CipherChess

By

Adam Glaser

Emilio Lahr-Vivaz

Errol McEachron

Presented to

Professor Dave Evans

December 5, 2001

On my honor as a student I have neither given nor received unauthorized aid on this assignment.

_____

_____

_____

Table of Contents

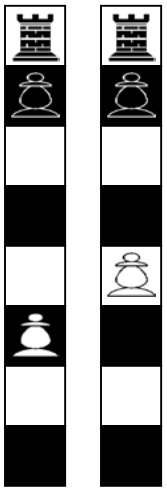
Glossary of Terms

Board Position – The positions of every chess piece currently on the chessboard; a snapshot of the board.

Ciphertext String – The six-bit sequence of binary digits that results from XORing a six-bit plaintext string with a six-bit letter key.

Column Parity – A zero or a one, defined by the following three cases: The number of empty spaces between pieces in one column of a chessboard, modulo two. If there is only one piece in a column, the parity is the number of spaces between that piece and its color's side of the board, including the final square, modulo two. If there are no pieces in a column, the parity is zero.

Examples: In the first example (a), there are three spaces between pieces, thus the parity is one. In the second example (b), there are only two spaces between pieces, so the parity is zero.



(a) (b)

Column Parity String– A six-bit sequence of binary digits obtained from a chessboard by reading the column parity of each of the middle six columns from left-to-right.

Letter Key – The six-bit sequence of binary digits used to encrypt a single plaintext letter; may be a chunk from a larger key.

Plaintext String – The six-bit sequence of binary digits used to represent a letter or number.

XOR – A binary operation performed in a bit-wise manner. The function is summarized in the following table:

XOR Operation		
Inputs		Output
0	0	0
0	1	1
1	0	1
1	1	0

1.0 Motivation

Deep in the jungles of Honduras, secret agent Givus Anay is in dire straights. While investigating the dangerous Kilos Forall drug cartel, Givus has just been captured and brought to a prison deep inside the cartel's secret jungle work camp. Right outside the camp, Givus clumsily dropped his standard issue secret agent distress beacon before he could activate it. If only he could have activated it, the camp would be swarming with agents in a matter of minutes. If only Z hadn't made the beacon look like a little tube of chapstick, the Kilos Forall guard wouldn't have tried to grab it out of Givus' hands (because the Honduran heat is known for causing dry, chapped lips) forcing him to drop it.

As it turns out, all the single cells were taken at the prison, so Givus was put into one of the new doubles. These new rooms were part of an experiment to determine if happy prisoners worked harder, so they were stocked with common games such as Clue, Electronic Battleship, and chess. Unfortunately, the new rooms were also equipped with state-of-the-art monitoring technology such as cameras and microphones.

Givus Anay and Albee Yurbstfrend, his cellmate, were informed that if there was any talk of escape, they would both be shot. How unfortunate! As luck would have it, Albee was on the landscaping detail, and for the next few days she'd be working not 20 feet from where Givus dropped the distress beacon! If only there was some way to tell Albee about the beacon without tipping off the guards.

"Want to play a game?" asked Albee.

How could Givus think of games at a time like this? He was busy contemplating the rest of his life as a prisoner all because of Z's stupid idea. "Nah, don't feel like it." replied Givus.

"Do you like....chess?" asked Albee, with a detectable hint of hope in her voice.

Givus' heart jumped. Those words...could they mean what he thought they did? Givus' secret agent training kicked in as he recited the exact wording of the answer to this question that he had memorized during secret agent challenge-response identification class. "Searching for Bobby Fisher was the worst movie ever," said Givus calmly. He stared intently at Albee...waiting, hoping, and wondering.

Albee's chapped lips curled slightly as she began to smile a subdued smile. "Siskel and Ebert gave it two thumbs up."

Givus silently rejoiced. They were saved. They were all saved. By this time tomorrow, Albee would know exactly how to find and activate Givus' distress beacon and the guards wouldn't know what had happened until it was all over. "Thanks, CipherChess" thought Givus as a single tear rolled down his cheek. They were going home.

2.0 Introduction

Under normal circumstances, Givus, Albee, and the rest of the prisoners would have been up the proverbial "creek" without the requisite "paddle." However, as the story alludes to, CipherChess was their ticket to freedom. What is CipherChess? How did Givus and Albee use it to communicate their intentions undetected? This report will answer these questions and many others as it details the newest cryptographic breakthrough that has the potential to save lives!

CipherChess is the brainchild of Errol McEachron, Emilio Lahr-Vivaz, and Adam Glaser. It is the latest addition to the small family of so-called "low-tech ciphers." In a nutshell, a low-tech cipher is a cryptographic process (not necessarily mechanical) that relies on readily available materials for the encoding and decoding of secret messages. The need for such a cipher exists because there are many scenarios in which utilizing a high-tech cipher (such as

any that necessitate a computer) is impractical. In these situations, a cipher that takes advantage of certain properties of relatively common items and provides a high degree of security is called for. We first learned about low-tech ciphers through Solitaire, created by Bruce Schneier. Solitaire is a low-tech cipher based on a deck of cards. The encryption method of Solitaire has the advantage of being secure and innocuous, in that there is nothing suspicious about a deck of cards. However, the encrypted "messages" of Solitaire are nonsensical strings of letters, which means that transmission of these messages involves writing down and physically transferring the strings to the receiver. Our goal was to improve on this idea by making the mechanism for our encryption double as the medium for message transmission. We wanted to maintain the security of a regular cipher, while at the same time making it possible to hide the very existence of a message.

The science of hidden messages is known as steganography. Steganography, unlike cryptography, is not concerned with encrypting messages. Instead, it focuses on hiding messages in unlikely places. In a sense, this is even more secure than most ciphers, since there is an infinite number of ways to hide messages when there is no suspicion that messages are being hidden!

As the name CipherChess implies, our cipher uses an ordinary chessboard and pieces to encode, transmit, and decode secret messages with a high degree of security. We chose the game of chess for two reasons. First, chess already has a great amount of variety built into it, from the different types of pieces and the way they move to the number and colors of squares on the board. All these variations provide the potential for complexity, which is important for any cipher. Second, chess is a common game. There is nothing suspicious in two people playing chess, or even one person playing by him or herself. The use of an innocent game

for encoding and hiding messages is the steganographic core of CipherChess. Combining principles of steganography, low-tech ciphers, and shared-key encryption, CipherChess delivers security and secrecy all at once.

3.0 Related Research

The design and analysis of a low-tech cipher based on chess requires a fundamental knowledge of cryptography and a basic understanding of the game. The University of Virginia's Cryptology course provided most of the cryptographic principles necessary for the creation of a low-tech cipher. However, some additional resources were used to assist with the encryption and decryption algorithms, as well as the cryptanalysis of the chess cipher. Bruce Schneier's book, *Applied Cryptography,* provides a good discussion of the necessary protocols for a secure encryption algorithm in a systematic manner. More specifically, the chapters regarding protocol building blocks and keys served as an informal guide for the development of the low-tech cipher. Schneier also provides a good basis for analyzing the time and cost requirements for a brute force attack, which will naturally assist in the selection of certain properties for the cipher. Another discussion that was beneficial is Schneier's description of a Randomized Stream Cipher. CipherChess is inherently a stream cipher because it encodes single message bits (in our case, letters) at a time. The combination of a chessboard with 64 black and white spaces, as well as 32 heterogeneous pieces, provides many different possibilities for creating a binary string-based key. According to Schneier, the objective of a stream cipher is to increase the number of bits the cryptanalyst hast to work with, while keeping the secret key small [Schneier, 367]. All of these factors played an important role in the development of our algorithm. Another commonly desired property for cipher systems is confusion and diffusion. Henry Beker and Fred Piper provide a good

description of this property in their book *Cipher Systems*. This provided a good basis to assess our algorithm's diffusion and confusion that helps "reduce the effectiveness of statistical attacks on cryptogram [Beker, 161]." As for the analysis of the chess cipher, the book *Cryptanalysis* by Helen Gains served as a good reference for comparison with similar ciphers. Studying the strengths and weaknesses of similar ciphers allowed us to avoid the common pitfalls associated with such ciphers.

In order to translate the game of chess into an encryption algorithm, it was helpful to consider the application of some other low-tech ciphers. Solitaire, a low-tech cipher devised by Schneier, was the inspiration for our chess-based cipher and motivated its design. Before we could begin to create our cipher, we needed to learn as much as possible about the game of chess. The U.S. Chess Online website was an excellent source for introductory information. It provided ample information on how to "read and write chess" as well as the official rules for playing the game.

4.0 CipherChess Algorithm

CipherChess is a symmetric low-tech cipher. It operates on a 96-bit shared key sequence, which is divided into 16 six-bit *letter keys* (see Glossary). To encode one plaintext letter, the first letter key is XORed with the six-bit binary representation of the plaintext character (henceforth, plaintext string), and the result is the corresponding six-bit binary representation of the ciphertext character (henceforth, ciphertext string). The ciphertext string is transmitted by the message sender moving the chess pieces, using only legal chess moves, to form the *column parity* string (see Glossary) that matches the ciphertext string.

The resulting board position has column parity that represents the ciphertext character and may now be read by the receiver. The receiver decrypts the ciphertext string by XORing the column parity string with the letter key. The result of this operation is the original plaintext string, which can then be converted back to the plaintext letter according to Table 1. To encode another plaintext letter, this process is repeated using the next letter key. If a message of more than 16 characters is required, the sender and receiver may generate a new 96-bit key sequence (see Section 6.0).

Binary Encoding of Alphanumeric Characters					
Lowercase Alphabet				Decimal Digits	
a	000000	n	001101	0	011010
b	000001	o	001110	1	011011
c	000010	p	001111	2	011100
d	000011	q	010000	3	011101
e	000100	r	010001	4	011110
f	000101	s	010010	5	011111
g	000110	t	010011	6	100000
h	000111	u	010100	7	100001
j	001000	v	010101	8	100010
i	001001	w	010110	9	100011
k	001010	x	010111		
l	001011	y	011000		
m	001100	z	011001		

Table 1 – Conversion table for alphanumeric characters to six-bit binary string

Although column parity provides us with up to eight bits of information, our encryption and decryption algorithm is based on only six of those eight bits. We chose to ignore the first and last columns of the chessboard, caring only about the parity of the six middle columns. This decision is important for two reasons: First, six bits of information is more than adequate to represent all 26 letters of the English alphabet and all 10 decimal digits. More importantly, six bits of information uses a 96-bit key sequence more efficiently. Dividing a 96-bit key sequence into letter keys will allow us to generate 16 ciphertext characters, as opposed to only 12, which an eight-bit letter key would create.

4.1 Encryption Algorithm

The encryption algorithm describes the steps necessary to encrypt a single plaintext character. The encryption algorithm proceeds as follows:

1. The plaintext character is converted into a plaintext string using the mapping presented in Table 1.
2. The 96-bit key sequence is divided into 16 six-bit letter keys.
3. The next unused letter key is used for this round of encryption.
4. The letter key is XORed with the plaintext string to produce the corresponding ciphertext string. The used letter key is now discarded.

4.2 Transmission Algorithm

Transmission of a ciphertext string is accomplished by permuting the chess pieces such that the column parity string of the resulting board position matches the ciphertext string.

The transmission algorithm proceeds as follows:

1. Determine the chess parity of the current board position.
2. Determine which column parities must change in order to match the ciphertext string.
3. Change the necessary column parities by moving a chess piece in, into, or out of that column.
4. If the column parity string of the new board position matches the ciphertext string, move to step 5. Otherwise, if the column parity string of the new board position does not match the ciphertext string repeat step 3, moving a piece opposite in color to the last piece moved.
5. The ciphertext string is now represented by the current board position. Set a flag to signal the receiver that the message is ready to be received.

4.3 Decryption Algorithm

Decrypting a transmitted message requires the receiver to simply read the board position once it contains the message to be received. The decryption algorithm is as follows:

1. The receiver recognized the flag which signals that a message is ready to be read.
2. The receiver quickly computes the column parity string of the board position and commits it to memory.
3. The column parity string, which is also the ciphertext string. is XORed with the letter key. This results in the original plaintext string.
4. The plaintext string is then converted to its associated plaintext character according to Table 1.

4.4 Concealing Message Transmission

To maintain the steganographic properties of CipherChess, it is important that the encryption, transmission, and decryption algorithm be undertaken in a method that does not arouse suspicion. Obviously, the circumstances in which CipherChess is being used will dictate, to a large extent, the way that communication between sender and receiver can take place. Returning to the story of Givus and Albee from the introduction, we will now take a look at how they used CipherChess to orchestrate their dramatic liberation from the Honduran work camp.

Assume that Givus and Albee have a shared key that they will use for encryption and decryption. In Section 6.0, a procedure for selecting a key using a chessboard will be given, but for explanation purposes the origin of the key does not matter. Also, the assumption must be made that Givus and Albee have agreed upon some sort of flag to indicate when messages are ready to be read.

To assuage any distrust by the guards, Givus and Albee sit down together and beginning playing chess. They play numerous games together, before Albee finally announces that she is tired of playing and goes to his bed to watch T.V. (clearly, these are luxury cells!). All of the games Givus and Albee played up until this point were meaningless; the real game is about to begin. Givus arranges the chessboard in the initial configuration for a game and begins to play.

Givus moves a white piece, rotates the board, studies it, and then moves a black piece. It becomes clear to the guards (observing through the security cameras placed in the room) that Givus is just pretending to be both himself and Albee. Givus moves each piece in accordance with the chess rules for the piece, consistently alternates black and white moves,

and captures pieces at will. The guards can only conclude that this is perfectly harmless behavior. However, this could not be further from the truth. Givus is not merely moving pieces at random, as it may appear. Instead, Givus is actually using the CipherChess algorithm to encode the letter "c." He intends to sent Albee the message, "chapstick beacon near front gate." After he finishes representing the ciphertext of the letter "c" on the chessboard according to the transmission algorithm, Givus folds his hands together, leans closer to the board and utters an audible "hmmmm." Hearing this and recognizing this gesture as the flag, Albee nonchalantly glances toward the chessboard. She quickly computes the parity of the middle six columns, XOR's the column parity string with the first six bits of the shared secret key, and memorizes the plaintext letter. Approximately 30 seconds later, more than enough time for Albee to receive the ciphertext letter, Givus proceeds to encode and represent another plaintext letter on the board, and the communication continues.

5.0 Example Encoding

We will now present an example of how encryption and transmission take place. This particular example illustrates how one can encode the message "cs" using the letter key "111111." The message "cs" is represented in binary plaintext by "000011 010011" (See Table 1). The ciphertext representation of "cs" is obtained by XORing the key with the plaintext string. The ciphertext message is then represented on the chessboard as "111100 101100" using our column parity scheme.

To represent the ciphertext letter "c," a series of moves is required to change the column parity string from its initial "000000" to the desired "111100." Any sequence of moves on the chessboard that changes the column parity string to the ciphertext string is acceptable.

Since the encryption algorithm is only concerned with the middle six columns of the chessboard, we use the term “first column” to denote the first of the six middle columns from the leftmost side of the board.

The initial setting of the board makes every column parity zero. First, white moves a pawn forward one space (Figure 1, b). Note that this will not change the parity of the first column, since there are still four spaces between pieces in that column. Next, it is black’s turn, and black advances its pawn two spaces (Figure 1, c). Once again, this will not change any column parity. Now, white moves out its knight (Figure 1, d). This has the effect of changing the parity in both the first and second columns. Both columns now only have three empty spaces between pieces, so the parities of both the first and second columns are one.

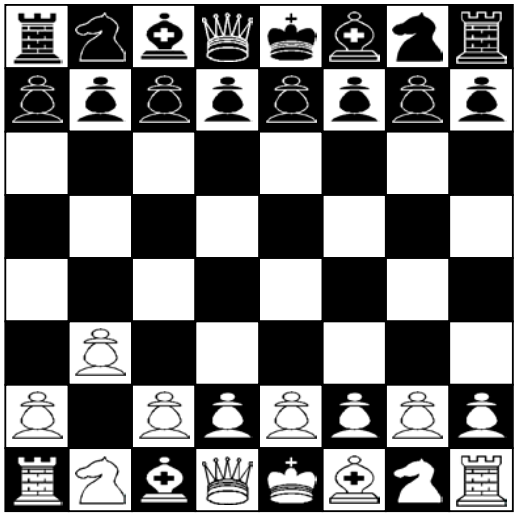
Continuing on, black moves its bishop to the right and up one square (Figure 2, a). This move changes the parity of the third column to one, since there are now only three empty spaces between pieces in that column. Note that the second column’s parity does not change, since the empty space created is not between two pieces. White now moves a pawn forward two spaces (Figure 2, b). This does not change any column parities. Black then moves a pawn in the outermost column (Figure 2, c). Since this column is not being considered, this move is unimportant in and of itself. However, it gets the turn back to white. White can now move a bishop up one space (Figure 2, d). This changes the parity of the fourth column to 1.

At this point, the parity of the center six columns, read left to right, is “111100,” which is the representation of our first ciphertext letter. The receiver is now able to read the board parity and obtain the ciphertext letter.

Encryption continues from the current position. The sender can now begin to represent the next ciphertext letter on the chessboard. The ciphertext encoding of the second plaintext

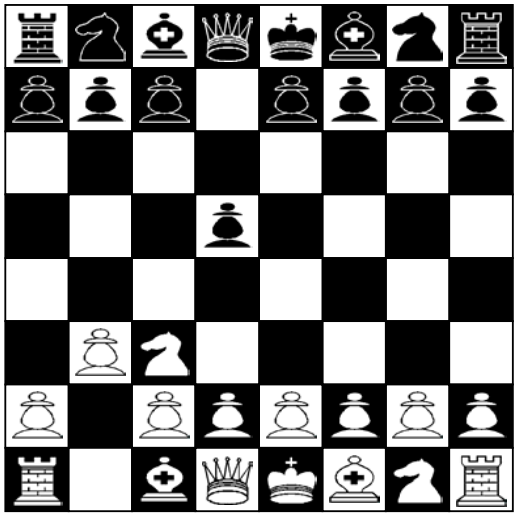
letter is “101100.” Note that the only difference between this and the current column parity string is the parity of the second bit, which corresponds to the second column. To change this, black only has to move one pawn forward (Figure 3, a). Now the column parity string is equal to the representation of the second letter. Once again, the receiver can read the column parity string, obtain the ciphertext, and recover the plaintext message.

a) Initial Board Position
Parity = “000000”



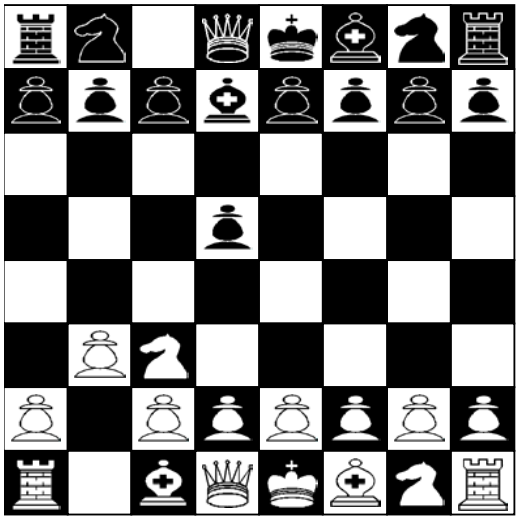
b) White pawn advances one square
Parity = “000000”

c) Black pawn advances two squares
Parity = “000000”

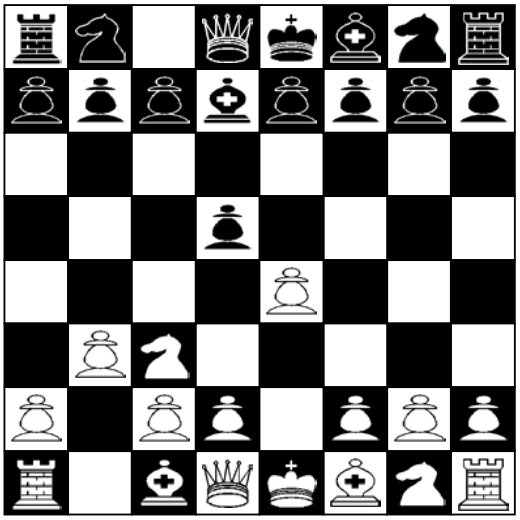


d) White knight advances to right
Parity = “110000”

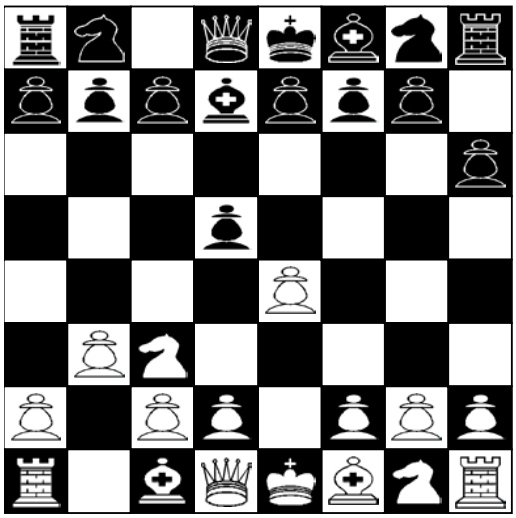
Figure 1 – Example of encoding and transmitting "cs" on the chessboard



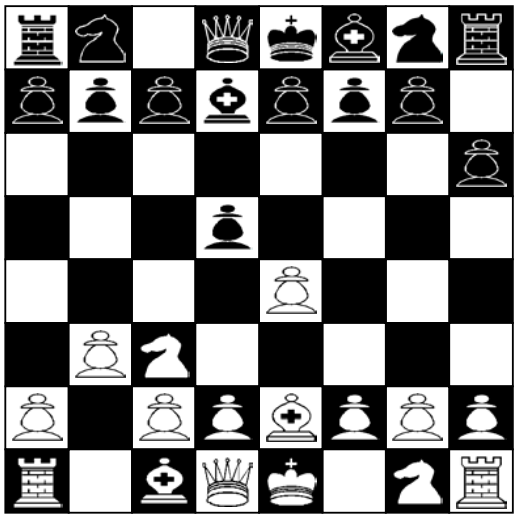
a) Black bishop advances to right
Parity = “111000”



b) White pawn advances two squares
Parity = “111000”



c) Black pawn advances one space
Parity = “111000”



d) White bishop advances to the left
Parity = “111100”

Figure 2 – Example of encoding and transmitting "cs" on the chessboard

a) Black pawn advances one square
Parity = “101100”

Figure 3 – Example of encoding and transmitting "cs" on the chessboard

6.0 Shared Key

As mentioned earlier, this cipher relies on a shared secret key between the sender and receiver of the secret message. After much debate, it was decided that the most logical source for this key would be an encoding of one or more board positions into a sequence of bits. This bit sequence will be the shared secret key, and 6-bit chunks of the key will be taken at a time and used as the letter key described in the encryption and decryption algorithm. This derivation of a key from board positions raises the following two questions that will be answered in the following order: How is a board position encoded? Where do these board positions come from? For the purpose of explanation, only one board position will be considered. Note, that this scheme is scalable; multiple board positions can be encoded and then aggregated to make longer keys as desired.

6.1 Shared Key Generation Algorithm

The motivation when choosing the process used to encode a board position into bits for the shared secret key was to maximize the amount of unique information obtained from a given board position.

To this end, we devised an encoding scheme to take a single board position and derive a 96-bit sequence from it. The encoding scheme that we created is simple, yet it provides an acceptable level of security that is dependent on the secrecy of the board position it was encoded from. A complete analysis of the security of this encoding scheme and the 96-bit sequence that it generates is given in the Section 7.0 of this report.

Encoding a board position into the 96-bit sequence is relatively simple. For purposes of explanation, assume that a chessboard containing the board position to be encoded is present (in practice, this will always be the case). The board is turned such that the White origin (the side that all the White pieces began on) is near to the encoder, and the Black origin is farther away. Start at the square in the upper leftmost corner, and proceed square-by-square going left-to-right across each column, and then moving down one row and repeating. While examining each square in this fashion, the first time a chess piece is encountered, a three bit binary string is recorded using Table 2 to determine which three bits are recorded.

Table for Encoding 96-bit Key Based on a Given Chess Board Position			
Square Color	*Piece Color*	*Piece*	*Binary Encoding*
Black	Black	Pawn	000
Black	Black	Other	001
Black	White	Pawn	010
Black	White	Other	011
White	Black	Pawn	100
White	Black	Other	101
White	White	Pawn	110
White	White	Other	111

Table 2 – Encoding 96-bit key from board position

For all subsequent pieces encountered during the remainder of the left-to-right, top-to-bottom iterative sweep of the board, the appropriate three-bit string is appended to the right side of the recorded sequence using Table 2. The result of this encoding process is a 96-bit string for every board position it is applied to, which assumes that the board position contains all 32 chess pieces. Clearly, not all board positions will encode to different strings, a fact that will be discussed further in Section 7.0 of the report.

6.2 Selecting a Board Position

It makes sense that the board position used to create the shared secret key should come from a game played by the sender and receiver. This has an enormous benefit over choosing an arbitrary board position, because it is believed that above average chess players (which our two heroes most certainly are) have the ability to remember the entire proceeds of any game they have played. Couple this with the fact that this particular game might someday save our heroes' lives, and it is a pretty safe assumption that they won't forget a move. Because the security of the key relates largely to the secrecy of the board position chosen to make the key, it is highly undesirable for Givus and Albee to have to generate a new key in captivity and under the watchful eye of the security guards. However, given that Givus and Albee have never worked together, they have never had the chance to establish a shared key. Also, since they cannot communicate without being overheard, they cannot agree to use well-known chess games (for example, Kasparav vs. Deep Blue, May 11th 1997) for keys. Thus, they prisoners have no choice but to use the key generation algorithm and rely on the incompetence of the Honduran guards to keep their key safe.

Another convincing argument for choosing the board position from a game played by the sender and receiver is that they have the ability to create any board position that they want. This is important because, in Section 7.0 of the report, it will be shown that the security of

the key is highly dependent on the board position chosen. Additionally, and probably most importantly, the number of pieces in the board position used to create the key dictates the length of the key! If there are only 31 pieces (instead of the usual 32) in the board position chosen to encode the key from, the encoding scheme described in the previous section will only produce a 93-bit key. A more in-depth discussion of the choice of the board position used to encode the key is given in Section 7.0.

7.0 Security Analysis

Our cipher is unique in that it combines the application of steganography and cryptography to hide a message. If an attacker does not realize that a message is being transmitted, then of course they will not be able to obtain any information. If an attacker suspects that there is a transfer of information, but does not know the CipherChess algorithm, then the attacker has almost no chance of breaking the cipher. However, even if the attacker somehow obtains the algorithm, and is aware that a message is being transmitted, the attacker must still crack the underlying security of the encryption. In this case, the security of CipherChess relies solely on the key.

Modern ciphers are extremely complex. An algorithm such as DES relies on confusion and diffusion to hide statistical properties of the plaintext that might be vulnerable to attack. RSA uses math that is too complicated to do by hand to encrypt and decrypt, and relies on math that is too complicated to do by computer (factoring) for security. Since CipherChess is a low-tech cipher, it must be practical to implement by hand in a reasonable amount of time. Because of this, CipherChess cannot rely on complexity for security.

CipherChess uses XOR operations for encryption in the same manner as a one-time pad. Because of this, if the key length is at least as long as the message, then the cipher is

perfectly secure, given a random key. This is not impractical for CipherChess, as it is for a one-time pad, since we assume that the message will be short to begin with, and messages will only be sent infrequently, in dire circumstances. There is an obvious tradeoff between key length (relative to message length) and security. Having more message bits than key bits will essentially result in a Vigenère cipher. This is still fairly secure, given a short message and a relatively long key.

As mentioned above, CipherChess has the potential to behave like a one-time pad, giving perfect security. It is important to note, however, that in order to achieve this behavior, a completely random key must be used. The key generation scheme described in this report DOES NOT produce completely random keys, as it admittedly cannot even produce all 96-bit keys. If perfect security is desired, users of CipherChess must share a truly random key. However, because the use of CipherChess will not necessarily be planned ahead of time, it is unlikely that the users will have time to set up a shared key. In this case, some security must be sacrificed in order to gain the convenience of dynamic key generation. We will now analyze the security of the key generation scheme described in Section 6.0.

7.1 Key Security

Given a completely random board position, all 2^{96} 96-bit keys are not possible using our scheme. This is because of the way in which our encoding algorithm distinguishes between different pieces. The pieces are divided into four major categories: White Pawns, Black Pawns, White Non-Pawns, and Black Non-Pawns. All eight pieces in each category are indistinguishable from one another. Note also that these are the same categories used for encoding a board position in Section 6.1. Depending on the color of the square it is on, a piece from a specific category will generate one of two distinct strings. Since there are only

eight pieces in each category, each unique three-bit string can only appear at most eight times in the larger 96-bit string. However, these 3-bit strings *can* appear in any order in the key. Given this information, we can approximate the actual number of 96-bit keys that result from our algorithm.

First, we define cases. Each case is defined by how many pieces of each category are on white squares, and how many are on black squares. An example of a case would be the situation where all the chess pieces are on black squares. Another case is the situation where all the chess pieces are on black squares except one white pawn, which is on a white square. Since each category of pieces has nine possible states (all pieces on black squares through no pieces on black squares), and there are four categories of pieces, the total number of cases is 9^4.

For the case that all pieces are on black squares, the total number of possible unique strings is as follows:

$$\binom{32}{8} \times \binom{24}{8} \times \binom{16}{8} \times \binom{8}{8} = 9.96\,10^{16}$$

This is the same as the number of ways to place the eight indistinguishable pieces from one category (in our case, 3-bit strings) in 32 slots (the position each 3-bit string occupies in the 96-bit string) times the number of ways to place eight other pieces (from a different category) in the remaining 24 slots, etc.

It should be apparent that this case (with all pieces on black squares) is the simplest (base) case, and will result in the least number of unique strings. Moving one piece from a black square to a white square will result in the following number of unique strings, which is greater than the first case:

$$\binom{32}{1} \times \binom{31}{7} \times \binom{24}{8} \times \binom{16}{8} \times \binom{8}{8} = 7.96 \times 10^{17}$$

This number is greater due to the fact that

$$\binom{n}{q} \times \binom{n-q}{p} > \binom{n}{q+p}$$

for any n, q, and p such that $n > q + p$.

Now assume that half the pieces of a given type are on white squares, and the other half on black squares. The total number of unique strings is as follows:

$$\binom{32}{4} \times \binom{28}{4} \times \binom{24}{4} \times \binom{20}{4} \times \binom{12}{4} \times \binom{8}{4} \times \binom{4}{4} = 2.39 \times 10^{24}$$

This is the case that results in the greatest number of unique strings.

To obtain a low estimate on the number of unique keys, we take the case with the lowest number of keys and assume that each case has at least that many. This gives the total number of unique keys possible under our scheme to be at least equal to:

$$9^4 \times \left(9.96 \times 10^{16}\right) = 6.53 \times 10^{20}$$

This is approximately the same as 2^{69}. This is the total number of keys that a brute force attack would have to search. If we assume that a brute force attack could look at one trillion keys per second, it would take approximately 21 years to search all possibilities. This does not seem very secure, given that DES can be broken relatively easily, and our key in effect only has five more key bits. However, the numbers presented are a low estimate of the number of keys.

The real number of unique keys may be substantially higher. There are four distinct cases wherein one of the 32 pieces is on a white square and the rest of the pieces are on black

squares. Compare this to the one distinct case where all the pieces are on black squares. The number of unique strings that result from each of these four cases is an order of magnitude higher than the base case. If we take this into account, and assume that the average number of unique strings per case is equal to the number obtained from one of these four cases, a brute force attack becomes eight times harder.

Due to lack of funding, we were not able to compute the exact number of keys our scheme can generate. The only method we could think of to do so involved counting every single case, a task beyond our means. A more in-depth analysis would compute the exact number of keys, and thus provide a more accurate picture of the algorithm's susceptibility to a brute-force attack.

Even if an attacker is not willing to go through a brute force attack, there may be other ways of attacking the key. These must rely on the fact that the chessboard configuration used to generate the key may not be entirely random. The two people setting up the key can do a lot to avoid this. This issue is covered in the next section.

7.2 Key Selection Analysis

Given a completely random board position, the key generated using our algorithm will be completely random. However, a random board position is not easy to achieve in a real game of chess. The key-generation algorithm is supposed to make the key easier to remember by using an actual game, since chess players can recall games of chess they have played. There is a tradeoff between the randomness of the key and the predictability of the game used to generate it.

For instance, if a snapshot of a classical chess game is used for the key, then the key will not be entirely random. Chess games tend to follow certain trends that, although complex, would make it easier to find the key. After observing several professional chess games, we

noticed one simple trend: the pawns in the outermost columns rarely move. Such trends can be avoided, however. If the key is chosen from a game played between the two people establishing a key, then they can work together to make the game go in unusual directions. This will effectively mask the trends present in a real game of chess.

To obtain a good key, a game should be played for several turns, to ensure that most of the pieces have moved from their original position, in order to randomize most of the key bits. Moving an individual piece can at most change the value of just three key bits, and then swap the location of those bits with three others. If the key is chosen after each player has only had one move, then only a few bits will be unpredictable. This would be a major security flaw.

If our cipher is used as intended, and the key length is greater than the message length, then it is only vulnerable to an attack on the key. This is because it functions essentially as a one-time pad. Thus, achieving a good (pseudorandom) key is essential to security. The best solution is to randomly generate a key and memorize it. Using our scheme to generate a key from a chessboard compromises some security, but enhances ease of use and key maintenance.

8.0 Conclusion

This report presents CipherChess as a viable solution to the problem that Givus and Albee faced in the heart of the Honduran jungles. However, the uses of CipherChess need not be limited to secret agents in top secret drug cartel work camps. In the small family of low-tech ciphers, CipherChess stands above the rest in terms of its ease of use and steganographic qualities. Its ability to function both as a one-time pad and a less-secure stream cipher provides the user with a flexible scheme that can be adapted as the situation dictates.

Analysis of the cipher revealed that the single point of weakness in the cipher was the key generation algorithm. It was understood from the onset that requiring the encryption, transmission, and key generation algorithms to follow strict chess rules may expose CipherChess to possible security flaws. More research is needed to determine those board positions that generate more "random" keys.

Overall, CipherChess provides a simple and flexible method for facilitating secret communication between two parties in a potentially hostile environment. Look for other exciting products coming soon from Group 2, such as: CipherSoccer, CipherSynchronizedSwimming, and CipherThumbWrestling!

9.0 Bibliography

Beker, Henry.Cipher Systems: The Protection of Communications.New York, NY: John Wiley & Sons, Inc, 1982

Gaines, Helen Fouche.Cryptanalysis.Boston, MA: American Photo-Graphic Publishing Co, 1941

Schneier, Bruce.Applied Cryptography.New York, NY: John Wiley & Sons, 1994

"U.S. Chess Federation."New Windsor, NY: United States Chess Federation Website.Online.Internet.Available: http://www.uschess.org/