 5.0687
Connect4 (Random)	58.3141 ± 0.5939
Connect4 (Greedy)	56.3826 ± 0.7859
Go (Random)	817.8934 ± 11.7460
Quarto (Random)	81.5978 ± 0.3347
Rock-Paper-Scissors	4.7542 ± 0.0863
Sim (Random)	31.4589 ± 0.1780
Sim (Greedy)	34.9976 ± 0.0571
Tic-tac-toe (Random)	17.5516 ± 0.0473
Tic-tac-toe (Greedy)	12.5691 ± 0.0958
Tic-tac-toe (Optimal)	9.047 ± 0.0715
Tic-tac-toe (DB)	9.2395 ± 0.2127

Table 9.4.
Detection accuracy of feature-based detector on Tic-tac-toe.

Stego-Agent	**Accuracy**
Random	97.7317%
Greedy	97.3738%
Optimal	99.9517%
70pOptimal	65.9749%
DB	73.2605%

ence between human and computer-generated gameplay for the Random, Greedy, and Optimal agents. The 70pOptimal agent plays optimally only 70% of the time and, as predicted, is much more difficult to detect than the pure strategies. The DB agent uses a database of authentic human data to generate synthetic gameplay. The detector is only able to achieve moderate accuracy against synthetic data generated by this agent due to the close statistical similarity between the synthetic and authentic data.

9.4 Conclusion

Out of the box, STASI includes a number of common structured interactions and can generate example data and run basic analyses on that data. Basic analysis tasks include measuring covert channel capacity and measuring steganographic security. In STASI, stego-systems are implemented using software agents. An agent is a participant in a structured interaction and behaves according to a set of rules for deciding which action the agent should take for any given state of the interaction. For each defined interaction, STASI provides a basic agent which acts randomly. Advanced agents can be designed and implemented by users. STASI can load agents from local storage, or from a remote server. Example data can be loaded from local or remote data repositories, such as a MySQL database. Even definitions for structured inter-

actions can be loaded from local or remote storage. This gives STASI a high level of extensibility, since data, agents, and interactions can be sourced from remote storage, opening the door for users to share their work with others. STASI can generate example data using the agents and interaction definitions, but this data will all be synthetic. However, since STASI can load data from a file or a database, data collected from authentic interactions can be loaded in STASI for analysis. Having access to both authentic and synthetic data allows STASI to conduct experiments that determine whether, and how well, analysis techniques are able to distinguish between synthetic and authentic interactions. The basic analyses that STASI conducts are behavioral agent-based. So, writing a new analysis technique is as simple as writing a new agent. Of course, since STASI is open source, users can also add to STASI core functionality to include analysis techniques that are not behavioral-based.

10 SUMMARY

In this dissertation, we have considered the problem of synthetic steganography: generating and detecting covert channels in generated media. We developed, implemented, and analyzed several novel techniques for hiding data in generated media such as time series, binary trees, tiled images, Sudoku puzzles, as well as games and other structured interactions. We also designed and tested steganalysis techniques to detect the differences between authentic objects and synthetic objects, which could be hiding data. Recognizing that structured interactions represent a vast field of novel media for steganography, we designed and implemented a software testbed for analyzing steganographic interactions. We used the testbed to measure the covert channel capacity of several interactions as well as to study certain behavioral properties of human-generated gameplay. In this chapter, we summarize our conclusions and provide suggestions for future work.

10.1 Conclusions

In our work on synthetic steganography in time series, we showed that we could develop relatively sophisticated and practical secret-key stego-systems in a variety of applications including the kinds seen in financial markets. The stego-system's layers of complexity enabled us to hide information in an arbitrary series of values with independent control over information density, location and size of perturbation. The information density and size of perturbation both affect the final time series directly. However, the main effect of the choice of embedding location showed up in the complexity of the key required to recover the information hidden in the time series.

We presented a novel information hiding scheme which uses the structure of Huffman trees to encode secret data. We showed that the capacity c of the channel is related to the degree of the tree and the size of the symbol alphabet by the equation $c = \lfloor \log_2(\frac{D!}{B!}) \rfloor + \lfloor \log_2(D!) \rfloor \frac{m-(D-B)}{(D-1)}$, where m is the number of symbols, $D \geq 2$ is the degree of the tree and $B = (D-2) - ((m-2) mod (D-1))$ so that $(D-B)$ is the number of symbols merged at the deepest level of the tree. A binary Huffman tree has embedding capacity $m-1$ bits, where m is the number of symbols in the alphabet. The covert channel in the tree structure is robust against any and all modifications to the content of the cover-object. An active warden can reduce or eliminate the capacity of the channel by recoding the object using a different but equivalent tree. A passive warden can achieve perfect accuracy of detection in the case where the Huffman tree is generated using the empirical symbol distribution of the content. The use of a private *a priori* distribution to build the tree requires the warden to compare the tree to one built using the empirical symbol distribution and reduces her detection accuracy to guesswork. The cost to the steganographers for this increased security is not in capacity, but rather in compression ratio.

Our work on steganography with context-sensitive tiling systems showed that it is not generally possible to distinguish between synthetic objects which are hiding data and synthetic objects which are not hiding data. Standard image-steganalysis methods fail to detect the existence of hidden information in tiled images. A directed steganalysis technique designed specifically for the tiling system is able to distinguish clean images from dirty images with high accuracy only in certain cases where a weak parameter set is used. Several strong parameter sets were found against which the warden is not able to achieve better accuracy than guesswork. The commonality among these was the use of a shared secret key.

Our stego-system for hiding data in Sudoku puzzles exploits 105.5425 ± 0.0016 bits of capacity in Sudoku puzzles, five times as much capacity as the only other method known for hiding data in Sudoku puzzles. The theoretical embedding capacity of a Sudoku board is 72.5 bits. StegoDoku is able to hide an average of 70.0929 ± 0.1172

bits, or 96.7% of expected maximum capacity, by using Huffman coding to decode secret bits into the choice of number to place in the next cell. The best order to visit cells when constructing a StegoDoku board is in order of most-constrained first. This ordering results in the least backtracks, and therefore also the lowest decoding error rate. StegoDoku can hide an additional 32.4327 ± 0.1366 bits in the configuration of clues provided to solve the puzzle. Boards can be generated in 9.0560 ± 0.02970 milliseconds. Generating a complete StegoDoku puzzle requires 30.7368 ± 0.065 seconds, indicating that it is much more computationally difficult to generate a solvable clue configuration which encodes secret bits than it is to generate a solved board which encodes secret bits. Without the use of a secret key, StegoDoku boards can be distinguished from authentic boards with near perfect accuracy, although it may require seeing several, as many as ten or more, before making a decision. When a shared secret key is used, the Sudoku boards generated by StegoDoku are indistinguishable from authentic Sudoku boards. In addition to the secret key controlling the board and puzzle generation processed, secrets should be encrypted before hiding and short secrets should be padded with random bits.

The result that it is generally not possible to determine whether a synthetic object is hiding data was confirmed by our work on covert channels in combinatorial games. In that work, we demonstrated a general methodology for exploiting covert channels in games and showed that detection of the usage of the covert channel depended on the warden having a correct and accurate model of clean gameplay. When all gameplay was synthetic, but only a portion contained hidden data, the warden's accuracy was no better than guesswork. We also showed that enforcing a perfect play requirement was not sufficient to eliminate the covert channel capacity.

Our follow-up work on covert channels in games utilized gameplay data collected from humans to develop an accurate anomaly detector that could distinguish between authentic human-generated gameplay and synthetic computer-generated gameplay. We trained a decision tree using multiple behavioral features of both human-generated and computer-generated gameplay. The resulting tree was able to correctly

classify more than 97% of the test set examples. However, the results suggest that a gameplaying-agent could be designed that would generate gameplay which is sufficiently statistically similar to authentic human gameplay that the detector would be fooled and reduce its accuracy to guesswork. The only way for the warden to regain the advantage is to develop a more accurate model of authentic gameplay. This is equally true for the steganographers. We also found that the method of collecting gameplay data influenced the accuracy of the detector which uses it. Sequential gameplay data collection, recording games played from start to finish, yielded higher accuracy than non-sequential collection (recording data as game state and move pairs).

In response to the lack of a framework for studying steganography in games, we designed and implemented the first software testbed for analyzing steganographic interactions. The testbed is open source, extensible, customizable, efficient, and easy to use. The testbed provides basic functionality which enables users to analyze the steganographic properties of any structured interaction. Data can be generated by the testbed and stored for later use. Data can also be loaded from local or remote storage. We used the testbed to generate and analyze gameplay data for several games. For most agents, the testbed requires only a few seconds to generate 10000 examples of gameplay and can measure the average capacity of those examples in less than 1 second. Observed capacities for the games studied range from 4.7542 ± 0.0863 bits per game of Rock-Paper-Scissors to 869.7631 ± 5.0687 bits per game of Chess. The embedding capacity of a game decreases as the space of possible next moves becomes smaller. Agents which seek to mimic the play of humans will choose from a smaller set of potential next moves and therefore exploit less of than the maximum capacity that the game can support. A good estimate of Tic-tac-toe's maximum embedding capacity is 17.5516 ± 0.0473 bits per game. However, the agent which only selects moves that it has observed a human make has an embedding capacity of 9.2395 ± 0.2127 bits per game because humans rule out certain moves, such as those which would allow their opponent to win.

10.2 Suggestions for Future Work

There are several directions in which to continue the work described in this dissertation. Some of these directions are included here:

1. Economic analysis of the cost of using covert channels in financial data. Given sufficient resources it is unquestionably possible to temporarily control the price of a stock. Or, as a market maker, to control the listing of orders for stock. In both cases, there is some capacity for covert communication. But there is also cost and risk. It costs money to buy enough stock to control the price. And it entails risk to hold the stock and control the price longer enough to transmit the secret message. If the secret has a certain value to the sender and receiver, then an equilibrium can be found to determine the cost and level of activity required to safely transmit it through a covert channel in financial data.

2. Extend the context-sensitive tiling stego-system to higher dimensions and apply it to other media. Our work demonstrated the feasibility of tiling systems for steganography using 2 dimensional images. Our work is also applicable to higher dimensional space, such as spatial images, and other media, such as audio and video. One particularly interesting direction this work could go is into watermarking of source code, which is context-free and therefore trivially context-sensitive. Another direction could be an application to natural language generation, utilizing the context of previous statements to guide the construction of the next statement.

3. Develop extensions to the Huffman tree stego-system to cover canonical trees and dynamic Huffman codes. One of the ways in which our stego-system for Huffman trees can be defeated is to convert the tree to its canonical form. Since there is only one canonical tree for a given symbol distribution, this countermeasure completely eliminates the capacity of the channel as presented. However, the number of canonical trees for all distributions over n symbols is greater

than 1, thus there is capacity in the choice of distribution. Minor changes to the symbol distribution may be able to provide a choice of canonical tree, and thus covert channel capacity in that choice. A similar idea should work for using dynamic Huffman codes as a covert channel.

4. Deeper analysis of the steganographic capacity of Sudoku clue configurations. Our work presented a method capable of exploiting 32.4327 ± 0.1366 bits of capacity in the placement of clues. However, there is very likely more capacity that can be exploited. By taking the log of the sum over the number of clue configurations with between 17 and 32 clues, we can say that the maximum capacity of a clue configuration for a Sudoku puzzle is 76.259 bits. A more accurate value can be obtained by counting only those clue configurations which are solvable.

5. Build stego-system that incorporates a model of human interaction which uses a cognitive architecture such as ACT-R. Rule-based models for describing the behavior of gameplaying agents are sufficient for conducting covert communication through the game or interaction. However, a warden with access to actual human data will eventually be able to detect a difference between the observed and expected gameplay behavior. To further delay this detection, a stego-agent should learn while playing the game, and that learning should proceed in a manner which appears similar to a human's learning. This is not possible with a rules-based model, but is possible with a cognitive architecture.

6. Further and advanced development of the software testbed for analyzing steganographic interactions. There are several avenues for future work in this direction: (1) Rewrite the GUI using JavaFX and include functionality for plotting graphs of results; (2) Expand the library of interactions in which covert channels can be studied by including a parser for games written in a game description language; (3) Extending and strengthening the library of methods for steganalysis would greatly improve the utility of the testbed.

REFERENCES

REFERENCES

[1] Ronald L. Rivest. Cryptology. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 1. Elsevier, 1990.

[2] David Kahn. *The Code-Breakers*. Scribner, 1996.

[3] Fabien A.P. Petitcolas, Ross J. Anderson, and Marcus G. Kuhn. Information hiding: A survey. *Proceedings of the IEEE (special issue)*, 87(7):1062–1078, 1999.

[4] Jim Reeds. Solved: The ciphers in book III of Trithemius's Steganographia. *Cryptologia*, 1998.

[5] Simon Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Anchor, 2000.

[6] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[7] Craig H. Rowland. Covert channels in the TCP/IP protocol suite. *First Monday [Online]*, 2(5), May 1997.

[8] Richard A. Kemmerer. Shared resource matrix methodology: An approach to identifying storage and timing channels. *ACM Transactions on Computer Systems*, 1(3):256–277, August 1983.

[9] Virgil D. Gligor. Covert channel analysis of trusted systems (light pink book). Technical Report NCSC-TG-030, National Computer Security Center, 1993.

[10] J. Zollner, H. Federrath, H. Klimant, A. Pfitzmann, R. Piotraschke, A. Westfeld, G. Wicke, and G. Wolf. Modeling the security of steganographic systems. In *Proceedings of the International Workshop on Information Hiding*, pages 345–355, April 1998.

[11] Christian Cachin. An information-theoretic model for steganography. *Information and Computation*, 192(1):41–56, 2004.

[12] Stefan Katzenbeisser and Fabien A.P. Petitcolas. Defining security in steganographic systems. In E.J. Delp and P.W. Won, editors, *Proceedings of SPIE 4675, Security and Watermarking of Multimedia Contents IV*, pages 260–368, 2002.

[13] Nicholas J. Hopper, John Langford, and Luis von Ahn. Provably secure steganography. In *Proceedings of Advances in Cryptology*, pages 77–92. Springer, 2002.

[14] Nicholas J. Hopper, John Langford, and Luis von Ahn. Provably secure steganography. *IEEE Transactions on Computers*, 58(5):662–676, May 2009.

[15] Neil F. Johnson and Stefan C. Katzenbeisser. A survey of steganographic techniques. In Stefan Katzenbeisser and Fabien A. Petitcolas, editors, *Proceedings of Information Hiding Techniques for Steganography and Digital Watermarking*, Norwood, MA, USA, 2000. Artech House, Inc.

[16] Lisa Marvel, Charles Boncelet, and Charles Retter. Spread spectrum image steganography. *IEEE Transactions on Image Processing*, 8:1075–1083, 1999.

[17] Ingemar J. Cox, Joe Kilian, F. Thomson Leighton, and Talal Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing*, 6:1673–1687, 1997.

[18] Ross Anderson and Fabien Petitcolas. On the limits of steganography. *IEEE Journal of Selected Areas in Communications*, 16:474–481, 1998.

[19] Oktay Altun, Gaurav Sharma, Mehmet Celik, Mark Sterling, Edward Titlebaum, and Mark Bocko. Morphological steganalysis of audio signals and the principle of diminishing marginal distortions. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2:21–24, 2005.

[20] Carlo Blundo and Clemente Galdi. Hiding information in image mosaics. *The Computer Journal*, 46(2):202–212, 2003.

[21] Phil Sallee. Model-based steganography. In *Proceedings of International Workshop on Digital Watermarking*, pages 154–167. Lecture Notes in Computer Science 2939, Springer-Verlag, 2004, October 2003.

[22] Regunathan Radhakrishnan, Mehdi Kharrazi, and Nasir Memon. Data masking: A new approach for steganography? *Journal of VLSI Signal Processing Systems*, 41(3):293–303, 2005.

[23] Andrew H. Sung, Gopalakrishna Reddy Tadiparthi, and Srinivas Mukkamala. Defeating the current steganalysis techniques (robust steganography). *International Conference on Information Technology: Coding and Computing*, 1, 2004.

[24] Gopalakrishna Reddy Tadiparthi and Toshiyuki Sueyoshi. StegAnim – A novel information hiding technique using animations. *Engineering Letters*, 13(3), 2006.

[25] Philip C. Ritchey, Jorge R. Ramos, and Vernon J. Rego. A framework for synthetic stego. In *Proceedings of the Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–4, 2009.

[26] Brian Lange. Steganography using the chess PGN standard format. Technical report, SANS Institute, 2004.

[27] Abdelrahman Desoky and Mohamed Younis. Chestega: chess steganography methodology. *Security and Communication Networks*, 2(6):555–566, 2009.

[28] Peter Wayner. Mimic functions. *Cryptologia*, 16(3):193–214, 1992.

[29] Keith Winstein. Lexical steganography through adaptive modulation of the word choice hash. `http://alumni.imsa.edu/~keithw/tlex/lsteg.ps`, January 1999.

[30] Mark Chapman and George I. Davida. Plausible deniability using automated linguistic stegonagraphy. In *Proceedings of the International Conference on Infrastructure Security*, pages 276–287, London, UK, 2002. Springer-Verlag.

[31] Krista Bennett. Linguistic steganography: Survey, analysis and robustness concerns for hiding information in text. Technical report, Center for Education and Research in Information Assurance and Security, Purdue University, 2004.

[32] Peter Winkler. The advent of cryptology in the game of bridge. *Cryptologia*, 7(4):327–332, 1983.

[33] Naomoto Niwyama, Nasen Chen, Takeshi Ogihara, and Yukio Kaneda. A steganographic method for mazes. In *Proceedings of the Pacific Rim Workshop on Digital Steganography*, Kitakyushu, Japan, 2002.

[34] Hui-Lung Lee, Chia-Feng Lee, and Ling-Hwei Chen. An improved method for hiding data in a maze. In *International Conference on Machine Learning and Cybernetics*, pages 3161–3165, 2008.

[35] Hui-Lung Lee, Chia-Feng Lee, and Ling-Hwei Chen. A perfect maze based steganographic method. *The Journal of Systems Software*, 83(12):2528–2535, December 2010.

[36] M. Hassan Shirali-Shahreza and Mohammad Shirali-Shahreza. Steganography in SMS by Sudoku puzzle. In *Proceedings of the International Conference on Computer Systems and Applications*, AICCSA '08, pages 844–847, Washington, DC, USA, 2008. IEEE Computer Society.

[37] En-Jung Farn and Chaur-Chin Chen. Novel steganographic method based on jig swap puzzle images. *Journal of Electronic Imaging*, 18(1):013003–1–013003–10, January 2009.

[38] En-Jung Farn and Chaur-Chin Chen. Jigsaw puzzle images for steganography. *Optical Engineering*, 48(7):077006–1–07006–12, July 2009.

[39] Zhan-He Ou and Ling-Hwei Chen. Hiding data in tetris. In *International Conference on Machine Learning and Cybernetics*, pages 61–67, 2011.

[40] Steven J. Murdoch and Piotr Zielinski. Covert channels for collusion in online computer games. In *Proceedings of the International Workshop on Information Hiding*, pages 355–369. Springer, 2004.

[41] Julio C. Hernandez-Castro, Ignacio Blasco-Lopez, Juan M. Estevez-Tapiador, and Arturo Ribagorda-Garnacho. Steganography in games: A general methodology and its application to the game of go. *Computers and Security*, 25(1):64–71, 2006.

[42] Malte Diehl. Secure covert channels in multiplayer games. In *Proceedings of the ACM Workshop on Multimedia and Security*, pages 117–122, 2008.

[43] H.W. Kuhn. A simplified two person poker. In *Contributions to the Theory of Games*, pages 97–107. Princeton University Press, 1950.

[44] Sebastian Zander, Grenville Armitage, and Philip Branch. Covert channels in multiplayer first person shooter online games. In *Proceedings of the 33rd IEEE Conference on Local Computer Networks*, pages 215 –222, October 2008.

[45] Sebastian Zander, Grenville Armitage, and Philip Branch. Reliable transmission over covert channels in first person shooter multiplayer games. In *Proceedings of the 34th IEEE Conference on Local Computer Networks*, pages 169–176, October 2009.

[46] Sebastian Zander. *Performance of Selected Noisy Covert Channels and Their Countermeasures in IP Networks*. PhD thesis, Swinburne University of Technology, Melbourne, Australia, 2010.

[47] John von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton Press, 1944.

[48] R. Pool. Putting game theory to the test. *Science*, 267:1591–1593, 1995.

[49] Robert L. West, Christian Lebeire, and Dan J. Bothell. Cognitive architectures, game playing, and human evolution. In R. Sun, editor, *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, pages 103–123. Cambridge University Press, 2006.

[50] Allen Newell. Production systems: Models of control structures. In W.G. Chase, editor, *Visual Information Processing*, pages 463–526. Academic Press, 1973.

[51] John R. Anderson and Christian Lebiere. *The atomic components of thought*. Erlbaum, 1998.

[52] John R. Anderson, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004.

[53] Christian Lebiere and Robert L. West. A dynamic ACT-R model of simple games. In *Proceedings of the Conference of the Cognitive Science Society*, pages 296–301, 1999.

[54] Robert L. West and Christian Lebiere. Simple games as dynamic, coupled systems: Randomness and other emergent properties. *Journal of Cognitive Systems Research*, 1(4):221–239, 2001.

[55] Philip C. Ritchey and Vernon J. Rego. Synthetic steganographic series and finance. *IEEE Latin America Transactions*, accepted 2014.

[56] Stefan Katzenbeisser and Fabien A.P. Petitcolas, editors. *Information Hiding: techniques for steganography and digital watermarking*. Artech House, 2000.

[57] Momcilo Novkovic. On exponential autoregressive time series models. *Novi Sad Journal of Mathematics*, 29:97–101, 1999.

[58] D.P. Gaver and P.A.W. Lewis. First order autoregressive gamma sequences and point processes. *Advances in Applied Probability*, 12:727–745, 1980.

[59] A.J. Lawrance and P.A.W. Lewis. A new autoregressive time series model in exponential variables NEAR(1). *Advances in Applied Probability*, 13:826–845, 1981.

[60] Lee S. Dewald and Peter A. W. Lewis. A new Laplace second-order autoregressive time-series model – NLAR(2). *IEEE Transactions on Information Theory*, IT-31:645–651, 1985.

[61] P.A.W. Lewis, E. McKenzie, and D.K. Hugus. Gamma processes. *Stochastic Models*, 5:1–30, 1989.

[62] Benjamin Melamed. An overview of TES processes and modeling methodology. In L. Donatiello and R. Nelson, editors, *Performance Evaluation of Computer and Communication Systems*, pages 359–393. Springer-Verlag, 1993.

[63] Benjamin Melamed. The empirical TES methodology: Modeling empirical time series. *Journal of Applied Mathematics and Stochastic Analysis*, 10:333–353, 1997.

[64] Philip C. Ritchey and Vernon J. Rego. A context-sensitive tiling system for information hiding. *Journal of Information Hiding and Multimedia Signal Processing*, 3(3):212–226, 2012.

[65] John C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill Higher Education, 1997.

[66] Alfonso Muñoz. Stegsecret. `http://stegsecret.sourceforge.net`, 2007.

[67] Michal Wegrzyn. Virtual steganographic laboratory. `http://vsl.sourceforge.net`, 2009.

[68] Kathryn Hempstalk. Digitial Invisible Ink Toolkit. `http://diit.sourceforge.net`, 2006.

[69] Jessica Fridrich, Miroslav Goljan, and Rui Du. Detecting LSB steganography in color, and gray-scale images. *Multimedia, IEEE*, 8(4):22–28, 2001.

[70] Andreas Westfeld and Andreas Pfitzmann. Attacks on steganographic systems – Breaking the steganographic utilities EzStego, Jsteg, Steganos, and S-Tools – and some lessons learned. In *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 2000.

[71] Ismail Avcibas, Mehdi Kharrazi, Nasir Memon, and Bulent Sankur. Image steganalysis with binary similarity measures. *EURASIP Journal of Applied Signal Processing*, 2005:2749–2757, 2005.

[72] Pedro Vit. On the equivalence of certain markov chains. *Journal of Applied Probability*, 13(2):357–360, 1976.

[73] Auguste Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, 9:5–83,161–191, 1883.

[74] David A. Huffman. A method for the construction of minimum redundancy codes. *Institute of Radio Engineers*, 40:1098–1101, 1952.

[75] David Salomon. *Data Compression: The Complete Reference*. Springer, 4th edition, 2007.

[76] Kuo-Nan Chen, Chin-Feng Lee, Chin-Chen Chang, and Huan-Ching Lin. Embedding secret messages using modified Huffman coding. In *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 278–281, 2009.

[77] Kuo-Nan Chen, Chin-Feng Lee, and Chin-Chen Chang. Embedding secret messages based on chaotic map and Huffman coding. In *Proceedings of International Conference on Ubiquitous Information Management and Communication*, pages 336–341, 2009.

[78] Philip C. Ritchey and Vernon J. Rego. Hiding secret messages in Huffman trees. In *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 71–74, 2012.

[79] Francesc Sebe, Josep Domingo-Ferrer, and Jordi Herrera. Spatial-domain image watermarking robust against compression, filtering, cropping, and scaling. In *Proceedings of the Information Security Workshop*, pages 44–53. Springer-Verlag, 2000.

[80] Mikhail J Atallah, Victor Raskin, Christian F. Hempelmann, Mercan Karahan, Radu Sion, Umut Topkara, and Katrina E. Triezenberg. Natural language watermarking and tamperproofing. In *Proceedings of the International Workshop on Information Hiding*, Lecture Notes in Computer Science, pages 7–9. Springer Verlag, 2002.

[81] Jessica Fridrich. *Steganography in digital media: principles, algorithms, and applications*. Cambridge University Press, 2010.

[82] Peter Wayner. *Disappearing Cryptography: Information Hiding: Steganography and Watermarking*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2009.

[83] Philip C. Ritchey and Vernon J. Rego. Stegodoku: Data hiding in Sudoku puzzles. *Designs, Codes and Cryptography*, submitted 2015.

[84] Bertram Felgenhauer and Frazer Jarvis. Enumerating possible Sudoku grids. `http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf`, 2005.

[85] Gary McGuire, Bastian Tugemann, and Gilles Civario. There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem. *CoRR*, abs/1201.0749:1–43, 2012.

[86] Ross Anderson. Stretching the limits of steganography. In R. J. Anderson, editor, *Proceedings of the International Workshop on Information Hiding*, volume 1174 of *Lecture Notes in Computer Science*, pages 39–48. Springer-Verlag, Berlin, May 30–June 1 1996.

[87] D.H. Lehmer. Machine tools of computation. In E.F. Beckenbach, editor, *Applied Combinatorial Mathematics*, pages 27–30. Wiley, 1964.

[88] Aviezri Fraenkel. Combinatorial games: Selected bibliography with a succinct gourmet introduction. In R. J. Nowakowski, editor, *Games of No Chance*, pages 493–537. MSRI Publications, Cambridge University Press, 1996.

[89] Gustavus J. Simmons. The prisoner's problem and the subliminal channel. In *Proceedings of Advances in Cryptology*. Plenum Press, 1983.

[90] Philip C. Ritchey and Vernon J. Rego. Covert channels in combinatorial games. In *Proceedings of the International Workshop on Distributed Simulation and Online Gaming – ICST Conference on Simulation Tools and Techniques*, pages 241–248, 2012.

[91] Ronald L. Rivest. Chaffing and winnowing: Confidentiality without encryption. Technical report, MIT Lab for Computer Science, March 1998.

[92] Kevin Crowley and Robert S. Siegler. Flexible strategy use in young children's Tic-tac-toe. *Cognitive Science*, 17(4):531–561, 1993.

[93] George T. Heineman, Gary Pollice, and Stanley Selkow. *Algorithms in a Nutshell*, chapter 7, pages 213–217. Oreilly Media, 2008.

[94] P Forczmanski and Michal Wegrzyn. Virtual steganographic laboratory for digital images. In *Proceedings of Information Systems Architecture and Technology: Information Systems and Computer Communication Networks*, pages 163–174, 2008.

APPENDIX

A ACCESSING THE SOURCE CODE

All of the source code developed for use in this dissertation and other related research projects is available in public GIT repositories. These repositories are listed below.

GGP-Base	`https://bitbucket.org/pritchey/ggp_base`
libPhil	`https://bitbucket.org/pritchey/libphil`
MATLAB Code	`https://bitbucket.org/pritchey/matlab`
MyPidgin	`https://bitbucket.org/pritchey/my-pidgin`
PGN Parser	`https://github.com/kingpiko/pgn-parser`
STASI	`https://bitbucket.org/gdihong/stegame`
SteGo	`https://bitbucket.org/pritchey/stego`
StegoDoku	`https://bitbucket.org/pritchey/stegodoku`
TTS Pidgin Plugin	`https://bitbucket.org/pritchey/pidgin-tictacstego`

VITA

VITA

Philip Carson Ritchey received the B.S. degree with honors in computer engineering from Texas A&M University in 2008. He received the Ph.D. degree in computer science from Purdue University in 2015 under the advisement of Professor Vernon Rego. At Purdue University, he was a member of the Center for Education and Research in Information Assurance and Security and the Center for Science of Information. He is a two-time recipient of the Department of Computer Science's award for Outstanding Service to the Department and served as a graduate lecturer.