Global Information Assurance Certification Paper

Steganography using the Chess PGN Standard Format

GIAC Security Essentials Certification (GSEC)

Practical Assignment
Version 1.4b
Option 2 – Case Study in Information Security

Brian Lange, CISSP
SANS Rocky Mountain 2004
July 2004

Table of Contents

Abstract

This paper provides a practical example of how to conceal information inside a chess game using steganography techniques in three snapshots. The before snapshot introduces the concepts of how to protect sensitive information, discusses why the Chess PGN Standard was chosen, explains the security goals for this project, and reveals the risks of using a macros program in a word processing document. The during snapshot shows how a binary string is converted into a chess game and what steps are taken to address the risks that are discovered along the way to make the hidden information more secure. The after snapshot demonstrates how to install and run the programs.

A single chess game is able to hold about 2 ½ pages of text information and still look inconspicuous to the casual observer. The program provides for strong authentication, good encryption, and several methods for hiding data. Running the program can take some time for larger files and it is possible for the enemy to potentially destroy the hidden data by removing text formatting or changing the order of the chess games. There are many other ways to hide information, but the principles that are discussed in this paper can be used for various applications of steganography.

1.0 Before Snapshot

1.1 Cover, concealment, and camouflage

There are some situations where simply scrambling or encrypting your data is not sufficient. A spy sending encrypted messages to a foreign country would arouse suspicion within the host country. A businessman working overseas may be susceptible to eavesdropping by other businesses and government agencies that would focus in on encrypted transmissions to find sensitive information. A hostage forced to read a ransom note would endanger their life if they sent obviously cryptic messages. If you need to conceal the fact that you are communicating any message at all, it is necessary to safeguard the information using more than just cryptography.

Protecting data can be compared to protecting a soldier. In the military, there are three methods used to protect soldiers from the enemy: cover, concealment and camouflage.[1] "Cover" gives protection from bullets, bombs and shrapnel. It can take the form of bunkers, helmets, flak jackets, tanks, walls, or other solid objects.[2] The way "cover" protects a solider is similar to how cryptography protects information: it changes data into random-looking characters so that the information is not exposed to the enemy. Direct fire from the enemy is the equivalent to brute-force guessing of the key needed to decrypt a message. The better the cover, the less chance there is of getting hit; the better the algorithm or the larger the keyspace, the less chance there is of the secret message being

[1] United States. Department of the Army. US ARMY Field Manual 21-75. 1984, 1-1.
[2] FM 21-75 1-2.

discovered. Cryptography won't camouflage data; it just scrambles it so no one can guess what the original message is.

"Concealment" is anything which hides you from enemy observation, but doesn't necessarily protect you from enemy fire. This can take the form of foliage, camouflage nets, or even shadows.[3] Steganography is a method of hiding information from enemy observation just like "concealment" can hide a soldier. Using today's technology, it is possible to hide data inside digital pictures, songs, or even text files. Although steganography helps prevent the enemy from detecting your message, if they find out what you are doing, there is nothing to stop them from reading your communications.

"Camouflage" is anything you use to keep yourself or your equipment from looking like what they are. This can involve painting your face or equipment a different color to match your surroundings.[4] Camouflage is related to steganography because it deals with the context of the hidden message. Holding a piece of paper in front of your face would provide concealment, but not camouflage. Likewise, sending your boss a digital picture that contains concealed information doesn't provide any camouflage unless this is a regular practice and wouldn't arouse suspicion. Camouflage also will not provide any protection of the actual data.

The best way to protect sensitive data is to use all three principles together. Cryptography is used to prevent the data from being understood; steganography is used to keep the data from being detected; and camouflage is used to make the message blend into its environment.

Military protection	Purpose	Data protection	Purpose
Cover	Protects from enemy fire	Cryptography	Protects from brute force attack
Concealment	Hides from enemy observation	Steganography	Hides data in a different form
Camouflage	Disguises yourself or equipment	Message's context	Hidden data blends into environment

Figure 1: Comparison between military and data protection.

1.2 Using chess for steganography

Methods of hiding data using steganography are only limited by one's creativity and imagination. This paper describes the creation of a program to hide as much data as possible inside a chess game. The Portable Game Notation (PGN) standard is used because it is a universally accepted text-based format for documenting chess games. It is easily read by both people and computers and

[3] FM 21-75 1-3.
[4] FM 21-75 1-3.

is the de facto standard among users.[5] Many chess databases are stored in this format and provide a good camouflage for hiding information. A description of the specifications for the PGN standard format can be found at http://pgn.freeservers.com/Standard.txt.

During World War II, the United States banned various types of international mail for fear that they may contain hidden information. This ban included crossword puzzles, instructions for knitting, and correspondence chess games.[6] There is no evidence that hidden messages were ever sent using chess moves, but it was still done as a precaution. There are even some state prisons that forbid correspondence chess games for this reason.[7] One drawback to hiding data using a correspondence chess game is that there are only a few places available on the postcard for hiding data (see Appendix A), so only very short messages could be sent using this method. In order to hide large amounts of text data, it is necessary to make use of the entire chess game, instead of just a couple of chess moves on a postcard.

There are several goals for this project. One goal is to investigate how much data can be hidden inside a chess game using the PGN standard format. Since an entire chess game is being used, not only are chess moves used for hiding information, but also:

- the number of chess moves for the game,
- who was the winner of the game,
- what type of chess opening was used for the game,
- and the values in the PGN header.

The PGN header contains several options that can be sent with a chess game[8]. The following options are used for the PGN header:

- event where the game was played,
- city and country where the game was played,
- date of the chess game,
- white player's name,
- black player's name,
- tournament round the game was played in,
- result of the chess game,
- white player's ranking,
- black player's ranking,
- and the type of opening move that was used.

[5] Weeks, Mark. "Portable Game Notation (PGN)," About Chess. 12 Oct. 2002. URL: http://chess.about.com/library/weekly/aa101202a.htm (12 July 2004).
[6] Kahn, David. The Codebreakers. 2nd ed. New York: Scribner, 1996, 515.
[7] Campbell, J. Franklin. "USCF Abandons Prison Inmates," The Campbell Report. 23 Dec. 2002. URL: http://www.correspondencechess.com/campbell/articles/a021223.htm (12 July 2004).
[8] Edwards, Steven J. "PGN Specification and Implementation Guide." 8.1-9,1. 12 Mar. 1994. URL: http://pgn.freeservers.com/Standard.txt (12 July 2004).

Each header is used to hide a certain amount of information along with the chess game.

A second goal is to make use of cryptographic principles to improve the security of the hidden information. There are two basic methods for encrypting a message: confusion and diffusion. Confusion is accomplished by substituting the original data with a chess move or a portion of the PGN header. This helps hide the association between the original data and the encrypted chess game. Diffusion is achieved by distributing the original data throughout the chess game and the PGN header. This prevents a casual observer from noticing any obvious patterns that may alert him to a hidden message.[9] Other cryptographic principles are discussed as they are used during the process of creating this algorithm.

A third goal is to make use of steganography to hide data inside the chess game itself. This includes using additional white-spaces, font formats, and paragraph formats. Since spaces are very narrow, it is difficult to determine whether or not there are one or two spaces between words. This allows for the ability to hide information. If there is only one space between two words, then a "0" bit is encoded; if there are two spaces, then a "1" bit is encoded. Spaces are also convenient for hiding information by changing the format of its font characteristics. For example, if a space is "bold", a "1" bit is encoded; if the space is not "bold", then a "0" bit is encoded. Since formatting of a white-space won't appear on the document, they will go unnoticed by a casual reader. Paragraph formats can also be slightly altered to hide information. For example, if there are no additional pixels between two lines, then a "0" is encoded; if there is one additional row of pixels between two lines, then a "1" is encoded. Since the width of a pixel is very small, any variances would not be easily noticed by someone viewing the document.

1.3 Risks of using Visual Basic macros

For this project, Visual Basic Macros are utilized, which are available with Microsoft Word 2003. There are several reasons for using these tools. First, Microsoft Word is commonly used in the marketplace and is available to most users. Second, Microsoft Word is a robust word processor that has many formatting options that can be used for steganography. Third, Visual Basic is tightly integrated with Microsoft Word. This allows for complex computing and formatting changes to a document in order to hide information.

There are some risks associated with running macros. Macro viruses have been created that can cause problems on your machine. To help limit this exposure, untrusted documents should be scanned with an anti-virus software application to ensure it is safe.

[9] Schneier, Bruce. Applied Cryptography. 2nd ed. New York: John Wiley & Sons, Inc., 1996, 237.

Second, be sure to have the macros security setting configured to a safe level. In Microsoft Word, go to the Tools menu, and pick “Macros” and then “Security”. On the “Security Level” tab, you are given several options for how to treat macros. You should select at a minimum the “Medium” level, which allows you to choose whether or not you want to enable macros for this document.

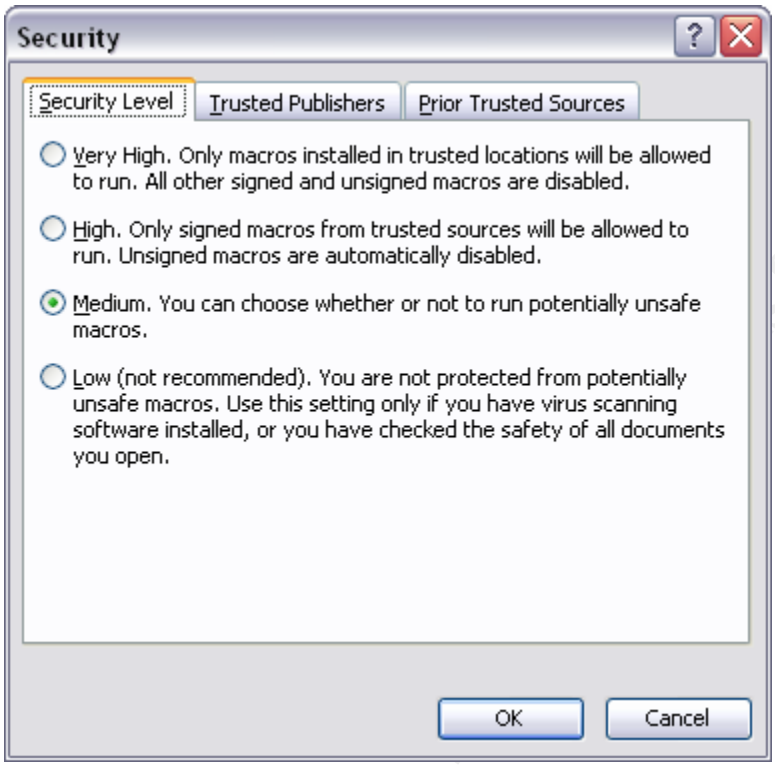

Figure 2: Screen print of the Macros Security window, Security Level tab

Third, on the “Trusted Publishers” tab on the Macros Security window, clear the “Trust all installed add-ins and templates” checkbox. This tells Word to apply security warnings to all files, including add-ins and templates based on the current security level. Further protection can be provided by digitally signing the macros, keeping a list of trusted sources, and running the macros from a trusted location.[10]

[10] “General Office Security”. MSDN Library. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdsdk/html/sdconSecurityGeneralOfficeSecurity.asp (12 July 2004).

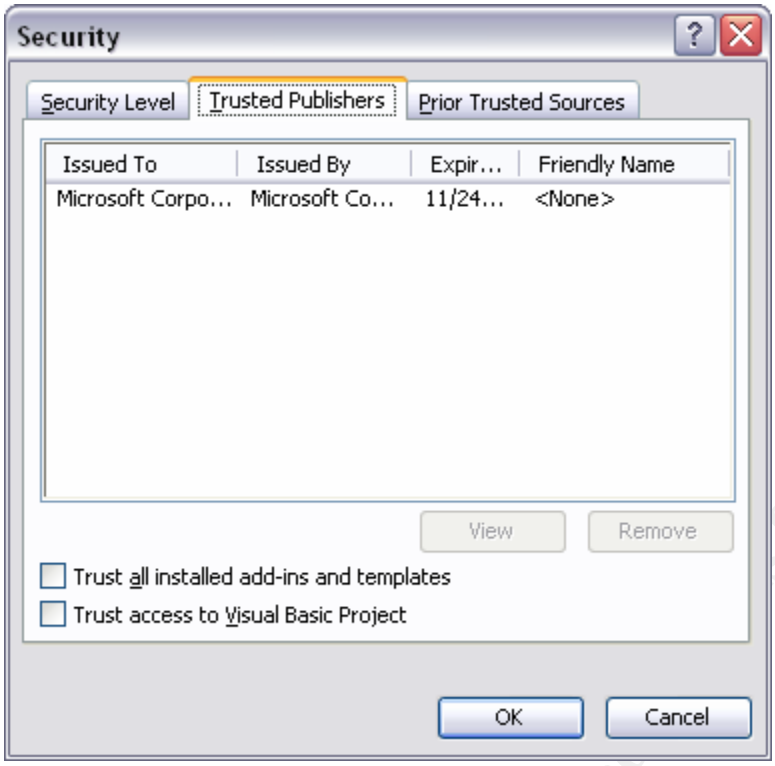

Figure 3: Screen print of the Macros Security window, Trusted Publishers tab.

2.0 During Snapshot

2.1 Encoding binary into chess moves

The first step in creating an encrypted chess game is to convert the plaintext into binary. This is done by simply converting ASCII characters into their binary values. For example, the letter “q” has an ASCII hex value of “71”, which has a binary value of “1110001”. Notice that the ASCII value for capital letters has different values than their lower-case counterparts. In this case, the letter “Q” has an ASCII hex value of “51”, which has a binary value of “1010001”.

Each ASCII character is padded on the left with zeros so that it always has seven binary numbers. This makes it easier to decrypt back into text because you always know that each ASCII character is made from seven binary numbers. The result of the conversion from the plaintext is a string of 1’s and 0’s. Only seven out of the eight bits are used because the eighth bit (on the very left) is always 0 for printable ASCII values. Using only the seven significant bits makes the binary string 12.5% smaller. This reduces the amount of information that needs to be hidden and lessens the cycle time for the algorithm. It also increases security because someone trying to crack the binary values would otherwise know that every eighth bit is a “0”.

The next step is to determine what valid moves are available for the player to make. For this example, let's say there are only four valid chess moves for the white player: a3, a4, b2 and b4. Based on these moves, one of four different binary values can be encoded.

Chess move	Binary value
a3	00
a4	01
b3	10
b4	11

If the first two bits of the binary string have the value "10", then the next chess move chosen would be "b3". This move is written as part of the chess game and two bits of information are encoded. Let's say that on black's turn, there are eight possible moves that can be made: a6, a5, h6, h5, Nc6, Nf6, Bf5, and Qd7. Based on these moves, one of eight different binary values can be encoded.

Chess move	Binary value
a6	000
a5	001
h6	010
h5	011
Nc6	100
Nf6	101
Bf5	110
Qd7	111

In this case, it is possible to encode three bits of information because there are more possible moves for the player. As the number of possible moves increases, the number of bits that can be hidden also increases. If the next three bits of the binary string have the value "101", then the next chess move chosen would be "Nf6". This process would continue until the end of the binary string is reached. The result is a string of chess moves that contain the hidden information.

1. e4 d5 2. exd5 Qxd5 3. Nc3 Qa5 4. d4 e5 5. dxe5 Bb4 6. Bd2 Nc6 7. Nf3 Bd6 8. Bc1 Bxe5 9. Kd2 b6 10. a4 g6 11. Nxe5 Qd5+ 12. Nd3 Bf5 13. f3 h6 14. Rg1 Rd8 15. Ne2 Ra8 16. c4 Bc8 17. Ke1 f5 18. Rb1 Nd8 19. Bd2 Kf8 20. Bc3 Ba6 21. Ndf4 Qa5 22. h3 Qe5 23. Ba5 Rc8 24. Qd2 Qb5 25. Ne6+ Ke8 26. Nxc7+ Kf8 27. Qc3 bxa5 28. Na8 Nf6 29. Rd1 Qe8 30. Kf2 Qxe2+ 31. Kxe2 Rb8 32. Rd5 Rb3 33. Qe3 Rb7 34. Qe7+ Rxe7+ 35. Kd1 g5 36. Bd3 Rb7 37. Ke2 Rb8 38. Bc2 Kf7 39. Bd3 Nc6 40. Rxf5 Ke7 41. Rd1 Kf8 42. Rf4 h5 43. g4 Ne7 44. Rb1 Kg8 45. Bf5 Nxf5 46. Nc7 Ne3 47. Na8 h4 48. Kd3 Re8 49. b4 Bc8 50. Ke2 Nexg4+ 51. Kd2 Re4 52. Rf5 Nf2 53. Kc2 Be6 54. Rxf6 Re3 55. bxa5 Ra3 56. Re1 Rb3 57. Kd2 Rb2+ 58. Kc3 Rb7 59. Rf8+ Kxf8 60. Re2 Rb1 61. Kd2 1/2-1/2

Figure 4: Sample output of plaintext that has been hidden using a chess game.

One of the security benefits of this algorithm is that it encodes data in variable-sized blocks. Depending on the current setup of the chess board, a move can signify anything from an eight bit block to none at all. This makes steganalysis, the science of discovering and uncovering a hidden message[11], more difficult because there isn't a one-to-one correlation between the moves in the chess game and the number of bits that are encoded.

One problem with the algorithm described so far is that it does not prevent anyone from determining what the actual hidden information is because there is no key needed. If you suspect there may be a chess game that is hiding information, then all that is needed to do is run the game through the algorithm and see what comes out. Additional steps are needed to provide "cover" for the data.

2.2 Using vectors and passwords

If the procedure for creating the list of chess moves is static, then it would be easy for someone to re-create the original message. You would simply determine the list of valid moves, see which move is chosen on the chess game, and find the binary value that the chess move corresponds to. If this process were repeated for each move in the game, the entire message would be discovered.

If the procedure for creating the list of chess moves is dynamic, then it would be much more difficult for someone to determine the bits that are encoded for each chess move. If the set of priorities for a certain game are to check the bishops first, followed by the knights, pawns and the queen, then the list of eight valid moves would look like this:

Chess move	Binary value
Bf5	000
Nc6	001
Nf6	010
a6	011
a5	100
h6	101
h5	110
Qd7	111

If the wrong priorities for the chess pieces are used, then the binary values would not match what is encoded:

[11] Cole 188.

Chess move	Correct binary values	Wrong binary values
a6	011	000
a5	100	001
h6	101	010
h5	110	011
Nc6	001	100
Nf6	010	101
Bf5	000	110
Qd7	111	111

If someone wanted to try and decode the message, they would need to have the correct list of prioritized pieces to be successful.

Not only can the pieces be prioritized, but the way the individual pieces move can be prioritized. For example, a King can move in eight different cardinal directions: N, S, E, W, NE, NW, SE and SW. Each of these directions can be given a different priority when checking for valid chess moves.

Based on the chess pieces and their moves, the total number of possible priorities that can be created are:

(White pieces) * (Black pieces) * (Rooks) * (Knights) * (Bishops) * (Queens) * (Kings)
(16!) * (16!) * (4*4!) * (4*8!) * (4*4!) * (2*8!) * (2*8!)
(20922789888000) * (20922789888000) * (96) * (161280) * (96) * (80640) * (80640)
$= 4.231 * 10^{45}$, which is approximately 2^{151}.

This is the equivalent of having a 151-bit key, which is very strong. If you were to try and perform brute-force guessing to determine what the value of the key is and were given one million guesses per second, it would take almost 7 sextillion times the *age of the universe* to only get through 50% of the number of possible values!

This list of pieces and move priorities serve as an initialization variable or an initialization vector. A vector is something that helps make each encrypted message unique, even if the same text is being encoded. In most cases, the vector is a small random value and discarded once used.[12] In this case, the vector is a large and deliberate value (See Appendix B) and is referred to during the execution of the algorithm when determining valid chess moves.

The way the algorithm uses the vector is similar to how a cryptographic algorithm uses what is called a “secret key”. A secret key is a value which is used for both encoding and decoding the data.[13] If the key value is not known, then decryption is impossible. A strong key has a large number of bits and a large keyspace. The keyspace is the number possible values that can be used for encrypting and

[12] Schneier 194.
[13] Cole, Fossen, et al. Track 1.4 - Secure Communications. Vers. 2.2. Track 1 – SANS Security Essentials and the CISSP 10 Domains. 2004, 50.

decrypting data.[14] For this algorithm, the key length is 336 bits, but the keyspace is only 151 bits. Since the vector also functions as a secret key, it is referred to in the algorithm as a vector key.

One risk with using keys or vectors is if someone is able to acquire them, then the secret message is in jeopardy. To help with this, a password that is converted (or hashed) into a binary string is combined with the vector key using the binary exclusive-or (XOR) function. An XOR function takes two binary strings that are laid side-by-side with each other and the bit from one string is compared with the corresponding bit from the other string. If the bits match, then the value is false or "0"; if the bits don't match, then the value is true or "1". For example, if the binary strings 1100 and 1010 are put through an XOR function the result is:
1100 (value 1)
1010 (value 2)
0110 (result)

What is useful about the XOR function is that if the function is performed again with the result and one of the previous values, you retrieve the other original binary string.[15] For example:
0110 (result)
1010 (value2)
1100 (value1)

After the algorithm combines the password hash and the vector key, the result is passed through another XOR function with the plaintext binary string. This helps to provide more "cover" for the sensitive information. Now when the string is converted into a chess game, only encrypted information is found instead of the plaintext data.

Using a vector key and a password helps to provide strong authentication. There are several ways to authenticate someone, which is simply proving your identity: by something you have, something you know, and something you are. Strong authentication should provide at least two out of three of these methods.[16] The combination of the vector key (something you have) and the password (something you know) meet these criteria.

2.3 Distribution of the vector key

One issue that inevitably comes up when dealing with encryption is how to distribute the keys to the party you're communicating with. If you go through the trouble of trying to hide a covert message, you don't want to blow it by sending in your message, "By the way, attached is the key…" Sending the password may not be too hard because it is usually fairly short, but sending the vector key would look very conspicuous.

[14] Schneier 3.
[15] Cole et al. 36.
[16] Harris, Shon. CISSP Certification All In One Exam Guide. New York: McGraw-Hill/Osbourne, 2002, 132.

To work around this issue, an investigation was performed to see if there was a way to send the vector key covertly with the encrypted chess game. It's a little like leaving your keys in the ignition of your car, but since you also need the password, it's more like leaving the key in your car with the door locked.

One thing that was noticed about the size of the vector key is that it is 336 bits long, which is the product of 12 * 28. Why is this significant? Well, there are twelve months in the year, and there are at least 28 days in each month. Since one of the headers on the PGN format is the Date field, it is possible to send the vector key using this field, one bit at a time. Each bit in the vector key corresponds to one day on the calendar, minus the 29th, 30th and 31st dates of the month. If the bit in the vector key is equal to "1", then the Date of the chess header would equal to the corresponding calendar date of the vector key.

For example, if the first five bits of the vector key are 10011, then for the first chess game, the Date field would equal January 1st because the first bit is on. On the second chess game, the Date field would equal January 4th. This is because the second and third bits are "0", so by setting the date to January 4th, the program can imply that the corresponding bits for January 2nd and January 3rd are "0". The third chess game would have a Date field of January 5th because the fifth bit is on. This process would continue until all the bits of the vector key are encoded in the Date fields. The program can then scan through the chess games provided, extract the vector key value, and decrypt the chess games using the extracted vector key. Since the vector key is combined with the password, it is still protected as long as the password can be sent discretely.

Once the vector key is sent, it can be reused so that only the encrypted message is sent in the future. Otherwise, you would have to send over 100 games for each message, which is inefficient if your messages aren't very long.

2.4 Using the chess board as a substitution box

The vector key that is used to determine the list of valid chess moves to this point is static throughout the algorithm. Since the vector key is a critical part of the security of the algorithm, it is important to make it as difficult as possible for someone to try and find a pattern and discover its value. To do this, the priorities should be shifted a random amount after each move. That way, patterns that may develop over a chess game will only be limited to a single move instead of the entire game.

The shifting can't be random, though, because otherwise there would be no way to decrypt the data. There must be a method to shift the priorities by various amounts in a non-consistent manner, but predictable so that the process can be duplicated on the decryption side. A good way to accomplish this is by using a substitution box.

A substitution box is used in the popular Data Encryption Standard (DES) algorithm, and is one of the most important steps in the process because it gives the encryption a more non-linear appearance and adds to its security.[17] The substitution box is a table of constants that are periodically added to the data to further confuse and randomize the encrypted data. The substitution box that can be used for the chess steganography algorithm is the chess board itself. The pieces on the chess board are constantly shifting and changing as the game progresses. By assigning numbers to each of the 64 squares, and summing the values of the squares that have pieces on them, this value can be used to shift the priorities of the pieces.

For example, if the lower-left-hand square has a value of 1, and there is a chess piece on this square, then the priorities for the pieces are shifted by one to the right, and the piece with the lowest priority leapfrogs to the front to become the piece with the highest priority.

Pawn → Bishop → Knight → King → Queen → Rook → ~~Pawn~~

Figure 5: After the list of priorities is shifted to the right one space, the pawn priority would be moved to the beginning of the list.

If there is a piece on the 2nd square and the 44th square, then the values for these squares are summed together and the priority list is shifted to the right that many times. To make the process more randomized, a loop is first performed to check the squares to see if a piece is on them. The sum is added to a seed value which varies depending on the type of piece it is, and then summed with the previous total. For example, if there are pieces on the 1st, 2nd and 3rd square and the seed value is 2, then the shift value for the first round is 1 (square's value) + 2 (seed's value) = 3. The shift value for the second round is 3 (previous total) + (2+2) = 7. The shift value for the third round is 7 + (3+2) = 12. This process continues until all the pieces are summed together to create the final shift value. This helps add to the complexity of the algorithm and makes it more difficult to try and crack the secret information.

2.5 Encoding binary into the PGN header

To complete the algorithm, the binary data must also be encoded in the PGN header. The header contains several options for hiding information, but they are encoded using various steganography methods.

[17] Schneier 274-275.

```
[Event "31st Masters"]
[Site "Bergen NOR"]
[Date "2001.11.05"]
[White "Bulmuteng, F."]
[Black "Esyrro, I."]
[Round "7"]
[Result "0-1"]
[WhiteElo "1262"]
[BlackElo "1363"]
[ECO "E16"]

1. d4 Nf6 2. c4 e6 3. Nf3 b6 4. g3 Bb7 5. Bg2 Bb4+ 6. Bd2 Be7 7. Bh3 Ng8 8. b3
Kf8 9. Rf1 d6 10. Ne5 Bg5 11. Bxg5 h6 12. Bc1 Bh1 13. Ng6+ Ke8 14. Bf4 e5 15.
Bg2 exd4 16. Kd2 Qd7 17. Nf8 Na6 18. f3 Nf6 19. Kc1 Ne4 20. Bxh1 Rh7 21. Qd3
Qc6 22. Qc3 Ng5 23. Qd3 g6 24. Ne6 Qd7 25. Qc2 Rb8 26. h4 Qa4 27. Nf8 Rg7 28.
Qf5 Qa3+ 29. Kc2 Ra8 30. Rd1 Nh7 31. Bg5 Nxf8 32. Qd7+ Nxd7 33. e3 Ndb8 34. c5
Rh7 35. Rd3 Kd7 36. b4 Qa5 37. b5 Qd2+ 38. Kxd2 dxc5 39. Ke1 c4 40. Ra3 hxg5
41. exd4 Nb4 42. 0-1
```

Figure 6: Sample output of a chess game with the PGN header.

Substitution-based steganography is a method of hiding information by overwriting data that is already on the file.[18] This method is used to hide binary information in the Site section of the PGN header. A series of 256 different cities are listed as possible locations of the encoded chess game, which is able to hide eight bits of information. For example, if the next eight bits of the binary string to be encoded are 11001001, which equals 201 in decimal, then the program would pick the 201st city for the Site from the list of possible cities. The other PGN header that is encoded using substitution is the Round the game was played in during the fictional tournament.

Generation-based steganography uses the plaintext binary string to generate the hidden information.[19] This is different from the previous method, which is basically a one-for-one substitution of the binary data for the displayed data. This method may take several iterations before the data is completely encoded. For example, the player's ratings (WhiteElo and BlackElo) are created using this method. The first number is always one because some of the games would look suspicious if they claimed to be played by a "master" level player, which have ratings above 2000 points. The next three digits can be any number between 0 and 9. If the next bits to be encoded are 011, which is 3 in decimal, then the program would place the number "3" in the next slot of the player's rating. This process would continue until all four slots are filled in. The generation-based method is used to create the values of the PGN header on the Event, Date, and the players' names and ratings.

[18] Cole, Eric. Hiding in Plain Sight: Steganography and the Art of Covert Communication. Indianapolis, Indiana: Wiley Publishing, Inc., 2003, 112.

[19] Cole 112-113.

Algorithmic-based steganography uses some sort of algorithm to determine where in a file the data is hidden.[20] The Encyclopedia Chess Opening (ECO) code in the header makes use of this method. The ECO value is a code which describes what kind of chess opening is being used. Using standard openings for encoding information helps set the game off to a more normal-looking start, instead of having openings that look random and very unorthodox, which might arouse suspicion. No information is encoded while the opening from the ECO is being created, so you must know the chess opening that the ECO code refers to in order to know where to begin checking for the encoded chess moves. For example, if the ECO code "E76" is being used, then the first seven moves in the chess game refer to the chess opening and do not hold any encoded values. The program must read through to the eighth move before looking for hidden data in the chess game.

To help promote the diffusion of the data across the entire chess game, the process for encoding the chess game and the PGN header flip flop. Binary values are determined and encoded in a chess move, and then binary values are determined and encoded in part of the PGN header. The process then repeats until all the binary values have been encoded.

The PGN header that is being encoded is also not performed in a linear fashion. It is determined by a complex summation of the piece locations, priorities, and which move was last chosen. This way the hidden data is not laid out in a top-to-bottom fashion, but jumps around and is spread out across the entire header. This adds to the complexity and diffusion of the data and in turn adds to the security of the algorithm.

The PGN values also have a seed value added to them, based on the current priorities of the various pieces for both sides. For example, if the Round is being determined, and the next three bits to be encoded are 101, then the value for the Round header is 5. To make cryptanalysis, the act of finding ways to break an encoded message[21], and steganalysis more difficult, the current priorities of certain chess pieces are summed together and added to this value. If the sum of the priorities is 4, then the value for the Round header is now 9. If the summed value is greater than 9, then the value is divided by 10 and the remainder is used as the value for the Round header. Obtaining the remainder after dividing two numbers is commonly known as the modular function. For example, if the sum of the priorities is 6, then the new value for the Round header is now 11. Since this is greater than 9, the value 11 is divided by 10, leaving a remainder of 1 and this value is used for the Round header. This process helps further confuse the data so it can't be easily extracted from the encrypted text from someone trying to deduce what the message is.

[20] Cole 109-110.
[21] Cole 187.

2.6 Encoding binary into chess comments

The PGN standard allows for more than just a documentation of a game, but also an analysis of the game. For some games, a player may go back and review the game for great insights, blunders, and how the game compares with others that have been played. These comments can be written into the chess game and give additional opportunities for hiding information.

There are three ways in the PGN standard to encode comments for a chess game. The first way is using a method called the Numeric Annotation Glyph (NAG)[22]. There are 128 commonly used comments in chess that are assigned integer values from 1 to 128. These codes are then displayed in the chess game as the integer value beginning with a dollar sign (e.g. 12. Nxe5 $14). These values can be randomly inserted into the text using insertion-based steganography. This method inserts additional information into a file, but it doesn't effect the actual representation of the data.[23] In this case, if an NAG is added to a chess game, it does not change the way the game was played. For this algorithm, there is a four percent chance after each move that an NAG value is added.

The second method for adding a comment to a chess game is by using the Recursive Annotation Variation (RAV)[24]. If a series of alternate chess moves are discovered during analysis that a player finds interesting, he can document these moves and enclose them in parenthesis. For example, the following subset of chess moves:

12. Nxe5 Bxe5 …

may have an RAV added to it:

12. Nxe5 (12. dxe5 fxe5 13. Rc1) Bxe5 …

Creating RAV chess moves to encode hidden information makes use of generation-based steganography. In this case, the covert information is the binary string and the overt information is the chess moves. For this algorithm, there is a two percent chance after each move that an RAV is inserted into the chess game.

The third method for adding comments to a game is by simply adding a text comment surrounded by braces (e.g. 12. Nxe5 {This was a good move})[25]. In order to encode hidden information in a chess comment, you have to make use of what is known as grammar-based steganography. This method uses the hidden data to create an output file based on a predefined grammar.[26] By taking a large sample of chess comments that have been used with other games, a comment that encodes a certain number of bits can be created. For example, four comments can be predefined with the following bits:

---

[22] Edwards 8.2.4.
[23] Cole 111.
[24] Edwards 8.2.5.
[25] Edwards 5.
[26] Cole 110.

“This was a good move” = “00”
“This requires further analysis” = “01”
“Questionable, yet effective” = “10”
“Leads to good opening for white” = “11”

To encode “00” in a chess game, the comment, “{This was a good move}” is added. A comment can be further broken down to include synonyms, which also have bit values. For example, synonyms for the first comment can be assigned as the following:

“This was a great move” = “0000”
“This was a questionable move” = “0001”
“This was a good move” = “0010”
“This was a bold move” = “0011”

This helps mix the comments around so they don’t seem so canned, and also helps hide additional data. For this algorithm, there is a four percent chance after every move that a chess comment is added to the chess game.

By using the methods described so far, it is possible to hide on average almost 650 bits of information per chess game. By making use of the white space in the text to hide additional information, it is possible to encode over 90 times more data into each chess game.

2.7 Encoding binary into extra spaces

Spaces that are used between words are only a few pixels wide. Adding an additional space between words may go unnoticed if not closely analyzed. This additional space can be used to encode additional bits of information.[27] If there is only one space, then a “0” is encoded; if there are two spaces, then a “1” is encoded.

```
1. e4  e6  2. d4  d5  3. exd5  exd5  4.  Nc3  Nf6 5. Bg5  Nc6 6.  Bh4 Rb8 7.
Qe2+  Kd7  8.  Nf3  Rg8  9. Qb5  g5 10.  Qc4 Bb4  11. Be2 Bxc3+  12. Nd2 g4
13. b4  Rg7  14.  Qb5 Qg8 15. 0-0-0 Bxd4  16. Ne4 Qd8 17.  Kb1  g3 18. Rdf1
b6  19. f4  dxe4 20.  Rfg1 Ra8 21.  Kc1  Nd5 22.  Bh5  Ke8  23.  Bg4 h6 24.
Kd1  Be3 25.  f5  Kd7  26.  h3 Bc1  27. Qa5  Qg5 28. Qb5  Qxh4  29.  Bh5 Qf6
30.  0-1
```

Figure 7: Sample output of a chess game, with additional spaces between the words being used to hide information.

In addition to the spaces between the words, there are also spaces that can be added before the chess game and after the chess game. In the PGN Standard, a line is added to separate the header from the chess game and another line is added to separate the chess game from the next header. Any extra spaces that are used in these two lines are ignored by the PGN standard[28]. For each line, six

[27] Wayner, Peter. Disappearing Cryptography. Information Hiding: Steganography & Watermarking. 2nd ed. San Francisco, California: Morgan Kauffman Publishers, 2002, 285.
[28] Edwards 8.1.

bits of information can be hidden safely on one line. As a maximum, if all six bits that needed to be hidden were ones (i.e. “111111”), then this value would translate into 63 in decimal. Therefore, to hide these six bits, 63 spaces would be encoded on one line. If any more bits were to be hidden, it would possibly cause two lines to be seen instead of just one, which would arouse suspicion.

Finally, there is plenty of room to add additional spaces after the PGN header to hide information. Only five bits of information can be hidden per line, since there is already some printed text and according to the PGN standard, each line should not be longer than 80 characters[29]. This would result in a maximum of 31 extra spaces after each line in the PGN header.

Using the spaces within a chess game, an average of over 255 bits of additional information can be hidden per game. Because spaces are not visible on the screen or a printed page, they can be used to store more hidden information by using the formatting options available with the word processor.

2.8 Encoding binary into font and paragraph formats

The Microsoft Word application has many formatting options available for the user to modify the way their document looks. These formats apply to individual characters (font formats) or to the paragraphs (paragraph formats).

The font format options can be found by going to the Format menu in Word and selecting Font. On this window, there are many options on how to format the text in your document. If you modify only the white spaces of a document, none of those formats are visible on the screen or on printed paper. This gives an opportunity to hide information in places that are not visible to the user.[30] For example, if a certain space has a "bold" format, then a “1” is encoded; if a certain space does not have a "bold" format, then a “0” is encoded. Since a bolded space looks the same as a regular space, the change can be inserted into the text without changing the appearance of the document.

Another example is by changing the color of the white space. Since there is no text on the white space, there is no visible color on the screen. There are 2^{24} different colors that can be chosen from, which can hide 24 bits of information for each white space in a document. You can also slightly change the actual color of the text to be not quite solid black, but still have the appearance of black on the screen. There are about 2^{18} different color values that look very close to black, but not enough to make a noticeable difference. This allows 18 more bits of information to be hidden for every non-white space character in a document.

The paragraph format options can be found by going to the Format menu in Word and selecting Paragraph. By slightly altering the paragraph formats so that they aren’t visible to the user, you can hide some data depending on what format is chosen.[31] For example, if the amount of space before a paragraph is “1 pt”, then a value of “1” is encoded; if the space is “0 pt” between paragraphs, then a

[29] Edwards 8.2.1.
[30] Wayner 284-285.
[31] Wayner 285.

“0” is encoded. These values are so small, that no one would be able to notice the spaces between the lines are different.

To see how much information can be encoded on average in a chess game, please refer to Appendix C. To review the steganography methods that were used to encode information, refer to Appendix D.

3.0 After snapshot

3.1 Installing the applications

To make the applications functional, create a directory for the programs to reside and copy the zipped files below to this directory.

StegoChess.zip

There are three files that have been zipped. The first one is a Word document which encodes a text file into a chess program and is called, “Chess Steganography Encrypting Program.doc”. The second one is a Word document which decrypts an encoded chess program back into text and is called, “Chess Steganography Decrypting Program.doc”. The third file is used to hold all the constant values for the two programs and is called, “chess_steganography.ini”. As long as these three programs are in the same directory, they should work fine.

Since the programs are using macros, they must be enabled in order to work correctly. To do this from Microsoft Word, go to the Tools menu and select Macros and then Security. Change the security setting to medium. This way, whenever you open a document in Word that has macros, you will be prompted if you want them enabled. When you open the Word documents, be sure to select “Enable Macros” when prompted.

3.2 Encrypting data using Chess Steganography

First, you want to open the “Chess Steganography Encrypting Program.doc” document. When prompted, you should select “Enable Macros” so that you can run the program inside this document.

Once open, you can begin typing in a sample message that you want to encrypt.

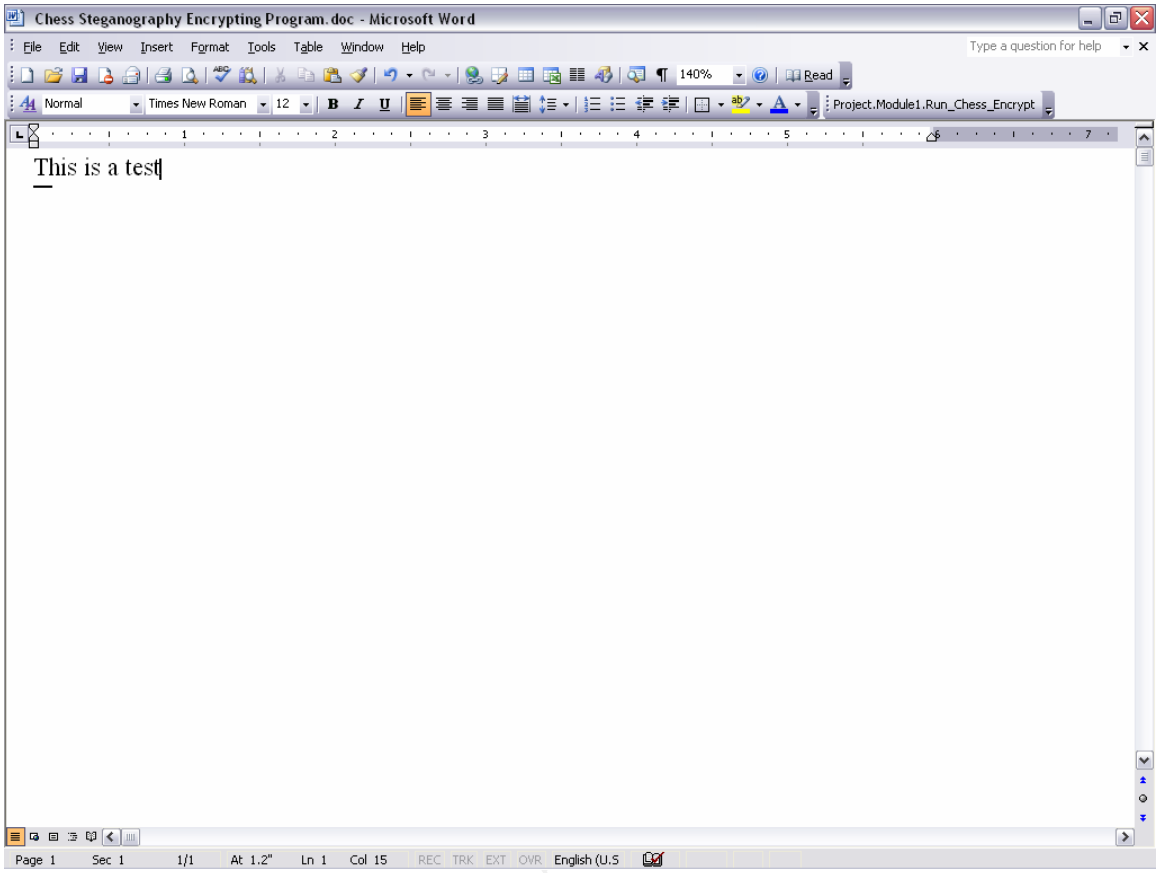

Figure 8: Screen print of the Chess Steganography Encrypting document.

When you are ready to encrypt your text message, you can do one of two things. You can either click the “Project_Module1.Run_Chess_Encrypt” button on the toolbar; or you can go the Tools menu, select Macro, select Macros, scroll down the list macros until you find the Run_Chess_Encrypt macro and click the Run button. Either method will open up the “Chess Steganography Encrypting Program” window.

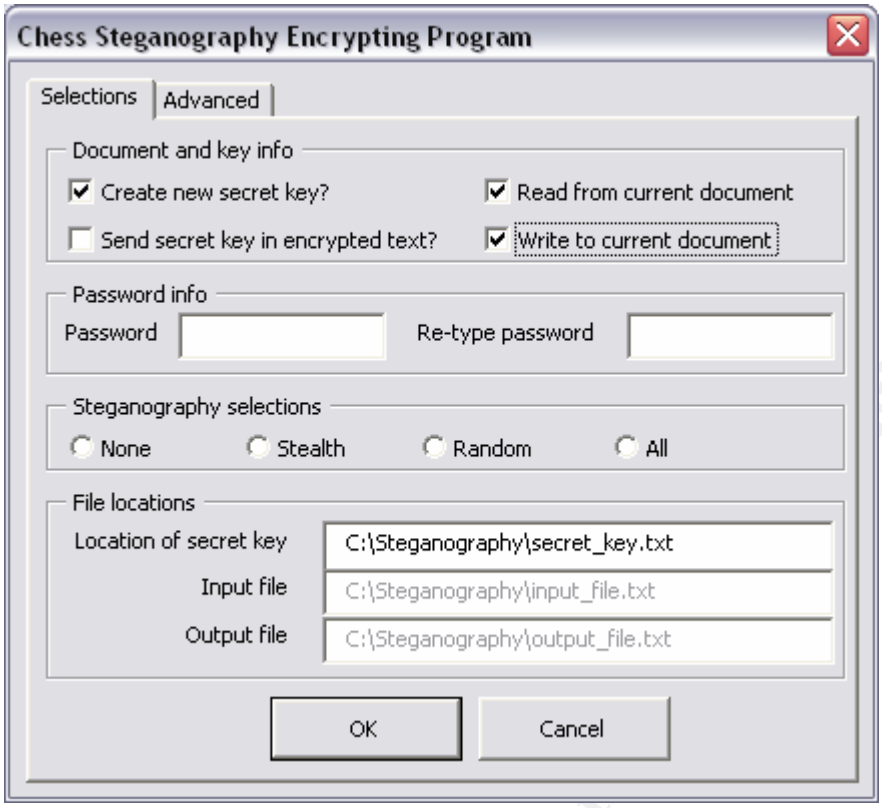

Figure 9: Screen print of the Chess Steganography Encrypting Program window.

On the Selections tab, there are four main sections. The first section is the "Document and key info" section. There are four checkboxes that are available here. The first one is the "Create new secret key?" checkbox. If selected, then a new vector key is created for this process; if it is not selected, then the vector key is read from the location specified in the "Location of secret key" field down below. The second field is the "Send secret key in the encrypted text?" checkbox. If it is selected, then the vector key is sent inside the Dates of all the chess games. The other two fields, "Read from current document" and "Write to current document", determine where the data is written to or from. If selected, the data is written to/read from the current Word document; otherwise, it is written to/read from the files listed below.

The second section is the "Password info" section. This is where you will input a password for the vector key. To ensure it is typed in correctly, there is a second field called the "Re-type password" field where you re-type the password.

The third section is the "Steganography selections" section. This section determines what types of word processing formats you want to include with your chess games. If "None" is selected, then only the chess game and the PGN header are used to encode information. If "Stealth" is selected, then only the formatting options that are not easily visible to the casual observer are used. The "Random" radio button picks random formatting options to encode the data. The "All" radio button makes use of all the available formatting options to encode

the data. To see the detailed selections of what formatting options are used, select the “Advanced” tab.

Figure 10: Screen print of the Advanced tab with the Stealth options selected.

The fourth section is the “File locations” section. This determines what directory and file names are being used for the secret key, input file (if being used) and the output file (if being used).

Once you have the desired settings selected, click the “OK” button and the program begins. The result is a chess game document in the PGN standard format.

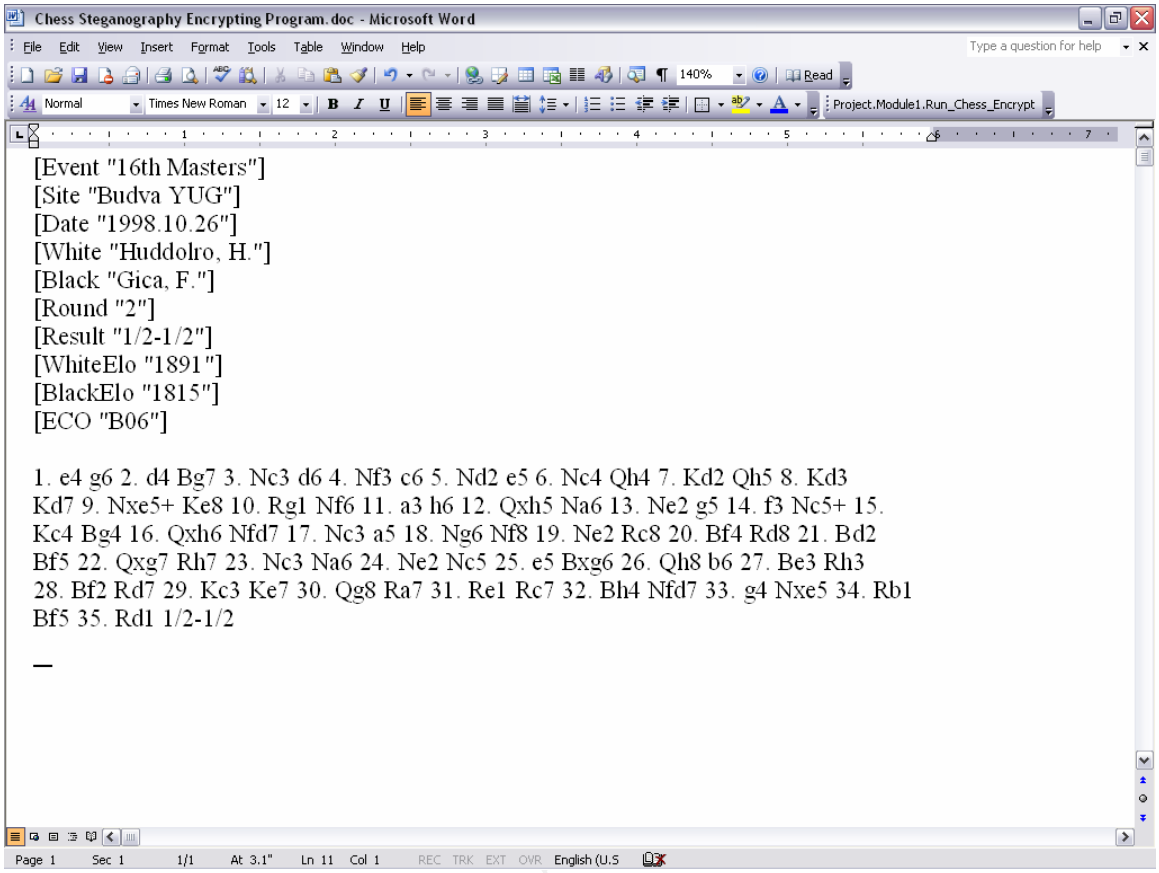

Figure 11: Screen print of the results after running the chess steganography program.

3.3 Decrypting data using Chess Steganography

Now that you have an encrypted chess game, the next step is to decrypt it. In order to do this, you must open the document that contains the decrypting program. This program is found in the "Chess Steganography Decrypting Program.doc" document. When prompted about the macros found in the document, select the "Enable Macros" option so that you can run the program.

The encrypted data may have many formatting options that would be lost if everything is not copied correctly to the decrypting document. To ensure this doesn't occur, go to the encrypting document and select Edit from the tools menu and select the "Select All" option. This option ensures all margin, paragraph, and font options are selected to be copied. Now select the "Copy" option from the Edit menu and go to the decrypting document. To ensure there are no left over formats or text in the document, use the "Select All" option and hit the Delete button. Now select the "Paste" option from the Edit menu. You are now ready to run the decrypting program.

This can be done in one of two ways. You can either click the "Project.Module1.Run_Chess_Decrypt" button on the toolbar; or click on the Tools menu, select Macro, select Macros, navigate down the list of macros and

select “Run_Chess_Decrypt” and click the Run button. Either method will open the “Chess Steganography Decrypting Program” window.

Figure 12: Screen print of the Chess Steganography Decrypting Program window.

There are several options to choose from when running the decrypting program. First, you must let the program know if you will be decrypting the current Word document, or if you will be reading from an input file. If the current Word document will be decrypted, then check the “Read from current document” checkbox. Otherwise, deselect it and make sure the Input file field is filled in correctly.

Second, you must let the program know if you will be writing the results to the current document or sending it to an output file. If the results will be written to the Word document, then check the “Write to current document” checkbox. Otherwise, deselect it and ensure the Output file field has the correct directory and file name where the results will go to.

Third, you must enter the password for the vector key file. It should be the same value that was used to encrypt the data.

Finally, you must enter the location of the secret vector key. This is used to determine what the priorities are for the chess pieces and their moves.

When ready, click the “OK” button and the program begins. It starts with scanning the document for any formatting to see if there is any hidden information in them. Depending on the length of the document, this may take a few moments. Once completed, the original text is displayed, or is placed in the output file if that is what was selected.

3.4 Chess steganography conclusions

After creating a steganography algorithm using a chess game to hide the information, several conclusions can be found. First, it is possible to use several different steganography methods inside one algorithm. This helps create a more robust solution than other algorithms that are more one-dimensional when trying to hide information.

Second, using word processing formats is a very effective way to hide information. For this algorithm, over 90% of the bits that are hidden were accomplished by using this technique. This same technique can be used not just with chess games, but with any text document.

Third, this algorithm still needs outside intervention when sharing the password. This process can be tricky, since it can lead to a suspicion of sending secret information, which can defeat the whole purpose of using steganography in the first place.

Fourth, the algorithm is rather slow. Because of the intensive process of determining chess moves and reading in the formatting information, it causes the program to run slowly, especially if the input is large. To help compensate for this, smaller documents and text files should be used.

Fifth, in order to make sending encoded chess games believable, a context must be developed. This can be done by having people who regularly talk about chess topics send the secret information, or establishing a context by regularly sending chess games through e-mails.

Sixth, there are several ways to counteract the hiding of data using this algorithm. You can change the order of the games, remove formatting of characters and paragraphs, or remove certain headers from the games. Slightly altering a file that is potentially hiding information can essentially destroy the information, no matter what method of steganography is used.

Lastly, close observation of the chess game may cause suspicion that it is not authentic. Since the moves are not based on strategy or tactics, a person familiar with the game may question certain moves that make no common sense. Using standard openings helps reduce this risk, but the rest of the game will still have this issue.

Using this method of steganography has some benefits and some drawbacks. Depending on the need of the individual and the context of the situation, it can provide an effective method for transmitting sensitive information.

Appendix A: How much data can be hidden using correspondence chess?

Correspondence chess is a version of chess where two people make each move by sending a postcard to their opponent. The postcard usually contains the following information which are options for hiding information:

- Opponent's last move number. You can probably have a chess game with as many as 60+ moves without looking suspicious. This would translate to about $2^6 = 64$ bits that can be hidden here.
- Opponent's last move. To ensure you recorded your opponent's move correctly, you typically echo their last move. There are 1,560 possible moves that can hold 2^{10} bits of information. (See below)
- Player's last move number. Since the next move must be sequential, there can be no additional information hidden here.
- Player's last move. To ensure your opponent recorded the last move correctly, you typically echo your last move. There are 1,560 possible moves that can hold 2^{10} bits of information. (See below)
- Player's move. This tells your opponent what your current move is for this game. You can hold 2^{10} bits of information as show above.
- Conditional move (optional). In correspondence chess, if you expect your opponent to make a certain move, you can send a conditional move. A sample one would look like: If Kg3, Then Qf1. Since you are hiding two possible chess moves, there are $2^{10} * 2$ moves = 2^{20} bits of information that can be hidden.
- Comments. Depending on how creative you want to be, a comment can stand for a single bit of information. For argument's sake, about $64 = 2^6$ different comments can be realistically encoded.
- Opponent/Player salutation. You can either put "Mr." or "Ms." (whichever is appropriate) or leave this blank. This can encode only one bit each.
- Opponent/Player first name. You can either put your entire first name or just your first initial. This can encode only one bit each.
- Opponent/Player last name. You probably can't modify this, or your opponent wouldn't receive your postcard.
- Opponent/Player address. You probably can't modify this, or your opponent wouldn't receive your postcard.
- Stamp orientation. You can place the stamp in various positions to hide additional bits (upside down, sideways, and diagonally). This can encode 8 bits = 2^3 of information.
- Upper/lower case. You can hide an additional bit of information depending on whether you write the postcard in upper or lower case.

To see how many possible bits can be hidden in a single chess move, you need to first determine how many possible chess moves there are.

1. Pawns. Move forward (8 pieces x 7 moves) + Capture left (7 pieces x 6 squares) + Capture right (7 pieces x 6 squares) = 140 possible moves.
2. Rooks, Knights, Bishops, Queen, King. Can move to all squares (5 pieces x 64 possible squares) and capture to all squares (5 pieces x 64 possible squares) = 640 possible moves.
3. Moves that cause the king to be in check. Technically, any move can cause the king to be in check, which is identified by a '+' after your move (e.g. Qc4+). 640 + 140 = 780 possible moves.

So the total number of possible chess moves available is 780 + 640 + 140 = 1560, which is about 2^{10}.

When playing correspondence chess, you play two games with your opponent: one where you are the "white" color and one where you are the "black" color. The total number of bits of information that can be hidden are:
(Game 1) * (Game 2) * (Comments) * (Player's info) * (Opponent's info) * (Stamp orientation) * (Upper/lower case) =
(Last move number * Opponent's last move * Player's last move * Player's move * Conditional move) * (Game 2) * (Comments) * (Player's salutation * Player's first name) * (Opponent's salutation * Opponent's first name) * (Stamp orientation) * (Upper/lower case) =
$(2^6 * 2^{10} * 2^{10} * 2^{10} * 2^{20}) * (2^6 * 2^{10} * 2^{10} * 2^{10} * 2^{20}) * (2^6) * (2^1 * 2^1) * (2^1 * 2^1) * (2^3) * (2^1) =$
$(2^{56}) * (2^{56}) * (2^6) * (2^2)$ x $(2^2) * (2^3) * (2^1) =$
2^{126}

This means that one correspondence chess postcard can hold up to 126 bits of information. If an ASCII character is converted into seven bits, then you can only hide about 18 characters of information, which isn't very much. To make this effective, you need to create a substitution chart for each item on the postcard you want to hide data on. For example, "axb5" would be "blue", or "Nc7+" would be "night". This way, you are able to hide much more information, but it is difficult to translate and the substitution table would be very large.

Appendix B: Vector key layout for the chess algorithm

This is the file layout for the vector key, which determines what priorities the pieces will be checked for valid moves.

Chess piece	Length	Start Position	End Position
White Queen's Rook	4	1	4
White Queen's Rook move North	2	5	6
White Queen's Rook move East	2	7	8
White Queen's Rook move South	2	9	10
White Queen's Knight	4	11	14
White Queen's Knight move North and West	3	15	17
White Queen's Knight move North and East	3	18	20
White Queen's Knight move East and North	3	21	23
White Queen's Knight move East and South	3	24	26
White Queen's Knight move South and East	3	27	29
White Queen's Knight move South and West	3	30	32
White Queen's Knight move West and South	3	33	35
White Queen's Bishop	4	36	39
White Queen's Bishop move North East	2	40	41
White Queen's Bishop move South East	2	42	43
White Queen's Bishop move South West	2	44	45
White Queen's	4	46	49
White Queen's move North	3	50	52
White Queen's move North East	3	53	55
White Queen's move East	3	56	58
White Queen's move South East	3	59	61
White Queen's move South	3	62	64
White Queen's move South West	3	65	67
White Queen's move West	3	68	70
White King's move North	3	71	73
White King's move North East	3	74	76
White King's move East	3	77	79
White King's move South East	3	80	82
White King's move South	3	83	85
White King's move South West	3	86	88
White King's move West	3	89	91
White King's Bishop	4	92	95
White King's Bishop move North East	2	96	97
White King's Bishop move South East	2	98	99
White King's Bishop move South West	2	100	101
White King's Knight	4	102	105

White King's Knight move North and West	3	106	108
White King's Knight move North and East	3	109	111
White King's Knight move East and North	3	112	114
White King's Knight move East and South	3	115	117
White King's Knight move South and East	3	118	120
White King's Knight move South and West	3	121	123
White King's Knight move East and South	3	124	126
White King's Rook	4	127	130
White King's Rook move North	2	131	132
White King's Rook move East	2	133	134
White King's Rook move South	2	135	136
White Queen's Rook Pawn	4	137	140
White Queen's Knight Pawn	4	141	144
White Queen's Bishop Pawn	4	145	148
White Queen's Pawn	4	149	152
White King's Pawn	4	153	156
White King's Bishop Pawn	4	157	160
White King's Knight Pawn	4	161	164
White King's Rook Pawn	4	165	168
Black Queen's Rook	4	169	172
Black Queen's Rook move North	2	173	174
Black Queen's Rook move East	2	175	176
Black Queen's Rook move South	2	177	178
Black Queen's Knight	4	179	182
Black Queen's Knight move North and West	3	183	185
Black Queen's Knight move North and East	3	186	188
Black Queen's Knight move East and North	3	189	191
Black Queen's Knight move East and South	3	192	194
Black Queen's Knight move South and East	3	195	197
Black Queen's Knight move South and West	3	198	200
Black Queen's Knight move East and South	3	201	203
Black Queen's Bishop	4	204	207
Black Queen's Bishop move North East	2	208	209
Black Queen's Bishop move South East	2	210	211
Black Queen's Bishop move South West	2	212	213
Black Queen's	4	214	217
Black Queen's move North	3	218	220
Black Queen's move North East	3	221	223
Black Queen's move East	3	224	226
Black Queen's move South East	3	227	229
Black Queen's move South	3	230	232
Black Queen's move South West	3	233	235
Black Queen's move West	3	236	238
Black King's move North	3	239	241

Black King's move North East	3	242	244
Black King's move East	3	245	247
Black King's move South East	3	248	250
Black King's move South	3	251	253
Black King's move South West	3	254	256
Black King's move West	3	257	259
Black King's Bishop	4	260	263
Black King's Bishop move North East	2	264	265
Black King's Bishop move South East	2	266	267
Black King's Bishop move South West	2	268	269
Black King's Knight	4	270	273
Black King's Knight move North and West	3	274	276
Black King's Knight move North and East	3	277	279
Black King's Knight move East and North	3	280	282
Black King's Knight move East and South	3	283	285
Black King's Knight move South and East	3	286	288
Black King's Knight move South and West	3	289	291
Black King's Knight move East and South	3	292	294
Black King's Rook	4	295	298
Black King's Rook move North	2	299	300
Black King's Rook move East	2	301	302
Black King's Rook move South	2	303	304
Black Queen's Rook Pawn	4	305	308
Black Queen's Knight Pawn	4	309	312
Black Queen's Bishop Pawn	4	313	316
Black Queen's Pawn	4	317	320
Black King's Pawn	4	321	324
Black King's Bishop Pawn	4	325	328
Black King's Knight Pawn	4	329	332
Black King's Rook Pawn	4	333	336

Appendix C: Amount of data hidden in the chess game

There are three main categories in the chess game that can hide information: the chess game, the font formats, and the paragraph formats. Each of these can hold various amounts of hidden data. 100 rounds of this algorithm were run to determine the average amount of data that can be hidden per chess game for each category.

Chess steganography category	Number of bits that can be hidden	Average bits hidden per game
Chess game		360
Length of game	5 bits	
Last move's color	1 bit	
Chess move	0-7 bits	
PGN header		124
Event	7 or 12 bits	
Site	8 bits	
Date	0 or 12 bits	
Length of player's name	6 bits	
Player's name	34-142 bits	
Round	3 bits	
Result	1-2 bits	
Player's rating	18 bits	
Opening used (ECO)	10-11 bits	
NAG	7 bits	26
RAV		77
Length of game	0 bits	
Last move's color	0 bits	
Chess move	0-7 bits	
Comments		56
Opening comment	13-37 bits	
Messages	5 bits	
Synonyms	2-6 bits	
Additional spaces		255
After PGN header	50 bits	
Between header/game	12 bits	
Between words in header	14-17 bits	
Between words in game	3 bits per move	
Between words in comments	2-10 bits per comment	
Font formats		
All caps	1 bit per white space	532
Bold	1 bit per white space	532

Text color	18 bits non-white space	12,779
Emboss/Engrave/Shadow	2 bits per white space	1,065
Font	1 bit per character	1,350
Italics	1 bit per white space	533
Kerning	1 bit per white space	4,260
Outline	1 bit per white space	532
Position	10 bits per white space	4,574
Scaling	2 bits per white space	1,065
Size	2 bits per white space	1,065
Spacing	2 bits per white space	1,065
Superscript	1 bit per white space	533
Underline color	24 bits per white space	12,779
White-space color	24 bits per white space	14,712
Paragraph formats		
Spacing before	1 bit per line	25
Spacing after	1 bit per line	25
Hyphenation	1 bit per line	25
No line number	1 bit per line	25
Keep with next	1 bit per line	25
Widow control	1 bit per line	25
Keep together	1 bit per line	25
TOTAL		58,449

If each character takes seven bits to be encoded, then a chess game can hide about 8,350 characters. This works about to be about 2 ½ pages of single-spaced text.

Appendix D: Types of steganography used

Type used	Part of chess game it is used in
Insertion	NAG, additional spaces
Algorithmic	ECO code, length of player's names, length of chess game
Grammar	Chess comments
Substitution	Site, Round, Result, font formats, paragraph formats
Generation	Event, Date, Player's names, Player's ratings, RAV, chess moves

Definitions:

Insertion-based steganography - inserts additional information into a file, but it doesn't effect the actual representation of the data.

Algorithmic-based steganography - uses some sort of algorithm to determine where in a file the data is hidden.

Grammar-based steganography - uses the hidden data to create an output file based on the predefined grammar.

Substitution-based steganography - hides data by overwriting data that is already on the file.

Generation-based steganography - uses covert information to create the overt information.

References

"General Office Security". MSDN Library. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdsdk/html/sdconSecurityGeneralOfficeSecurity.asp (12 July 2004).

United States. Department of the Army. US ARMY Field Manual 21-75. 1984.

Campbell, J. Franklin. "USCF Abandons Prison Inmates," The Campbell Report. 23 Dec. 2002. URL: http://www.correspondencechess.com/campbell/articles/a021223.htm (12 July 2004).

Cole, Eric. Hiding in Plain Sight: Steganography and the Art of Covert Communication. Indianapolis, Indiana: Wiley Publishing, Inc., 2003.

Cole, Fossen, et al. Track 1.4 - Secure Communications. Vers. 2.2. Track 1 – SANS Security Essentials and the CISSP 10 Domains. 2004.

Edwards, Steven J. "PGN Specification and Implementation Guide." 12 Mar. 1994. URL: http://pgn.freeservers.com/Standard.txt (12 July 2004).

Harris, Shon. CISSP Certification All In One Exam Guide. New York: McGraw-Hill/Osbourne, 2002.

Kahn, David. The Codebreakers. 2nd ed. New York: Scribner, 1996.

Schneier, Bruce. Applied Cryptography. 2nd ed. New York: John Wiley & Sons, Inc., 1996.

Wayner, Peter. Disappearing Cryptography. Information Hiding: Steganography & Watermarking. 2nd ed. San Francisco, California: Morgan Kauffman Publishers, 2002.

Weeks, Mark. "Portable Game Notation (PGN)," About Chess. 12 Oct. 2002. URL: http://chess.about.com/library/weekly/aa101202a.htm (12 July 2004).

Last Updated: July 23rd, 2017

Upcoming Training

SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS
Mentor Session - SEC401	Arlington, VA	Oct 04, 2017 - Nov 15, 2017	Mentor