HIDENQA: HIDE INFORMATION IN MULTIPLE-CHOICE QUESTIONS/ANSWERS

A Project

Presented to the faculty of the Department of Computer Science
California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Rekha Godha Hamilton

FALL
2012
HIDENQA: HIDE INFORMATION IN MULTIPLE-CHOICE QUESTIONS/ANSWERS

A Project

by

Rekha Godha Hamilton

Approved by:

__________________________________, Committee Chair
Ted Krovetz, Ph.D.

__________________________________, Second Reader
Ahmed Salem, Ph.D.

____________________________
Date
Student:  Rekha Godha Hamilton

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

__________________________, Graduate Coordinator
Dr. Nikrouz Faroughi

Department of Computer Science
Abstract

of

HIDENQA: HIDE INFORMATION IN MULTIPLE-CHOICE QUESTIONS/ANSWERS

by

Rekha Godha Hamilton

This project implements a steganography tool named HideNQA that will hide text within the GRE’s multiple-choice questions and answers. With HideNQA, confidential information can be smuggled out in an innocuous looking GRE exam.

“Noise” here refers to random bits in the covertext that could be modified without discernible change in the quality of the image, audio or video file (covertexts). “Noise” could also mean the traces left behind by a particular steganographic technique.

This paradigm was first introduced by Abdelrahman Desoky. Most current steganography algorithms/methods leave some signature or noise in the covertext that can be detected by current steganalysis tools. These algorithms are then vulnerable to analysis and attacks because of these clues left behind. This project will implement a component of Noiseless Steganography, i.e., Nostega called Edustega (short for Educational Steganography) - as described in Desoky’s dissertation.

_______________________, Committee Chair
Ted Krovetz, Ph.D.

_______________________
Date
DEDICATION

To my Amma & Appa - my first teachers, you taught me to love learning. Everything that I’ve achieved is but a tribute to you.

To Andrew – my rock, my love, and my friend. I couldn’t have done this without you.
ACKNOWLEDGMENTS

I would like to thank professor, Dr. Ted Krovetz for volunteering his time and effort in reading through my project paper. His advice and positive comments were instrumental to my work on the project.

I would like to thank my kids, mother-in-law, family and friends for their encouragement and support throughout my Master’s program.

I would like to thank Dr. Ahmed Salem for being my second reader. I would also like to thank Dr. Nikrouz Faroughi, Graduate Coordinator of the Computer Science Department and the Computer Science Department staff for their support.
TABLE OF CONTENTS

Dedication........................................................................................................................................... v
Acknowledgements..................................................................................................................................... vi
List of Tables ........................................................................................................................................... viii
List of Figures ........................................................................................................................................... ix
Chapter
1. INTRODUCTION ................................................................................................................................. 1
   1.1 Background ....................................................................................................................................... 1
   1.2 Purpose ......................................................................................................................................... 6
   1.3 Scope ........................................................................................................................................... 10
2. HIDDENQA TOOL ARCHITECTURE AND DESIGN ................................................................. 12
   2.1 Introduction .................................................................................................................................... 12
   2.2 Encode Module ................................................................................................................................. 13
   2.3 Construct Module ............................................................................................................................ 23
   2.4 Compress and Encrypt Modules .................................................................................................... 26
   2.5 Decode Module ............................................................................................................................... 27
   2.6 Decrypt Module and Uncompress Modules ................................................................................. 28
3. HIDDENQA TOOL IMPLEMENTATION ..................................................................................... 29
   3.1 Introduction .................................................................................................................................... 29
   3.2 Development Platform and Language ............................................................................................ 29
   3.3 Graphical User Interface Design and Implementation ................................................................. 30
   3.4 Functions and Data Objects ............................................................................................................ 41
   3.5 HideNQA’s Database Backend ....................................................................................................... 43
4. ANALYSIS AND FUTURE WORK ............................................................................................... 45
   4.1 Analysis ......................................................................................................................................... 45
   4.2 Future Work ................................................................................................................................. 47
Bibliography ............................................................................................................................................. 49
LIST OF TABLES

<table>
<thead>
<tr>
<th>Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Table 2.1 Encoding for Option 1. - for correct answer</td>
<td>16</td>
</tr>
<tr>
<td>2. Table 2.2 Encoding for Option 1. – for wrong answer</td>
<td>16</td>
</tr>
<tr>
<td>3. Table 2.3 Encoding for Option 2., for chosen answers</td>
<td>19</td>
</tr>
<tr>
<td>4. Table 2.4 Latin Square Array for correct or chosen answers</td>
<td>21</td>
</tr>
<tr>
<td>5. Table 2.5 Latin Square Array for wrong answers</td>
<td>22</td>
</tr>
<tr>
<td>6. Table 2.6 Latin Square Array for wrong answers</td>
<td>23</td>
</tr>
<tr>
<td>7. Table 4.1 HideNQA Stegotext to Covertext ratio</td>
<td>46</td>
</tr>
<tr>
<td>8. Table 4.2 Bitrate comparison of Linguistic Steganography methods</td>
<td>47</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Figure 1.1 Stegotext transmitted via exam paper between Bob and Alice</td>
<td>8</td>
</tr>
<tr>
<td>2. Figure 2.1 High level overview of HideNQA’s Modules</td>
<td>12</td>
</tr>
<tr>
<td>3. Figure 2.2 Overview of Encode – for Option 1</td>
<td>15</td>
</tr>
<tr>
<td>4. Figure 2.3 Overview of Encode – for Option 2</td>
<td>19</td>
</tr>
<tr>
<td>5. Figure 2.4 Overview of Construct Module</td>
<td>24</td>
</tr>
<tr>
<td>6. Figure 3.1 HideNQA’s Main dialog box</td>
<td>31</td>
</tr>
<tr>
<td>7. Figure 3.2 Hide task flow</td>
<td>32</td>
</tr>
<tr>
<td>8. Figure 3.3 Reveal task flow</td>
<td>33</td>
</tr>
<tr>
<td>9. Figure 3.4 Hide task flow – Option 1</td>
<td>34</td>
</tr>
<tr>
<td>10. Figure 3.5 Hide task flow – Option 2</td>
<td>35</td>
</tr>
<tr>
<td>11. Figure 3.6 Sanitization of Input</td>
<td>36</td>
</tr>
<tr>
<td>12. Figure 3.7 Output for Hide task flow – Option 1</td>
<td>37</td>
</tr>
<tr>
<td>13. Figure 3.8 Output for Hide task flow - Option 1</td>
<td>38</td>
</tr>
<tr>
<td>14. Figure 3.9 Prompt to save file</td>
<td>38</td>
</tr>
<tr>
<td>15. Figure 3.10 File successfully saved message</td>
<td>39</td>
</tr>
<tr>
<td>16. Figure 3.11 Output for Hide task flow – Option 2</td>
<td>39</td>
</tr>
<tr>
<td>17. Figure 3.12 Output for Hide task flow – Option 2</td>
<td>40</td>
</tr>
<tr>
<td>18. Figure 3.13 Output for Reveal task flow – Option 1</td>
<td>41</td>
</tr>
<tr>
<td>19. Figure 3.14 Output for Reveal task flow – Option 2</td>
<td>41</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

1.1 Background

The science of hiding information, devising a way to communicate it securely and protecting the communicating parties from consequences is called steganography. It hides information and also the fact that the communication is taking place. The primary goal of steganography is to prevent suspicion of the existence of covert communications. Preventing the discovery of the hidden information is a subsequent goal of steganography. Just as other useful paradigms such as cryptography, steganography was adapted into the digital world and the first known digital steganography began to appear in about 1985 [1]. Steganography can be used in conjunction with other data manipulation techniques such as compression and encryption. The secret is usually called the stegotext and the innocent looking document, image, audio, or video file that is used to conceal the secret is called the covertext. Steganalysis is the counterpart to steganography and uses various techniques to determine the strengths and weaknesses of steganography algorithms and protocols. The field of steganalysis can be employed by law enforcement, government officials or by malicious attackers to detect steganography using reverse-engineering, brute-force attacks, packet analysis, etc.
Steganography is a closely related cousin of the well-established field of cryptography. While both steganography and cryptography hide data, the former is motivated by different reasons than the latter. Cryptography protects the data but steganography protects the data and the communicators. For example, there are political regimes which prohibit their citizens from communicating freely. These citizens could communicate using digital steganography without fear of repercussions from their oppressive governments. Another example would be where a whistle blower might want to use steganography to transmit information about illicit activities within his/her company without fear of discovery. Steganography can be used any place where not only the data but also the communication must be kept secret to protect the sender and the receiver from censorship and discovery. It can be used in conjunction with cryptography to not only secure a secret but also the communication of the secret itself.

Hiding information within images and audio files has been a very popular form of digital steganography. This is chiefly because these files can hide lots of data without losing any humanly discernible image quality or audio quality. Almost an eighth of an original image can be used to hide a secret [2]. Thus large files can be hidden within innocuous and contextually appropriate image/audio files and transmitted across digital channels without fear of accidental discovery. Most of the algorithms used to hide information within image and audio files take advantage of the fact that these files contain a certain amount of “noise” or redundant information that can be exploited to hide secrets. The
The basic algorithm is known as the least-significant bit algorithm. Consider the case where 24-bits are allocated per pixel in an image file. That is 8-bits for red, blue and green. If the least significant bit was changed for each color per pixel and this process is repeated over all the pixels in the file, a sizeable amount of information can be hidden in the image file.

There are a few problems with the above approach. Image files are prone to conversions when they are transmitted or stored on the internet for optimization purposes. Some or all of the stegotext contained within a covertext image could be lost if the image is inadvertently optimized during transmission. Another problem is that with the rise in popularity of image and audio steganography there has been a parallel increase in steganalysis techniques that can detect this type of steganography. Analysis that compares original image/audio file to the modified file could easily detect the fact that something is hidden and in many cases exactly what has been hidden. Even if the actual stegotext is encrypted or compressed or both, the analysis has discovered that something was hidden. In cases where the original image is unavailable, analysis techniques can still use histograms to detect the manipulation of bits – what the human eye cannot detect can be detected by image and audio analysis tools.

Another technique used in steganography is the use of error correction codes [2]. Small errors or “noise” can be introduced into covertexts. These errors will be corrected
automatically if they are opened and read by ordinary tools such as standard document readers, JPEG viewers, etc. If these files are opened by more sophisticated tools – the secret message hidden in the errors can be found by either comparing the “error” file to the original version or by using the error-correcting principles to point to the location(s) of the secret message. For example, if a covertext has 8-bit blocks then 3-bits of hidden information can be sent per block. So, if 000 needs to be sent then flip bit 0. If 011 needs to be sent then flip bit 3. The error-correction codes will take care of these intentional errors if a standard tool opens up the covertext file. One of the problems with the above approach is that covertexts with large amounts of errors could be flagged as files potentially hiding something. There exist analysis techniques that can reverse engineer such files and are used to discover the stegotext.

Text steganography is the science of hiding information within text documents [1]. This field in general includes the manipulation or fabrication of text in order to hide the secret information. Format-based methods, random and statistical generation, and linguistic methods [3] are often employed to hide the secret information. Linguistic steganography is a sub-field of text steganography and uses the syntax and semantics of text to hide the information [3]. The algorithms that work for image and audio files as covertexts do not generally work for text documents as covertexts. The algorithms and techniques suggested for linguistic steganography are generally more complex than the simple least-
significant bit algorithms suggested for image and audio covertext files. This fact is explained by Krista Bennett in [3]:

“Natural language cover texts must not only pass the statistical muster of automatic analysis, but also the minds of human readers. Linguistically naïve approaches to the problem use statistical frequency of letter combinations or random dictionary words to encode information. More sophisticated approaches use context-free grammars to generate syntactically correct cover text which mimics the syntax of natural text.”

One problem with linguistic steganography methods today is the lack of cohesiveness in the general semantics of a textual document. This can directly point to a steganography transaction. If a textual document has been manipulated using linguistic steganography, then random words, intentional misspellings, grammatical mistakes can all point to the fact that steganography has been employed. All of the above techniques are tantamount to adding noise to a textual document covertext. They can be viewed as noise since they would stand out as abnormal when viewed by a human reader or parsed by a syntactical steganalysis tool. Another problem is that current linguistic steganography techniques can hide only small amounts of stegotext in their covertext. Compared to image or audio steganography, the stegotext to covertext ratio is much smaller. There are certainly no current steganography algorithms/techniques to be able to use one-eighth of a textual document to hide secret information.
1.2 Purpose

This project will implement a standalone tool called HideNQA with a GUI (graphical user interface) that will include the following:

- Compression of the original data string entered (stegotext)
- Encryption of the stegotext
- Encoding and concealment of the stegotext within recommended multiple-choice questions and answers returned to the user in a convenient text file format.
- Decoding of the covertext to recover the stegotext.

HideNQA has the following requirements:

- Some information necessary for encoding will have to be transmitted by the user using secure out-of-band communication.
- HideNQA does not actually format the multiple-choice exam. In order to give credibility to the covertext, a formal test generator should be used in conjunction with the resulting text file to create a properly formatted exam. Test generators can be found online. [4], [5]

The purpose of this project is to implement a steganography tool named HideNQA that will use ideas in Linguistic Steganography and a novel paradigm called Nostega [6] – steganography that neither uses noise nor creates noise in the covertext. Noise as stated
in 1.1 above, refers to random bits in the covertext that could be modified without discernible change in the quality of the image, audio or video file (covertexsts). Noise could also mean the traces left behind by a particular steganographic technique – for example, error control code techniques make errors on purpose to hide information [2].

This paradigm was first introduced by Abdelrahman Desoky in 2008 [7] as part of his doctoral dissertation at the University of Maryland, Baltimore county. The excerpt below is from his book, Noiseless Steganography, published in 2012 covering Nostega [6]:

“Nostega opts to make the presence of the hidden data natural in the cover so that neither linguistic flaws nor peculiarities are introduced as a side effect. In addition, Nostega strives to legitimize the interaction among the communicating parties so that an adversary would not suspect the association between a sender and a receiver”

Edustega is one of many methodologies of the Nostega paradigm described in Desoky’s book [6]. Edustega deals with hiding information within educational materials such as homework, lectures, exam papers, etc. With this methodology, the steganography users would have an academic relationship. As an example, two generic users Bob and Alice could have a professor/student relationship. This cover allows them to legitimately communicate and exchange information via educational materials. The information to be hidden will be inside these materials. Prior agreements on where to look for the concealed information will have to be in place. The relationship does not have to be a
professor/student one. Any cover relationship that allows educational materials to flow back-and-forth between two users is adequate.

This project will focus on implementing the hiding of information within exam papers – in particular, answers to multiple choice questions within exam papers by creating a tool called HideNQA. HideNQA will facilitate the hiding of stegotext to be transmitted between two communicating parties.

![Figure 1.1 Stegotext transmitted via exam paper between Bob and Alice](image)

As shown in Figure 1.1 above: Bob, the professor, sends Alice an exam paper of multiple choice questions. This is a legitimate paper with corresponding legitimate choice of answers. Here, the information is hidden in both the correct answer and the wrong answers to increase stegotext to covertext bit ratio. Alice, the student, returns the
exam paper marked with some chosen answers. These answers may not all be the correct answers. Alice will pick the answers based on HideNQA’s encoding of her secret acknowledgment back to Bob. The tool will be able to encode the stegotext within an exam paper as needed by Bob or encode the stegotext within the answers to an exam paper as needed by Alice.

HideNQA will automate the following steps: the information to be hidden will be first compressed and encrypted before being encoded and camouflaged as multiple-choice questions/answers within an exam paper. Compression is used here primarily to increase the bitrate of hidden information. Symmetric key encryption is employed to increase security and randomize the secret to be encoded. Latin squares [6], [8] will be used to randomize how the multiple choice questions/answers are used to conceal the information by the tool. This is to thwart steganalysis attacks based on correlating multiple choice answers with particular steganographic codes.

The implementation will also include a database/data bank of multiple-choice questions that can be selected and manipulated to store the hidden information. Once particular questions/answers are chosen to hide the information, they will be delivered to the sender in a convenient electronic form (text files) so they can be imported into an actual exam generator. Commercial and Freeware Exam generators are available through the internet. These can be used manually by the sender to create a credible exam based on
the results returned by the tool. Decoding of a given exam covertext is done by entering all the answers or the chosen answers into HideNQA. The decode will also require the Latin square arrays used to encode and the number of bits in the stegotext in order to accurately decode the stegotext. Stegotext will be returned to the receiver in the form of a text file.

1.3 Scope

The intent of HideNQA is not to send huge amounts of text across the channel – instead it is meant to be used to send small crucial pieces of text that are highly sensitive and thus need a good steganographic cover. If a large chunk of sensitive information needs to be sent, then HideNQA can be used multiple times to encode the data into several different exam papers. HideNQA requires a secure cover to be established between sender and receiver such as a professor/student relationship. A secure out-of-band channel has to be established between sender and receiver so they may safely communicate the Latin squares, symmetric key and number of bits transmitted.

HideNQA provides the following benefits:

- Robustness – information cannot be destroyed as long as the content of the exam paper remains intact – the exam paper can be transported as a hardcopy and the stegotext will remain intact and ready to be recovered.
- Higher stegotext to coverted text bit ratio compared to existing linguistic steganography techniques. [6] This bit ratio is further enhanced by the compression used by the tool.

- Enhanced steganography – the coverted text is a legitimate exam paper or answers to a legitimate exam paper. Legitimacy of the coverted text, the fact that noise is not used to conceal the stegotext, and the use of Latin squares to confuse eliminates many of the possible steganalysis attacks.

- Security is provided by encryption of the stegotext
Chapter 2
HIDENQA TOOL ARCHITECTURE AND DESIGN

2.1 Introduction

HideNQA will hide text using four separate modules. Please see Figure 2.1 below for an overview of the modules. The top half shows the Hide task flow – stegotext is passed...
through the four modules to give either a text file containing the appropriate multiple-choice exam questions or a text file with the chosen answers to the exam questions.

In the Hide task flow, stegotext, i.e., the data to be hidden is passed through the first module which will compress the data for better stegotext to covertext bit ratio. Next, the Encrypt module will use a cryptographic algorithm and a symmetric key to encrypt the stegotext. The third module starts the work of steganography by assigning codes to the data returned by the Encryption module. Details on how codes are assigned to a given stegotext are discussed below in Section 2.3. Finally, the encoding returned by the Encode module will be used by the Construct module to convert the encoding into appropriate covertext. As described in Section 1.2, the appropriate covertext could either be a multiple-choice exam or answers to be chosen on the exam.

The bottom half of Figure 2.1 above shows the Reveal task flow – the user will enter the relevant encoding taken from the covertext. The Decode module will then return encrypted stegotext. The Decrypt module will ask for the symmetric key used to encrypt the stegotext and return the compressed stegotext. The Uncompress module will finally expand the compressed stegotext to return the original stegotext.

2.2 Encode Module

Desoky’s Edustega methodology generally describes the different options to hide information within educational materials. HideNQA chooses to implement certain
aspects of Edustega. The tables in this section are derived from the explanations and tables in Desoky’s book. [6] Section 2.3 is divided into three different subsections. Subsections 2.3.1 and 2.3.2 describe the two different ways to encode using the HideNQA tool. Section 2.3.3 describes the use of Latin Squares to add the element of confusion to the Encode module. There are two different ways to encode using the HideNQA tool.

Option 1. Encode the stegotext in all the answers of the exam paper. This would be the method adopted by Bob as shown in Figure 1.1 in Chapter 1. This option would generate an encoding that will eventually lead to the construction of an exam paper.

Option 2. Encode the stegotext in chosen answers of the exam paper. Alice in Figure 1.1 would use this method to send a reply back to Bob. This option would generate an encoding that can be used to mark certain answers on an exam paper (previously sent with some other stegotext).
2.2.1 Option 1. Encode stegotext into all answers of an exam paper

Figure 2.2 below is an overview of how the encrypted data is processed through the

Figure 2.2 Overview of Encode – for Option 1.
Encode module for Option 1. – i.e., the stegotext is encoded in all the answers of the exam paper. This encoding will eventually lead to the construction of an exam paper using the Construct Module. Note: A clarification about notation used in section 2.3, “18/19” should be read as “18 or 19” and “2/3” should be read as “2 or 3” below.

The module transforms the stegotext into characters that will eventually be built into all the answers of an exam paper. Step 1 in Figure 2.2 shows the stegotext coming into the Encode module and is processed using bit manipulation techniques. Each exam question can conceal 18/19-bits of data. Step 2 shows the bits being iteratively divided into 18/19-bit blocks. Each of these blocks can be encoded into one exam question. If the stegotext begins with a ‘000’ then it is cut up into a 19-bit block, otherwise it is cut up into an 18-
bit block. As the stegotext is processed into 18/19-bit blocks, some padding might be added to the end to ensure that the entire stegotext can be evenly divided into 18/19 bit chunks. For example, if the stegotext is 24-bits long, then Step 2 will divide the first part into 18/19-bit block and then pad the rest to make up another 18/19-bit block. Note: Due to this padding, the user will have to send the length in bits of the actual stegotext out-of-band so it can be properly decoded on the other end. Step 3 shows that the first 2-3 bits are encoded using the correct answer’s lead letter (the letter to the left of the answer – A, B, C, D, or E). The rest of the bits are split evenly into four 4-bit blocks and encoded with the first letter of the wrong answer. This process can be better explained with a few examples.

For example, encoding for each 18/19-bit block could be based on the tables below:

Table 2.1 Encoding for Option 1., for correct answers

<table>
<thead>
<tr>
<th>Correct Answer</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Pattern</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td>000</td>
</tr>
</tbody>
</table>

Table 2.2 Encoding for Option 1., for wrong answers

<table>
<thead>
<tr>
<th>Wrong Answer</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Pattern</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wrong Answer</th>
<th>N</th>
<th>O</th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
<th>T</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Pattern</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>0000</td>
</tr>
</tbody>
</table>
Consider an encrypted 18-bit stegotext block that has the following bit pattern: ‘00-1110-0010-0100-0001’ (Note: hyphens are provided for readability and are not part of the stegotext). The first two bits are ‘00’ which would be encoded to the correct answer’s lead letter – (A) (Table 2.1). The next sixteen bits are split into four-bit blocks and encoded into the first letter of each of the wrong answers – ‘O’, ‘C’/‘S’, ‘E’/‘U’, ‘B’/‘R’ (Table 2.2). The correct answer is placed in the first (A) position and the four wrong answers are picked such that their first letters correspond to ‘O’, ‘C’, ‘U’, and ‘B’. Note: Some encodings have two character choices for the same bit-pattern. This gives the Construct module choices in picking an answer. For example, for the encoding ‘C’/‘S’ – it would pick an answer beginning with a C or an S – not both.

The Encode module would return ‘AOC/SE/UB/R’ which could then be translated to the GRE antonym example below (this step is done in the Construct module which will be explained in section 2.4).

1. PALATABLE:

(A) Unappetizing \( \rightarrow \) Correct Answer – Lead letter is ‘A’
(B) Odorous
(C) Culinary
(D) Unflavored
(E) Bland

First Letters of wrong answers are ‘O,C,U,B’

Consider now an encrypted 19-bit stegotext block with a different bit pattern: ‘000-0011-0101-1011-0010’ – the first three bits are ‘000’ which would correspond to (E) – lead letter according to Table 2.1. The next four 4-bit blocks are encoded as ‘D’/‘T’,
‘F’/‘V’, ‘L’ and ‘C’/‘S’ according to Table 2.2. The same GRE antonym question/answers would have to reflect this encoding as is done in the example below:

1. PALATABLE:

(A) Delicious
(B) Flavorful
(C) Luscious
(D) Savory
(E) Unappetizing \(\Rightarrow\) Correct Answer – Lead letter is ‘E’

First Letters of wrong answers are ‘D,F,L,S’

2.2.2 Option 2. Encode stegotext to some chosen answers in the exam paper

The Encoding module also offers a second option – encode the bits of the stegotext into certain answers of the exam. This option can be used by Alice (see Figure 1.1) to encode a small reply back to Bob. This option can only encode 2/3-bits per question. The encoding will return a lead letter for each question. Alice will have to pick the answers on the exam sent by Bob, specified by the lead letters even if they are the wrong answers.
Figure 2.3 Overview of Encode – for Option 2

Consider that Alice wants to send back an acknowledgement – some 1 byte reply to Bob. Figure 2.3 above shows the process of encoding this stegotext into chosen answers of an exam paper. Step 1 would convert the stegotext into bit format – suppose it translates to ‘00011001’. Step 2 would divide the bits into 2/3-bits. Notice that the first 3-bits are divided into one block since they correspond to the bit pattern ‘000’. Padding of 1-bit (a ‘0’) is added to the end to make a 2-bit block. Step 3 uses the table below to encode this stegotext. It will return: ‘EDAC’ – Alice will pick answers that correspond to (E), (D), (A) and (C) on the exam paper sent by Bob and leave answers to other questions blank.

Table 2.3 Encoding for Option 2, for chosen answers

<table>
<thead>
<tr>
<th>Chosen Answer</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Pattern</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td>000</td>
</tr>
</tbody>
</table>
Option 2. offers limited encoding – only 2/3-bits per question versus 18/19-bits per question as offered by Option 1. It is used to encode acknowledgements to be sent back. This option bears some discussion since the encoder does not always pick the correct answers – rather it picks the answer that fit the encoding. Due to this fact, it is possible that a large number of wrong answers are picked by the encoder. This might alert some adversary observing the communication between Bob and Alice since the exam answers returned by Alice will consistently contain a large number of wrong answers.

As stated in Chapter 1, one of the main goals of Steganography is to avoid suspicion of covert communications. Since the preceding problem could possibly alert an adversary, HideNQA users might come up with some out-of-band agreement that every other reply starting from the top will contain an encoding. So, for example if Alice replies to Bob in an exam paper which contains ten multiple-choice questions, only five of those questions will contain an encoding. The other five will merely be the correct answers to the exam. So with this scheme, questions 1, 3, 5, 7, and 9 will contain encoding information while the even # questions will just be correct answers to the exam. This is only one example of a mitigating technique that can be used to avoid detection of Steganography, there could be other agreements in place that could change periodically using out-of-band communications.
2.2.3 Latin Squares

The Encode module employs the use of Latin squares to add some level of confusion to the encoding. If the tables described above are used for all answers, an attacker could detect the bits encoded by each character with some dedicated trial-and-error techniques. For example, if A – always stood for ‘00’ (Table 2.1 & Table 2.3) in the correct/chosen answers and ‘0000’ (Table 2.2) in the wrong answers, an attacker can eventually figure out this mapping. To thwart such attacks, HideNQA uses Latin squares during its Encode phase. A Latin square is an n x n array filled with n different symbols [8] where each symbol occurs exactly once in each row and exactly once in each column. In this case, Latin squares are used to change the bit pattern associated with each encode letter. The tables below illustrate this fact. For example, in Table 2.4, ‘A’ is associated with the bit pattern ‘00’ for the first question or the first chosen answer (depending on the Option used to encode). But if the bit pattern ‘00’ is encountered a second time, then the letter associated with it will be ‘E’. Each different question/chosen answer will use a different row in Table 2.4. The user setting up the exam paper or replying back can specify which row should be the starting point for the encoding. If Bob is encoding a secret into an

<table>
<thead>
<tr>
<th>Correct/Chosen Answer Encoding</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Question / Chosen Ans.  -</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td>000</td>
</tr>
<tr>
<td>Second Question / Chosen Ans. -</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td>000</td>
<td>00</td>
</tr>
<tr>
<td>Third Question / Chosen Ans.  -</td>
<td>10</td>
<td>11</td>
<td>000</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>Fourth Question / Chosen Ans. -</td>
<td>11</td>
<td>000</td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>Fifth Question / Chosen Ans.  -</td>
<td>000</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
exam paper (Option 1), he can specify that the Encode module start with row three (see arrow above) of the Table 2.4 above. So, for the first question, A will be ‘10’, ‘B’ will be ‘11’, ‘C’ will be ‘000’, ‘D’ will be ‘00’ and E will be ‘01’. The second question will then use row four, third question will use row five and so on. The fourth question will go back to the first row in the Latin square array and use row one. The rows are continuously used in a round robin manner as long as there’s encoding to be done. Table 2.5 and 2.6 below are one large table split vertically down the middle so it can fit on the page. They illustrate the Latin square array used to encode wrong answers in the exam paper.

Table 2.5 Latin square array for wrong answers

<table>
<thead>
<tr>
<th>Wrong Answer Encoding</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Question</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Second Question</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>Third Question</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td></td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
</tr>
<tr>
<td></td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
</tr>
</tbody>
</table>
Table 2.6 Latin square array for wrong answers

<table>
<thead>
<tr>
<th>Wrong Answer Encoding</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Question</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Second Question</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
</tr>
<tr>
<td>Third Question</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

The column indices for Tables 2.5 and 2.6 are representative of the four-bit pattern for each integer. So, 0 is ‘0000’, 1 is ‘0001’, 2 is ‘0010’, 15 is ‘1111’ and so on. When Bob would like to encode a secret in an exam paper using Option 1., he can specify which row of Table 2.5 and Table 2.6 should begin the encoding. Table 2.5, 2.6 are used similar to Table 2.4 – iteratively in a round robin manner starting from the row specified by Bob.

2.3 Construct Module

Option 1 of the Encode Module provides an encoded list that will be fed into the Construct Module to result in a text file containing multiple choice questions and answers.
that represent the encoding. Figure 2.4 below illustrates the Construct Module’s overall design. This Module will access a database that will store the questions and answers.

![Figure 2.4 Overview of Construct Module](image)

The database backend has to be configured carefully so that each question has one corresponding correct answer and several corresponding unique wrong answers. The wrong answers have to be varied such that they begin with all the letters of the alphabet – A through Z. Exam/Test types that lend themselves to have many wrong answers beginning with all the letters of the alphabet are the ideal choices to fit HideNQA’s requirement. Examples of such exams are the GRE Antonyms exam component, Chemistry multiple-choice exams, English Synonym exams, etc. Each question must have multiple wrong answers that begin with letters ‘A’, ‘B’, ‘C’, all the way to ‘Z’. In HideNQA’s design, each multiple-choice question has one correct answer and four wrong
answers. For example, the following encoded text might come in from the Encode Module to the Construct Module: ‘CAAAA’. In this example, the first question has to have the correct answer’s lead letter be the letter ‘C’. Each of the wrong answers for this question has to begin with an ‘A’. This is the reason why the database must contain plenty of choices for wrong answers. In this case, the database must have four unique wrong answers beginning with the letter ‘A’. If the database only had 3 wrong answers beginning with the letter ‘A’, the last wrong answer cannot be constructed and the Construct Module would return something like this:

1. PALATABLE:

   (A) Arsenal
   (B) Arsenic
   (C) Unappetizing \(\rightarrow\) Correct Answer – Lead letter is ‘C’
   (D) Arrythmia
   (E) A – “NONE FOUND”

   It would return the first letter – ‘A’ in this case – and the message “NONE FOUND”.

   The user will then have to manually find a wrong answer for PALATABLE with the first letter starting with ‘A’. Fortunately, as stated in Section 2.3.1, several of the bit patterns map to two letters which increases the choices that can be made by the Construct Module so that a scenario like the above example would be a rarity.
2.4 Compress and Encrypt Modules

HideNQA uses library calls to implement compression and encryption. The reason compression and encryption have been split into separate modules of their own is to stay true to the overall modular design of the tool. HideNQA uses lossless compression. Compression is mainly performed on the stegotext to increase the stegotext to covertext bit ratio. Without compression, the HideNQA tool can generally hide 2 to 3-bits in each correct answer plus 4-bits in each of the four wrong answers. Approximately \((2 + 4*4)\) 18-bits can be hidden in each question of a multiple-choice exam. By compressing the data in the first place, there are essentially fewer bits to hide in each question. Note: If the stegotext to be hidden is a small string, then compression will not be used. Compression in this case might actually make the data string bigger. The Compression module will compress the string and only return the compressed version if it is indeed smaller than the original. Otherwise, the original string will be returned for Encryption. Further details about the exact compression algorithm will be discussed in Chapter 3.

Encryption is used to enhance security in case HideNQA’s encoding is ever attacked and compromised. Encryption will use a symmetric key to encrypt. This symmetric key has to be transmitted to the decoding party using an out-of-band channel.
2.5 Decode Module

The Decode Module starts the work of the Reveal task flow – bottom half of Figure 2.1. The user enters either all the answers of an Exam paper (Alice would do this when she receives an exam paper from Bob) or particular answers of an exam paper (Bob would do this when he receives an acknowledgement back from Alice) as input to the Decode Module. This module will then return the encrypted and/or compressed stegotext as output. When entering all the answers of an exam paper, the user will enter the lead letter of the correct answer and then the first letters of the four wrong answers for each question. Consider the exam paper below with just two questions:-

1. PALATABLE:

(A) Delicious
(B) Flavorful
(C) Luscious
(D) Savory
(E) Unappetizing \(\rightarrow\) Correct Answer – Lead letter is ‘E’

First Letter of wrong answers are ‘D,F,L,S’

2. BAROQUE:

(A) Ornate
(B) Simple \(\rightarrow\) Correct Answer – Lead letter is ‘B’
(C) Universal
(D) Dull
(E) Boat-Like

First Letters of wrong answers are ‘O,U,D,B’

The encoding that would be entered as input to the Decode Module would be:

‘EDFLSBOUDP’.
When entering particular answers to an exam paper the user just enters the lead letter of each chosen answer. If the ‘A’ was circled as the answer for the first question above and ‘D’ was circled as the answer for the second question above, then the user would enter ‘AD’ into the Decode Module. The Decode Module will also have to know if it is decoding a full exam paper or particular answers to an exam paper and the number of bits that were previously encoded. The number of bits previously encoded would have been sent through out-of-band communication.

2.6 Decrypt and Uncompress Modules

The Decrypt Module will ask for a symmetric key (the user will have received this through some out-of-band communication) in order to decrypt the encrypted stegotext. Given the correct symmetric, decryption is a straight forward operation. The stegotext returned from the Decrypt Module may or may not also be compressed. As stated in Section 2.2, compression may not always provide a smaller string. The Uncompress Module will only be called if the string was previously compressed.
Chapter 3

HIDENQA TOOL IMPLEMENTATION

3.1 Introduction

This chapter will explain the details of HideNQA Tool’s implementation. Section 3.2 specifies the development platform and language used. Section 3.3 describes the graphical user interface (GUI) for this tool. Section 3.4 elaborates on relevant functions and data objects used in the implementation of the different modules. Finally, this chapter will conclude with a discussion on the database backend used in HideNQA’s implementation in Section 3.5.

3.2 Development Platform and Language

HideNQA tool has been developed on a Windows 7 platform in Python 2.7 [9]. Python is a high-level programming language and free to use. HideNQA has been implemented in Python since its programs can run on Windows, Linux/Unix and Mac OS X. The idea is to be able to run the HideNQA tool on eventually all four platforms without any dramatic changes to the underlying implementation. Regardless of the OS, users can install and run the HideNQA tool on their systems.

Python emphasizes code readability. A tool implemented with Python can later be enhanced or changed by others without having to decipher cryptic code. Since HideNQA
is a preliminary implementation of a new paradigm, the probability is high that its code will be reviewed and changed in the future.

Python has a comprehensive standard library and is highly extensible. New modules can be written in C, C++, Python, etc. HideNQA tool takes advantage of this extensibility and uses such an extended package for ease in bit manipulation (used in the Encode and Construct modules). Bitstring [10] is an open-source python module that makes the creation, manipulation and analysis of binary data as simple and natural as possible. This particular module has been written completely in Python.

Python is also a multi-paradigm programming language. It supports object-oriented programming as well as structured and functional programming. HideNQA’s GUI implementation takes advantage of the object-oriented programming paradigm and uses classes to develop the user interface. The tool uses the structured/functional programming approach when implementing the Compress, Encrypt, Encode and Construct modules. Python’s flexibility in allowing multiple paradigms allows the implementation of the HnQA tool to be equally flexible.

3.3 Graphical User Interface Design and Implementation

The HideNQA tool provides a GUI (graphical user interface) for the convenience of its users. The GUI has been constructed using Qt Designer [11] and PyQt [12]. Qt Designer
assists in building GUI’s using Qt components. PyQt are a set of python bindings used to incorporate GUI built using Qt Designer into Python applications. A well-constructed graphical user interface will help the user to easily use the tool in order to hide or reveal steganographic data within exam papers. It provides the necessary guidance for the user to be able to use the tool with just some basic knowledge of HideNQA’s features. The interface is laid out in such a way as to allow the user to efficiently enter the information needed by the tool. The uncluttered dialog boxes prompt the user for the required information only as needed. The graphical user interface additionally provides the advantage of validating the inputs entered by the user. This aids in security and efficient error-free running of the tool.

Figure 3.1 shows the main dialog box which opens up when the user first executes the tool. As can be seen, it provides two lines by way of introduction and prompts the user to

![Figure 3.1 HideNQA’s Main dialog box](image)
click on one of the three buttons. The Hide button will lead the user into a task flow if they would like to hide a secret within an exam paper. The Reveal button will take the user into the task flow where they can reveal a previously hidden secret by entering the chosen answers in the exam paper. Figure 3.2 below shows the dialog box that pops up when the “Hide” button in Figure 3.1 is pressed. The user is prompted to enter the data to be hidden (the stegotext), a key to encrypt the data and choose an option – data can be

![Figure 3.2 Hide task flow](image-url)
hidden in the correct and wrong answers of an exam or just within chosen answers of an exam (this depends on who is sending the steganography data as explained in Figure 1.1 in Chapter 1). Figure 3.3 below shows dialog box that pops up when the “Reveal” button in Figure 3.1 is pushed. This dialog leads to revealing secret information from all answers or chosen answers in an exam paper. The user will be prompted for the letter
choices (as explained in Chapter 2, Section 2.5), the key used to previously encrypt the stegotext, the total number of bits previously encoded and which option was used previously to construct the exam paper. Some of this information will be known to the user via out-of-band communication.

Figure 3.4 and 3.5 below, show the different extra choices that get revealed when the user
toggles between Option 1 and Option 2. Note: only the Hide dialog box is shown, but the same explanation applies to the Reveal dialog box as well. As explained in Chapter 2, section 2.2, the first option uses two Latin square arrays and the second only...

![Hide Dialog Box](image)

**Figure 3.5** Hide task flow – Option 2
uses one Latin square array. Also note in figures 3.4 and 3.5, that the row(s) to be used in the Latin square arrays are picked using spin boxes.

These spin boxes will only allow the user to pick numbers between 0-4 for the Correct answer Latin square and 0-25 for the Wrong answer Latin square. This design eliminates...
any possible out-of-range user errors in picking rows for the Latin squares. Figure 3.6 illustrates that the GUI is capable of sanitizing all the input when the user clicks the “Ok” button. The GUI ensures that all the text boxes have received the right types of input and that one of the options has been picked. If the input does not pass the sanity check, an information dialog box pops up alerting the user to remedy the problem. As seen in Figure 3.6, the user neglected to enter an encryption key and is prompted to enter a key of at least length 5.

Once the Hide or Reveal dialog input has been sanitized, the task flow continues. In the case of the Hide task flow the results are shown in a separate dialog box. Figure 3.7 and Figure 3.8 show the output when the stegotext is “secret” and Option 1 (hide the stegotext in all answers of an exam) were chosen in the Hide task flow.
The first dialog box (Figure 3.7) indicates how many bits were encoded. In this case, the number encoded is 48. This information will have to be sent using an out-of-band channel to the receiver of this stegotext. When the user clicks “OK” on this dialog box,

![Figure 3.8 Output for Hide task flow - Option 1](image1)

the next dialog box pops up with the exam in a read-only text browser as shown in Figure

![Figure 3.9 Prompt to save file](image2)
3.8. The “Save As” button below the text browser can be clicked by the user to save the file. When the user pushes this button another dialog box pops up (as shown in Figure 3.9) to ask for a filename to use. Once the user enters a filename, the tool will create a directory named “hnqaencodes” (if it doesn’t already exist) in the current working directory. It will save the file using the filename specified by the user in this directory and return a successful message dialog box as shown in Figure 3.10.

![Figure 3.10 File successfully saved message](image)

Figures 3.11 and 3.12 show the output dialog boxes for the Hide task flow, Option 2. The stegotext used in this case was the string “abc” and the number of encoded bits turns out to be 24. Also, notice that Figure 3.11 shows which replies the user should pick on a

![Figure 3.11 Output for Hide task flow - Option 2](image)
given exam to encode the stegotext “abc”. This file can also be saved for later reference.

The “Save As” button will behave exactly the same way as explained above.

![Output for Hide task flow - Option 2](image)

Figure 3.12 Output for Hide task flow - Option 2

Figures 3.13, 3.14 show the results from the Reveal task flow for Option 1 and Option 2.
The Figure 3.13 shows that the stegotext hidden within was the string “secret”. Figure 3.13 correlates to Figure 3.8, the exam shown in 3.8 is hiding the string “secret”. Figure 3.14 shows that the stegotext hidden within was the string “abc”. Figure 3.12 correlates to Figure 3.14, the multiple choice letters in 3.12 when picked on an exam would hide the string “abc”.

3.4 Functions and Data Objects

This section is important to any future design and source changes made to the HideNQA tool. It describes some of the important functions and data objects used in the compression, encryption, encoding and construction modules of this tool. This section
creates a record of what has been used so it may facilitate future changes in design and implementation.

3.4.1 Compression Module

The compression module uses the zlib 1.0 [13] compression library in Python 2.7. It is a lossless data-compression library. It is also a free, general-purpose compression library that can be ported to other operating systems and platforms. The HideNQA tool tries to compress every stegotext input before the main processing begins. It compares the compressed string with the original string and only keeps the compressed version if it is smaller than the original. Therefore, when the stegotext is extracted on the other end, it may or may not be compressed depending on the size and nature of the string. The decompression function checks to see if the string was previously compressed and only decompresses if this is true.

3.4.2 Encryption Module

The encryption module uses the ARC4 package from the built-in Python cryptography toolkit in Python 2.7. ARC4 (popularly known as RC4) is a widely used software stream cipher. It is a simple and fast software cipher. This cipher is well suited to the needs of the HideNQA tool since it deals with varying stegotext string sizes every time it is used to hide data. The key size for this cipher can be variable but the HideNQA tool imposes
a minimum of five characters restriction on the user to increase security of the encryption.

3.4.3 Encode Module

Latin Squares are employed by HideNQA to add a level of confusion to thwart any attacks that might try to uncover the stegotext from the original exam. Latin Square are explained in Chapter 2, Section 2.2.3 as n x n arrays. The tool implements these two-dimension arrays simply as a Python List object. A Python List can be compared to an array object in C or C++. This saves space but is more time consuming since the List object is constructed and shifted to the appropriate row specified by the user at the beginning of each Hide or Reveal operation by the tool. Additionally, the List object is shifted by one for each question encoded/decoded. As stated earlier, this representation saves some space but can end up taking time when encoding longer stegotext strings.

3.5 HideNQA’s Database Backend

HideNQA’s exam questions and answers are stored in a MySQL database backend. Python uses the module MySQLdb to interface with databases such as MySQL, PostgreSQL, Informix, Microsoft SQL Server 2000 and Oracle. A separate database API module has to be downloaded for each database type that needs to be accessed. MySQL was chosen for this project as it is open-source released under the GNU General Public
License. In addition, it is fast, reliable and free for most uses on all supported platforms including Windows, UNIX, and Mac OS. Since there is a tentative future plan to port this tool to UNIX and the Mac OS, using MySQL would offer the advantage of little to no change in the database backend. It is reasonably secure and includes encryption/decryption functions as well as other security measures.

The database backend has been primed with questions (GRE Words) and corresponding correct answers (antonyms of the GRE Words). In addition, the database also contains wrong answers for the questions. The wrong answers are carefully chosen such that they begin with all letters of the alphabet from A to Z. The use of tuples in choosing the first letter of the wrong answers (see Chapter 2, Sections 2.2 and 2.3 for more detail) increases the probability of finding a wrong answer in the database. Nevertheless, the database should be filled with plenty of wrong answers starting with letters A to Z. Technically each question could have all four of its wrong answer choices start with the same letter. In such a case, the tool would query for four distinct wrong answers starting with the same letter. In the event, the query fails and an answer was unable to be found, HideNQA tool will print out the letter and “NONE FOUND” message. The choices here are to insert a new distinct wrong answer that starts with such a letter into the database and retry the tool or have the user find a wrong answer that starts with the given letter.
Chapter 4

ANALYSIS AND FUTURE WORK

4.1 Analysis

As stated in Chapter 1, Section 1.3, one of the main ideas behind HideNQA is to be robust and untraceable. The traceability of the stegotext hidden by HideNQA could happen mainly through the compromise of any out-of-band communication between the sender and the receiver. If this communication is managed covertly and correctly, the exams constructed by HideNQA are basically “noiseless”. The exams maintain their integrity in both print and electronic form. Stegotext to covertext ratio is low compared to other forms of steganography although it is higher than most linguistic steganography techniques.

Table 4.1 below lists the stegotext to covertext ratio for the HideNQA tool to give some perspective on how much data can be steganographically transported using this method. The first column lists the stegotext to be hidden in number of bits, the second column lists the covertext in terms of the number of questions these bits would take if using Option 1 (See Chapter 2, Section 2.2) and the third column lists the covertext in terms of how many questions these bits would take if using Option 2 (See Chapter 2, Section 2.2).
Table 4.1 HideNQA stegotext to covertext ratio

<table>
<thead>
<tr>
<th>Stegotext (# of bits)</th>
<th>Covertext (Option 1)</th>
<th>Covertext (Option 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>54</td>
<td>3</td>
<td>27</td>
</tr>
<tr>
<td>128</td>
<td>8</td>
<td>64</td>
</tr>
</tbody>
</table>

Note: The above table shows an approximation of how many questions are needed in the worst-case scenario for either Option 1 or Option 2. As noted in Chapter 2, Section 2.2, Option 1 can encode 18/19 bits per question and Option 2 can encode 2/3 bits per question. The computations are based on Option 1 only encoding 18-bits and Option 2 only encoding 2-bits.

Table 4.2 below takes an excerpt from Desoky’s [6] research and shows the bitrate comparison between the HideNQA tool’s steganography bitrate and other common linguistic steganography methods. Bitrate here is defined as the size of stegotext relative to the covertext used. The two other linguistic steganography methods that are compared here are: Spammimic [2] and NICETEXT [14]. Spammimic is a tool that hides a
message inside a covertext that looks like spam. [2] Nicetext is an algorithm for converting stegotext into essentially normal looking text. [14].

Table 4.2 Bitrate comparison of Linguistic Steganography methods

<table>
<thead>
<tr>
<th></th>
<th>HideNQA</th>
<th>Spammimic</th>
<th>Nicetext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitrate</td>
<td>3.6%</td>
<td>0.9%</td>
<td>0.26%</td>
</tr>
</tbody>
</table>

As can be seen in Table 4.2 above, a larger amount of stegotext can be hidden inside a given covertext with HideNQA versus other methods.

4.2 Future Work

Future work to be completed on HideNQA tool is as follows:

1. Change the current database hook into HideNQA – Currently the database hook is hard-coded into the tool. It would be far better to have a flexible hook so that an administrator can connect a database to the tool upon demand. Something like this could be achieved by having an administrator’s user-interface to the tool, where they can connect a database and give its security information to the tool. Databases with different questions could then be attached to the tool as long as they follow the same basic question/multiple-choice answer format required by the tool. For example, the tool would be able to encode stegotext using a GRE antonym exam database and later encode another stegotext using a Chemistry
multiple-choice exam. It would be a matter of hooking up the tool with a GRE antonym database and then later hook it up to a Chemistry exam database.

2. Porting and consequent testing of the tool in Unix, Linux and Mac OS environments. Theoretically, the porting should not take much time since Python is meant to be a language that can be used across platforms. The only caution is the fact that the HideNQA tool has a user-interface and the porting of the user-interface part might be complicated.
BIBLIOGRAPHY


