Spring 2014

Cryptanalysis of Homophonic Substitution-Transposition Cipher

Jeffrey Yi
San Jose State University

Cryptanalysis of Homophonic Substitution-Transposition Cipher

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Jeffrey Yi

May 2014

The Designated Project Committee Approves the Project Titled

Cryptanalysis of Homophonic Substitution-Transposition Cipher

by

Jeffrey Yi

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2014

Mark Stamp	Department of Computer Science
Thomas Austin	Department of Computer Science
Richard M. Low	Department of Mathematics

ABSTRACT

Cryptanalysis of Homophonic Substitution-Transposition Cipher

by Jeffrey Yi

Homophonic substitution ciphers employ a one-to-many key to encrypt plaintext. This is in contrast to a simple substitution cipher where a one-to-one mapping is used. The advantage of a homophonic substitution cipher is that it makes frequency analysis more difficult, due to a more even distribution of plaintext statistics. Classic transposition ciphers apply diffusion to the ciphertext by swapping the order of letters. Combined transposition-substitution ciphers can be more challenging to cryptanalyze than either cipher type separately.

In this research, we propose a technique to break a combined simple substitution-column transposition cipher. We also consider the related problem of breaking a combination homophonic substitution-column transposition cipher. These attacks extend previous work on substitution ciphers. We thoroughly analyze our attacks and we apply the homophonic substitution-columnar transposition attack to the unsolved Zodiac-340 cipher.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The classic simple substitution cipher is one of the most basic forms of cryptography. Encryption with a simple substitution cipher involves a one-to-one substitution of ciphertext for plaintext characters based on a key. The simple substitution is easy to use; however, in addition to its ease of use is the ease with which it can be broken using letter frequency analysis. There have been different methods of breaking simple substitution ciphers [5, 14, 18, 21] but this paper will focus on a fast attack that uses a hill climb search [13]. An extension to the simple substitution cipher is the homophonic substitution cipher, which was developed to prevent a statistical attack [7]. Rather than use a one-to-one mapping for plaintext characters to ciphertext characters, homophonic ciphers use a one-to-many key so that a single plaintext character can map to multiple ciphertext characters. What this also means is that the key will contain more than 26 alphabetic characters. Previous work done on an attack against the homophonic substitution cipher has produced methods to attack homophonic ciphertext [8, 15]. This technique is based on an algorithm developed to quickly solve simple substitution ciphers [13] and recover at least 80% of the text for ciphertext of at least 300 characters, which is the minimum amount needed to be able to manually recover the rest of the key. It extends this work by using three nested hill climbing steps rather than only one. The algorithm showed good results with at least 1000 characters of text and less than 42 symbols.

Transposition ciphers encrypt plaintext by permuting the text based on a provided key, thus providing diffusion [11]. Previous research on the cryptanalysis of transposition ciphers has used a variety of heuristic searches to attack and deter-

mine the key of the cipher [6, 9, 10, 11, 17, 20]. Most of this research centered on using simulated annealing, genetic algorithms, and tabu search to determine the key [6, 9, 10, 11, 20]. Simulated annealing works by generating an initial solution and randomly altering it through a number of iterations and choosing the key that generates the best score based on putative text's digram statistics. Tabu search is similar but generates and maintains a list of keys and uses the best of those keys to perform the perturbation step and find new solutions. Finally, the genetic algorithm produces a pool of possible keys and each iteration generates a new pool of children based on the pairing of parents and scores all of these keys.

The Zodiac-340 is a famous cipher sent by the Zodiac killer in 1969 that has never been decrypted. The previous cipher, known as the Zodiac-408, was sent to three newpapers and was solved within a week [2]. It contained 408 characters with 53 distinct symbol and was encrypted with a homophonic substitution cipher. The Zodiac-340 contains 340 characters with 62 symbols and many approaches to solving it assume it to be a homophonic cipher as well [3, 4, 7, 8, 12, 16]. However, based on tests performed on the Zodiac-340 cipher using the efficient method mentioned above, a solution has not been found. This suggests that the cipher may not only use a homophonic cipher but also another encryption method, such as transposition.

The aim of this project is to extend the previous work done on cryptanalysis of homophonic and transposition ciphers by finding an attack on a homophonic-transposition cipher. In order to test the effectiveness of the attack, ciphertext of varying lengths and encryptions will be generated and tested against and with a successful series of tests completed, the algorithm will be used against the Zodiac-340 cipher. Chapter 2 will provide background on past work. Chapter 3 will introduce a new data structure that will be used in following chapters also present how this

new data structure can be used with an existing attack on simple substitution and a new attack on columnar transposition will be laid out in Chapter 4. The modified attack from Chapter 3 and the new attack in Chapter 4 will be combined in Chapter 5 to attack simple substitution-columnar transposition ciphers. Chapter 6 will extend the work in Chapter 5 to incorporate the homophonic attack in place of the simple substitution one. Chapter 7 will present the results of experiments done on these new attacks and finally, Chapter 8 will present conclusions of the work done.

CHAPTER 2

Background

In this section, we will discuss previous work done on simple substitution, homophonic substitution, and columnar transposition ciphers. Next, the history of the Zodiac-340 cipher will be discussed.

2.1 Simple Substitution

Substitution ciphers are a class of classic ciphers that performs encryption by replacing plaintext symbols with ciphertext symbols. There are a variety of ways to perform these substitution. The simplest of these is appropriately named the simple substitution cipher.

The simple substitution cipher encrypts plaintext by replacing each character with a ciphertext character using a one-to-one key [19]. Table 1 presents an example of a simple substitution key.

Table 1: Simple Substitution Key

plaintext	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
ciphertext	P E Z Q F A O W R B M S Y U T D C V H L N I G J K X

Using the key in Table 1, the ciphertext in Table 2 is generated from the corresponding plaintext.

This is one of the most basic forms of cryptography and with this simplicity comes an easy way to break the cipher.

Table 2: Simple Substitution Encryption

plaintext	THISISATEST
ciphertext	LWRHRHPLFHL

Assuming the key is made up of 26 symbols based on the English alphabet, the keyspace is quite large. If an exhaustive key search is performed, then there are $26! \approx 2^{88}$ possible keys and requires an average of $2^{88}/2 \approx 2^{87}$ keys to be tested before the correct key is found. Assuming that a million keys are tested per second, then it would take $2^{87}/10^6 = 1.55 \times 10^{20}$ seconds on average to find the correct key. This equates to 4.9×10^{12} years making an exhaustive key search impractical [8]. While this large keyspace may seem daunting, the simple substitution cipher has a weakness that greatly reduces the amount of work an attacker must do in order to find the corresponding key.

Since a one-to-one relationship is used, the frequencies of each letter will not change in value but will instead move from the plaintext letter to the ciphertext letter. For example, in the ciphertext in Table 2, the repeated use of L in the ciphertext means that they all correspond to the same plaintext letter. In the English language, the letter E is the most common letter and what this means for the ciphertext is that the most frequent letter in the text most likely corresponds with a plaintext letter E. This process can be applied for all letter frequencies and generate a putative key [19]. This putative key gives a reasonable starting point to find the final key and greatly reduces the keyspace that needs to be searched.

In order to determine if a new key is better, a scoring function is needed to test the putative key and compare it with the previous solution. This can be done by comparing the digram statistics of the English language with the digram frequencies

of the putative text. Using a lengthy text, such as Moby Dick, the matrix E can be constructed and represent the expected digram statistics for English. The matrix D contains the digram frequencies from the decrypted ciphertext. The score is then calculated using the following formula:

$$f(t) = \sum_{i,j} |D_{ij} - E_{ij}|. \tag{1}$$

Now the question becomes, how do we update this key to find a better solution. Using a hill climb heuristic search provides a way to manipulate the key and find better solutions to decrypt the ciphertext with [13]. The general algorithm is presented in algorithm 1.

The resulting key may not be the absolute solution depending on the amount of ciphertext used to test with but it will hopefully be enough for readable text to be decrypted and can then be adjusted manually to find the true key.

One drawback of this method is the amount of decryption that must be done. With each putative key, the ciphertext must be decrypted and the digram frequencies must be calculated. However, an interesting observation about the change in the D matrix between key changes shows that this decryption does not need to occur for every key. Instead, it has been observed that with each key swap, the corresponding rows and columns of the D matrix are simply interchanged and the new matrix represents the digram frequency matrix of the putative text after decrypting with the new key [13].

This can demonstrated using a smaller alphabet of

$$\texttt{E}, \texttt{T}, \texttt{I}, \texttt{S}, \texttt{H}, \texttt{R}, \texttt{L}, \text{and } \texttt{K}.$$

With the ciphertext below, a putative key can be generated based on the letter

Algorithm 1 General Hill Climb Attack

```
Let a = 1 and b = 1.
Get initial key based on letter frequencies in the ciphertext.
Construct E matrix based on English language text.
Decrypt the ciphertext with the putative key and construct D matrix.
Score the putative key with equation (1).
while b ≠ 26 do
  if a + b ≤ 26 then
    Let tempkey = key.
    Swap element a and b in tempkey.
    Decrypt and score with tempkey.
    if newscore < score then
      score = newscore
      key = tempkey
      a = 1
      b = 1
    else
      tempkey = key
      increment a
    end if
  else
    a = 1
    increment b
    tempkey = key
  end if
end while
```

frequencies in the text.

RIHILKHIERSKILEKKLKTRSKHRIHILKHLREIRTRSKLKTRSKHHLEKRS.

The counts for each ciphertext letter is as follows:

E	T	I	S	H	R	L	K
4	3	7	5	7	9	7	11

Based on these counts, the initial key in Table 3 is determined. Using this key, the following putative plaintext can be decrypted

TIHISEHILTREISLEESEKTREHTIHISEHSTLITKTRESEKTREHHSLETR.

Table 3: Initial Putative Key

plaintext	ETISHRLK
ciphertext	KRILHSET

This putative plaintext generates the digram frequency matrix in Table 4.

Table 4: Initial Digram Matrix

	E	T	I	S	H	R	L	K
E	1	1	1	2	4	0	0	2
T	0	0	2	0	0	5	1	1
I	0	1	0	3	2	0	1	0
S	4	1	0	0	0	0	2	0
H	0	1	3	2	1	0	0	0
R	4	0	0	0	0	0	0	0
L	2	1	1	0	0	0	0	0
K	0	3	0	0	0	0	0	0

The next step is to perform a swap on the putative key and generate a new digram matrix. The new key would be

plaintext	ETISHRLK
ciphertext	RKILHSET

This key decrypts the ciphertext to

EIHISTHILERTISLTTSTKERTHEIHISTHSELIEKERTSTKERTHHSLTER.

And this plaintext generates the digram frequency matrix in Table 5.

As can be seen in Table 5, the matrix is an update of the matrix in Table 4. It is updated by simply swapping the corresponding rows and columns based on the swap performed on the key. So rather than having to decrypt the ciphertext repeatedly,

Table 5: Updated Digram Matrix

	E	T	I	S	H	R	L	K
E	0	0	2	0	0	5	1	1
T	1	1	1	2	4	0	0	2
I	1	0	0	3	2	0	1	0
S	1	4	0	0	0	0	2	0
H	1	0	3	2	1	0	0	0
R	0	4	0	0	0	0	0	0
L	1	2	1	0	0	0	0	0
K	3	0	0	0	0	0	0	0

the digram matrix can be updated in this manner and be used in the scoring process. This fast attack is presented in algorithm 2.

The work needed for the Jakobsen algorithm in the best case is $N + \binom{26}{2} \approx \binom{26}{2}$ as opposed to $N\binom{26}{2}$ when each putative key must be used to decrypt the ciphertext [8]. This best case assumes that the algorithm does not need to reset and start the search from the beginning. In the worst case, the algorithm would require work of $\binom{26}{2}!$.

The homophonic substitution cipher, another substitution cipher, is explored next as well as a fast attack against it. This attack is based on the Jakobsen attack and requires consideration for the more complex nature of the cipher.

2.2 Homophonic Substitution

Homophonic substitution ciphers attempt to overcome the frequency attacks that simple substitution ciphers are vulnerable to by using a key with a one-to-many relationship rather than a one-to-one key. Each plaintext letter can be encrypted using one or more symbols, which allows more frequently used letters to be encrypted with more symbols and even out the distribution of ciphertext frequencies. Using

Algorithm 2 Jakobsen Attack

Let $a = 1$ and $b = 1$.
Get initial `key` based on letter frequencies in the ciphertext.
Construct E matrix based on English language text.
Decrypt the ciphertext with the putative key and construct D matrix.
Score the putative key with equation (1).
while $b \neq 26$ **do**
  if $a + b \leq 26$ **then**
    Let `tempkey` = `key`.
    Let $D' = D$
    Swap element a and b in `tempkey`.
    Swap rows and columns a and b in D'.
    Score with D'.
    if `newscore` < `score` **then**
      `score` = `newscore`
      `key` = `tempkey`
      $D = D'$
      $a = 1$
      $b = 1$
    else
      `tempkey` = `key`
      $D' = D$
      increment a
    end if
  else
    $a = 1$
    increment b
    `tempkey` = `key`
    $D' = D$
  end if
end while

the key below, the text `THISISATEST` can be encrypted to `GUVF1FN4RF5`. Due to the one-to-many relationship in the key, the letter T can be encrypted to L, 4, or 5 [8].

plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
ciphertext	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
	8				9				1			6								4						
					3															5						

With n possible ciphertext symbols for 26 plaintext letters and each plaintext letter has at least one ciphertext symbol associated with it, the key space is

$$\binom{n}{26} 26! 26^{n-26} < 26^n \approx 2^{2.7n}.$$

An exhaustive search for a key of length $n = 100$ would require $2^{470}/2^{20} = 2^{450}$ seconds, or 9.2×10^{127} years. For 62 symbols, which is the number used in the Zodiac-340 cipher, it would take 1.59×10^{74} years. Just as with the simple substitution cipher, it is not feasible to perform an exhaustive search to find the correct key so a statistical attack, similar to the Jakobsen attack, is required to find a solution [8].

Based on the Jakobsen algorithm, an efficient attack was created to determine the key for a homophonic ciphertext. This algorithm is broken up into three layers: an outer layer, random key layer, and inner layer [8].

2.2.1 Outer Hill Climb

The outer hill climb layer is designed to tackle the one-to-many property of homophonic ciphers. Assuming that there are n ciphertext symbols, and $n_a, n_b, \ldots, n_z$ are the number of ciphertext symbols for each plaintext letter, then $n_a + n_b + \cdots + n_z = n$. The hill climb approach used is slightly different from the approach used in the Jakobsen algorithm, in that the pairs of counts are not swapped but instead incremented and decremented by 1 [8].

The first step of this layer is to calculate the D_C matrix. This matrix holds the digram frequency statistics for the ciphertext and will be used later on in the other layers to perform the swapping and scoring procedures [8].

The next step is to find an initial distribution of values that constrains to $n_a +$

$n_b + \cdots + n_z = n$. Since the goal of the homophonic cipher is to evenly distribute the ciphertext symbol statistics, the initial starting distribution should attempt to get this even distribution based on English letter frequencies. So for the letter E, 12% of the ciphertext would be the value of n_e, and 9% would correspond to the letter T [8]. Table 6 shows the possible initial distributions for different key lengths that can be used in this layer. Once an initial distribution is determined, then it is sent to the random key layer, as outlined in algorithm 3.

Table 6: Initial Frequency Distribution

n	n_e	n_t	n_a	n_o	n_i	n_n	n_s	n_r	n_h	n_d	n_l	n_c	n_u	n_m	n_f	n_w	n_g	n_y	n_p	n_b	n_v	n_k	n_x	n_j	n_q	n_z
26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
27	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
35	4	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
45	5	4	3	3	3	3	3	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
55	7	5	4	4	4	3	3	3	3	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
65	8	6	5	5	5	4	4	4	4	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
75	9	7	6	6	5	5	5	4	4	3	3	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
85	11	8	7	7	6	6	5	5	5	3	3	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1
95	12	9	8	7	7	7	6	6	5	4	4	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1

A swapping process is needed to make slight changes to this initial distribution to determine the best distribution. This swapping method, shown in algorithm 4, entails incrementing and decrementing adjacent pairs based on letter frequency. For example, n_e would increment by one and n_t would decrement by one in the first swap. If the score improves, then this new distribution is kept. Else, the distribution is reset and n_e decrements and n_t increments. This process occurs with adjacent pairs, then pairs of distance 2, then pairs of distance 3, and so on [8].

Algorithm 3 OuterHillClimb

```
1: global K = bestInitKey = bestKey = NULL.
2: parse ciphertext to determine D_C.
3: initialize n_a, n_b, ..., n_z as in Table 6.
4: (m_1, m_2, ..., m_26) = (n_a, n_b, ..., n_z)
5: bestScore = RandomKeyLayer(m_1, m_2, ..., m_26)
6: bestKey = bestInitKey
7: for i = 1 to 25 do
8:    for j = 1 to 26 − i do
9:       (m'_1, m'_2, ..., m'_26) = (m_1, m_2, ..., m_26)
10:      OuterSwap(m'_j, m'_{j+i})
11:      score = RandomKeyLayer(m'_1, m'_2, ..., m'_26)
12:      if score < bestScore then
13:         (m_1, m_2, ..., m_26) = (m'_1, m'_2, ..., m'_26)
14:         bestScore = score
15:         bestKey = bestInitKey
16:      else
17:         (m'_1, m'_2, ..., m'_26) = (m_1, m_2, ..., m_26)
18:         OuterSwap(m'_{j+i}, m'_j)
19:         score = RandomKeyLayer(m'_1, m'_2, ..., m'_26)
20:         if score < bestScore then
21:            (m_1, m_2, ..., m_26) = (m'_1, m'_2, ..., m'_26)
22:            bestScore = score
23:            bestKey = bestInitKey
24:         end if
25:      end if
26:   end for
27: end for
```

Algorithm 4 OuterSwap

Input: m_i, m_j

```
1: increment m_i
2: decrement m_j
```

2.2.2 Random Key Layer

This layer, algorithm 5, uses the distribution calculated in the outer hill climb layer and randomly generates initial keys based on this distribution [8].

Due to the fact that the keys are randomly generated, there is no confidence that

Algorithm 5 RandomKeyLayer

Input: $n_a, n_b, \ldots, n_z$

1: `bestInitScore` $= \infty$.
2: **for** $r = 1$ to R **do**
3: randomly initialize $K = (k_1, k_2, \ldots, kn)$ satisfying $n_a, n_b, \ldots, nz$.
4: D_P = digram matrix from D_C and K.
5: `initScore` = `InnerHillClimb`(D_P).
6: **if** `initScore` $<$ `bestScore` **then**
7: $(m_1, m_2, \ldots, m_{26}) = (m'_1, m'_2, \ldots, m'_{26})$
8: `bestInitScore` = `initScore`
9: `bestInitKey` $= K$
10: **end if**
11: **end for**

the initial key is a good starting point for the hill climb process in the inner hill climb layer. So multiple keys need to be tested in order to ensure that the optimum score is found. Based on previous testing of the attack, it was determined that having a max R value of 40, or 40 iterations, provided similar results to an R of 100 [8].

Once a key is generated, it is used to calculate D_P, which is the 26×26 matrix that represents the digram frequencies of the initially decrypted ciphertext. This matrix along with D_C are used in the inner hill climb layer to perform the swaps and scoring [8].

2.2.3 Inner Hill Climb

The inner layer performs a hill climb search on a putative key in much the same way the Jakobsen algorithm, as shown in algorithm 6. With the Jakobsen attack, when elements of the key are swapped, the corresponding rows and columns are swapped in the digram matrix, D. Since the homophonic key can have the same letter appear multiple times, the process must be modified to account for the different situations that can arise [8].

Swapping of the same letter is avoided since there would be no change to the key. For situations involving swapping letters that only appear once in the key, then the corresponding rows and columns of D_P can be swapped just as with the Jakobsen algorithm. The more complex situation arises when the different letters that appear multiple times are swapped. For example, with the key in Table 7, shows that both 2 and 5 decrypt to `E`. Suppose the key in Table 7 is applied to ciphertext (2).

Table 7: Inner Hill Climb Putative Key

ciphertext	0 1 2 3 4 5 6 7 8 9
plaintext	I L E K T E R T H S

```
0949824950729852286107230939824805901076861076448520 7          (2)
```

The resulting plaintext (3) is decrypted based on this putative key.

```
ISTSHETSEITESHEEEHRLITEKISKSHETHIESILITRHRLITRTTHEEIT.          (3)
```

The ciphertext generates the 10×10 digram matrix, D_C,

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	1	0	5	0	2
1	3	0	0	0	0	0	0	0	0	0
2	1	0	1	1	2	0	0	0	1	1
3	1	0	0	0	0	0	0	0	0	1
4	0	0	0	0	1	0	0	0	2	2
5	1	0	2	0	0	0	0	0	0	1
6	0	2	0	0	1	0	0	0	1	0
7	0	0	2	0	0	0	2	0	0	0
8	1	0	2	0	0	2	2	0	0	0
9	1	0	0	1	1	1	0	0	3	0

Using the decrypted plaintext, the digram matrix, D_P is generated as follows

	E	T	I	S	H	R	L	K
E	3	2	2	2	1	0	0	1
T	2	1	0	2	2	2	0	0
I	1	5	0	2	0	0	1	0
S	1	1	1	0	3	0	0	1
H	4	0	1	0	0	2	0	0
R	0	1	0	0	1	0	2	0
L	0	0	3	0	0	0	0	0
K	0	0	1	1	0	0	0	0

If H and S are swapped then the columns and rows of matrix D_P just need to be swapped. However, if the first E and S are swapped, since there are multiple E's in the key, the matrix D_P matrix needs to be updated using D_C. This does not require a full recalculation but only an update of the affected rows and columns [8].

Algorithm 6 InnerHillClimb

Input: key K

innerScore $= f(t)$ from equation (1).
for $i = 1$ to $n - 1$ **do**
  for $j = 1$ to $n - i$ **do**
    $K' = K$
    swap(k'_j, k'_{j+i})
    D' = digram matrix from K' using D_P and D_C
    score $= ft$
    if score $<$ innerScore **then**
      innerScore $=$ score
      $K = K'$
      $D_P = D'$
    end if
  end for
end for

Once the inner hill climb is completed, the best key and score is returned to the random key layer and is repeated with a unique initial key [8].

2.2.4 Work Factor

The work required for this attack is based on the work for each layer. The outer hill climb layer deals with swaps of pairs similar to the swaps performed by the Jakobsen algorithm. However, these swaps can occur twice, where a pair can increment one and decrement the other, and vice-versa. So the amount of work is $2 \cdot \binom{26}{2}$ [8].

For the random key layer, there will always be 40 random keys generated and tested with in the inner hill climb. In the inner hill climb, the amount of work done is at most $\binom{n}{2}$, where n is the total number of ciphertext symbols. So for the entire attack, the total work would be

$$80 \cdot \binom{26}{2} \cdot \binom{n}{2} < 2^{15} n^2.$$

This work is based on the work needed to calculate the score based on equation (1) and does not include the work needed to recalculate the D_P matrix where necessary. Due to this more complex task, the time to complete each swap of the attack is not the same as with the Jakobsen algorithm and overall, the work is $64n^2$ greater than with the Jakobsen attack [8].

2.3 Transposition

Transposition ciphers encrypt plaintext by swapping the letters around rather than replacing plaintext symbols with ciphertext symbols. In particular, columnar transposition ciphers moves characters around in a given period, with this pattern occurring for every period. An easier way to visualize this is to place the plaintext into a matrix where the number of columns represents the period or key size [19]. For example, the plaintext `THISISPLAINTEXT` can be placed into a matrix with period of

5.

T	H	I	S	I
S	P	L	A	I
N	T	E	X	T

Based on a key that represents where each column should be moved, the ciphertext can be generated. So for the key:

key	32541

The resulting matrix is:

I	H	I	S	T
L	P	I	A	S
E	T	T	X	N

Three types of attacks have been used to attack columnar transposition ciphers: simulated annealing, genetic algorithm, and tabu search.

2.3.1 Simulated Annealing

Simulated annealing, algorithm 7, is a method that tries to mimic the process of annealing in physics. Annealing is the process of heating a metal to a high temperature and slowly cooling it in order to achieve an optimal energy state. The algorithm works by utilizing the Metropolis criterion, which is a set of conditions that determine whether a new key should be accepted as the current key [9, 10].

Metropolis Criterion:

$$\Delta E < 0$$

$$Probability(E_1 \Rightarrow E_2) = e^{-\frac{\Delta E}{T}}$$

where ΔE is the difference between the new score and the old score. The algorithm works as follows:

Algorithm 7 SimulatedAnnealing

```
Generate random key.
Construct E matrix based on English language text.
Decrypt the ciphertext with the putative key and construct D matrix.
Score the putative key with f(t) from equation (1).
for i = 1 to MAX do
    Let N_succ = 0
    for 100 × P times: do
        tempkey = key
        Swap two random elements tempkey.
        Decrypt and score with tempkey.
        Calculate ΔE = newscore − score.
        if Metropolis Criterion fulfilled then
            key = tempkey
            increment N_succ
        else
            tempkey = key
        end if
    end for
    if N_succ > 10 × P then
        T = T × T_fact
    else if N_succ = 0 then
        Break
    end if
end for
```

In algorithm 7, P is the period of the cipher, N_{succ} is a counter for the number of improved solutions found, and MAX is the maximum number of iterations to perform. T and T_{fact} represent the current temperature and the reduction factor, respectively. The initial value of T is chosen as a large number and is decreased at each iteration. The perturbation step randomly selects two parts of the key and swaps them [9, 10].

The score is calculated using the same formula (1) and technique as the method used with the simple substitution attack. However, rather than simply comparing the score with the previous score, the difference between the two is used in the Metropolis function and fitness of the solution is determined by the result. So if $\Delta E < 0$ or the probability based on the second equation is greater than some percentage, then the solution is accepted [9, 10]. The amount of work required for this algorithm is $100 \cdot \texttt{MAX} \cdot P$. This is in terms of decrypting the ciphertext, generating the D matrix, and calculating the score.

2.3.2 Genetic Algorithm

The genetic algorithm is based on the evolution of species and how parents pass on genetic information to their children. A pool of solutions is generated randomly and updated and maintained through each iteration. The children of the new generation are created by taking parts from each parent to create a new key. From these keys and the keys from the previous generation, the best of the bunch are used to form the new pool that will be used to create the next generation of children [9, 10, 20]. The pseudo-code is presented in algorithm 8, where N is the size of the pool and `MAX` is the number of iterations.

Algorithm 8 GeneticAlgorithm

```
1: Generate a pool of N random keys.
2: Construct E matrix based on English language text.
3: Decrypt the ciphertext with each key and score each one with equation (1).
4: for i = 1 to MAX do
5:     Pair up the keys and perform mating technique.
6:     Perturb each key by randomly swapping two elements.
7:     Decrypt and score with all the keys with equation (1).
8:     Select best N keys from new and old pool.
9: end for
```

The children are created by pairing parents and using each parent as the base for each child. A random number is generated to determine how much of the parent is used for the child. Then the other parent is used to fill in the remaining part of the key so that each child gets some part of its key from each parent. For example, with parents 31254 and 34251 and random number 3, the next generation would be 31245 and 34215. In this example, the first child uses the first three numbers of the first parent and fills in the missing numbers based on the order they appear in the second parent. The second child does the same but uses the second parent as a base.

All of these children and the parents are scored using equation (1) and the best of these keys are used as the new pool and act as the parents for the next generation. This process repeats for `MAX` iterations and the best key after the last iteration is returned [9, 10, 20]. The work factor for this attack is $\texttt{MAX} \cdot N$ and just as with the simulated annealing attack, this is in terms of decrypting the ciphertext, generating the D matrix, and calculating the score.

2.3.3 Tabu Search

Tabu search is similar to both simulated annealing and the genetic algorithm in the way it generates new keys and the pool of solutions it maintains, which is presented in algorithm 9, where N is the list size and `MAX` is the number of iterations. Much like simulated annealing, the perturbation step involves randomly swapping elements in the key. And much like the genetic algorithm, multiple possible solutions are generated and compared [6, 9].

Once the pool is created, the best of the keys is chosen and used in the next iteration. In addition to the best key, the worst is also tracked since it signals to the algorithm once the optimum solution is found. Once the worst solution is less

Algorithm 9 TabuSearch

```
1: Generate pool of N random keys.
2: Construct E matrix based on English language text.
3: Decrypt the ciphertext with each key and score each one with equation (1).
4: Score the putative key with f(t) from equation (1).
5: for i = 1 to MAX do
6:    Select the best and worst keys from the pool.
7:    for j = 1 to N do
8:       Perturb best key by randomly swapping two elements.
9:       if key is not in list then
10:         Add to list.
11:      end if
12:   end for
13:   Decrypt the ciphertext with each key and score each one with equation (1).
14:   Select the best and worst keys.
15:   if worst score < best score then
16:      break
17:   end if
18: end for
```

than the best solution or the maximum number of iterations is reached, then the best solution is returned. Again, as with the other algorithms discussed earlier, the score is calculated using digram frequencies and equation (1) [6, 9]. For this attack, the work factor is $\texttt{MAX} \cdot N$ where each step perturbs the key, decrypts the ciphertext, creates the digram frequency matrix, and scores the key.

2.4 Zodiac-340 Cipher

The Zodiac Killer was a serial killer active during the late 1960s and early 1970s. During this time, he sent letters to newspapers to taunt police and on July 31, 1969, he sent his first cipher, which has been labeled the Zodiac-408, in three parts. This cipher contained 408 symbols with 53 distinct symbols. About a week after it was published, it was cracked by two schoolteachers. This cipher utilized a homophonic substitution cipher and the message read:

"I like killing people because it is so much fun It is more fun than killing wild game in the forrest because man is the most dangerous anamal of all To kill something gives me the most thrilling experence It is even better than getting your rocks off with a girl The best part of it is that when I die I will be reborn in paradice and all the I have killed will become my slaves I will not give you my name because you will try to slow down or stop my collecting of slaves for my afterlife"

In November of the same year, the Zodiac-340 cipher, figure 1, which contained 340 symbols with 62 unique symbols, was sent and published and to this day, it has not been solved [7].

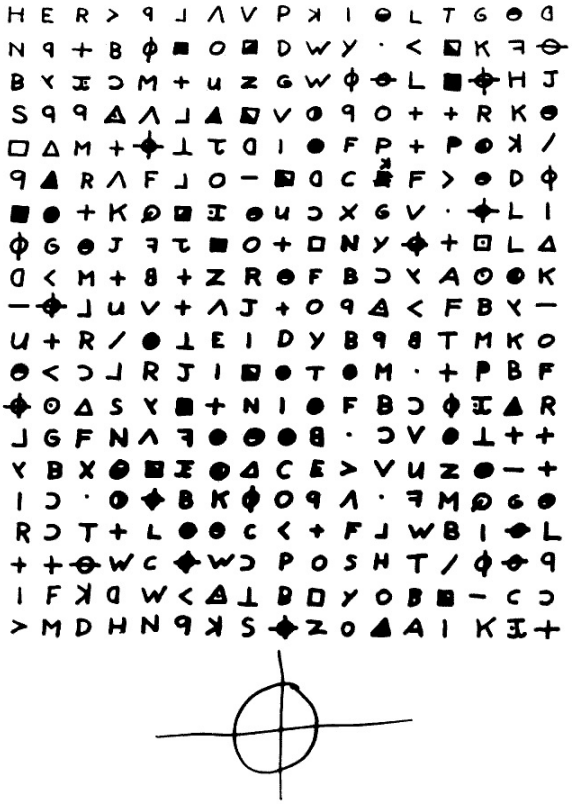


Figure 1: Zodiac-340 cipher [1]

.

Attempts to solve this assume that it is encrypted with a homophonic substitution cipher since the Zodiac-408 was [3, 8]. Previous work done to efficiently attack homophonic substitution ciphers was not successful in decoding this text and the paper concludes that if the text is in fact using this type of cipher, then the issue may be insufficient ciphertext to effectively find language statistics from and that a better way of determining key effectiveness would be necessary by using a better English language model [8].

Another type of attack performed on the cipher used the expectation–maximization algorithm to find the homophonic substitution key. Since the EM algorithm, much like the hill climb search, finds the local optimum, it requires multiple starting points to find the best solution. However, instead of 50 to 100 random starts, the method looked at using a million random keys. In order to speed up the attack, the method uses GPUs to perform the attack in parallel. The attack was able to find the solution to the Zodiac-408 cipher, but was unable to find a solution to the Zodiac-340 cipher, which could mean that the code is encrypted with another cipher in addition to the homophonic substitution cipher [4].

CHAPTER 3

New Digram Data Structure

In order to develop a quick way to perform a hill climb search to crack a columnar transposition cipher, it was necessary to find a way to eliminate the most time intensive step, which was the decryption and digram matrix construction steps. Since the key would change the order of the columns for each perturbation, a data structure to store these column relationships would make it easy to determine the digram statistics for any given key. This data structure would hold the digram frequency matrices of every possible combination of pairs of columns. So for column 1, it can be paired with every other column up to the size of the key, N. In addition to these pairings, if column 1 falls at the end of the row, it will also be paired with the first element of the next row, which means the digram statistics for column 1 and the columns in the next row must also be stored.

So for key $K = (1, 2, 3, 4, 5)$, the digram frequencies can be calculated as

$$D = D_{1,2} + D_{2,3} + D_{3,4} + D_{4,5} + D_{5,1'}.$$

where $D_{5,1'}$ represents the digram frequencies of column 5 and column 1 of the next row, which accounts for the wrap around of the text. Generally, this means that for key (4), the total matrix can be obtained using equation (5). The size of this structure is then $N \times (N \times 2) \times 26^2$.

$$K = k_1, k_2, ..., k_n \tag{4}$$

$$D = D_{k_1,k_2} + D_{k_2,k_3} + ... + D_{k_n,k_1'} \tag{5}$$

where $D_{k_n,k_1'}$ is the digram matrix of the last column and the following first column.

Scoring with transposition ciphers is just a matter of computing D as mentioned above and then running the scoring function (1). In order to extend this to simple substitution ciphers, it is necessary to find an alternative to swapping the rows and columns of the D matrix as described in the Jakobsen algorithm [13]. This can be done by swapping the columns and rows of the E matrix instead. Assume matrix E and D are both 4×4 matrices, where E is defined in 6 and D is defined in 7.

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{14} \\ e_{21} & e_{22} & e_{23} & e_{24} \\ e_{31} & e_{32} & e_{33} & e_{34} \\ e_{41} & e_{42} & e_{43} & e_{44} \end{bmatrix} \tag{6}$$

$$D = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{bmatrix} \tag{7}$$

If we swap columns and rows 1 and 3 in (7), we get the matrix 8.

$$D' = \begin{bmatrix} d_{33} & d_{32} & d_{31} & d_{34} \\ d_{23} & d_{22} & d_{21} & d_{24} \\ d_{13} & d_{12} & d_{11} & d_{14} \\ d_{43} & d_{42} & d_{41} & d_{44} \end{bmatrix} \tag{8}$$

The scoring equation would be

$$\begin{aligned} f(D', E) = {} & |d_{33} - e_{11}| + |d_{32} - e_{12}| + |d_{31} - e_{13}| + |d_{34} - e_{14}| \\ & + |d_{23} - e_{21}| + |d_{22} - e_{22}| + |d_{21} - e_{23}| + |d_{24} - e_{24}| \\ & + |d_{13} - e_{31}| + |d_{12} - e_{32}| + |d_{11} - e_{33}| + |d_{14} - e_{34}| \\ & + |d_{43} - e_{41}| + |d_{42} - e_{42}| + |d_{41} - e_{43}| + |d_{44} - e_{44}| \end{aligned}$$

Now if we did the same but with the (6) matrix instead, the updated matrix

would be matrix 9.

$$E' = \begin{bmatrix} e_{33} & e_{32} & e_{31} & e_{34} \\ e_{23} & e_{22} & e_{21} & e_{24} \\ e_{13} & e_{12} & e_{11} & e_{14} \\ e_{43} & e_{42} & e_{41} & e_{44} \end{bmatrix} \tag{9}$$

By scoring with this matrix and an unchanged D matrix, we get the following equation:

$$\begin{aligned} f(D, E') = &|d_{11} - e_{33}| + |d_{12} - e_{32}| + |d_{13} - e_{31}| + |d_{14} - e_{34}| \\ &+ |d_{21} - e_{23}| + |d_{22} - e_{22}| + |d_{23} - e_{21}| + |d_{24} - e_{24}| \\ &+ |d_{31} - e_{13}| + |d_{32} - e_{12}| + |d_{33} - e_{11}| + |d_{34} - e_{14}| \\ &+ |d_{41} - e_{43}| + |d_{42} - e_{42}| + |d_{43} - e_{41}| + |d_{44} - e_{44}| \end{aligned}$$

Comparison of the two equations shows that they are the same and performing the swap on the E or D matrices gives us the same result. So by using this new data structure, we are able to quickly determine the total D matrix from the column pair matrices and also perform the Jakobsen algorithm by manipulating the columns and rows of the E matrix.

Since the E matrix is being manipulated, it is necessary to perform the swaps on the inverse of the key. So in the original attack, the initial key is generated based on the letter frequencies of the ciphertext and any swap in the key means the column and rows of the D matrix are swapped. With the modified algorithm, this same initial key will be inverted and the swaps need to be performed on this new key, so at the end of the attack, the key found is actually the decryption key rather than the encryption key.

The new attack, shown in algorithm 10, will also maintain changes to the E matrix rather than the D matrix.

Algorithm 10 ModifiedJakobsenAlgorithm

```
1: Let a = 1 and b = 1.
2: Get initial key based on letter frequencies in the ciphertext.
3: Get decryption key invkey by inverting key.
4: Construct E matrix based on English language text encrypted with key.
5: Construct digram data structure based on description above.
6: Score the putative key with f(t) from equation (1).
7: while b ≠ 26 do
8:    if a + b ≤ 26 then
9:       Let tempkey = invkey.
10:      Let E′ = E
11:      Swap element a and b in tempkey.
12:      Swap rows and columns a and b in E′.
13:      Score with E′.
14:      if newscore < score then
15:         invkey = tempkey
16:         E = E′
17:         a = 1
18:         b = 1
19:      else
20:         tempkey = invkey
21:         E′ = E
22:         increment a
23:      end if
24:   else
25:      a = 1
26:      increment b
27:      tempkey = invkey
28:      E′ = E
29:   end if
30: end while
```

Since this updated attack still performs the same type of swap as the original algorithm, the work required is still $\binom{26}{2}$ in the best case scenario and $\binom{26}{2}!$ in the worst. Now that this new data structure has been defined, a fast hill climb attack against columnar transposition ciphers can be developed.

CHAPTER 4

Transposition Hill Climb Attack

Using the data structure presented in the previous chapter, it is now possible to quickly perform a hill climb attack on a transposition cipher. As previously mentioned, the existing attacks on transposition ciphers requires decryption of the ciphertext with each putative key and then calculation of the digram matrix. By using the data structure, all the possible digrams for any pair of columns is calculated and stored so for any putative key, and the total digram matrix can be determined based on these already stored values rather than decrypting the text again.

The attack uses a hill climb approach that is similar to the Jakobsen attack [13]. Initially, a random key needs to be determined. Since this key is a randomly generated key, it does not have the same performance as the simple substitution attack, which generated the key based on the letter frequencies in the ciphertext and provided a good starting point for the hill climb to start. This means that the attack must be run multiple times with different keys to get the best possible solution.

With an initial key generated, the attack proceeds to swap adjacent pairs, then pairs of distance 2, pairs of distance 3, and so on. With each key, the digram data structure is searched based on the current putative key and a D matrix is calculated to be used in the scoring phase against the E matrix using (5).

The scoring function (1) is the same as the one used by all of the algorithms discussed previously. If a better score is found, then this new key is saved and the algorithm is reset to start by swapping adjacent pairs. Algorithm 11 is run for m iterations and after each iteration, the best score is saved.

Algorithm 11 TranspositionHillClimb

```
1: Let a = 1 and b = 1.
2: Randomly generate initial key.
3: Construct E matrix based on English language text.
4: Construct digram data structure based on description above.
5: Construct D from data structure and key.
6: Score the putative key with f(t) from equation (1).
7: while b ≠ keysize do
8:    if a + b ≤ keysize + 1 then
9:       Let tempkey = key.
10:      Swap element a and b in tempkey.
11:      Construct D from data structure and tempkey.
12:      Score with D.
13:      if newscore < score then
14:         score = newscore
15:         key = tempkey
16:         a = 1
17:         b = 1
18:      else
19:         tempkey = key
20:         increment a
21:      end if
22:   else
23:      a = 1
24:      increment b
25:      tempkey = key
26:   end if
27: end while
```

The amount of work for this transposition attack is made up of the hill climb component and the number of iterations performed. For a transposition key of size n, the work factor is $\binom{n}{2}$. The total work factor for 50 iterations is then

$$50 \cdot \binom{n}{2}$$

In the worst case scenario, the work factor would be $50 \cdot \binom{n}{2}!$.

CHAPTER 5

Simple Substitution-Columnar Transposition Attack

In order to obtain more security from classic ciphers, simple substitution and columnar transposition ciphers can be combined to provide confusion and diffusion in the ciphertext. With the simple substitution-columnar transposition cipher, which will be referred to as the SSCT cipher in the rest of this paper, the key space size becomes

$$2^{88} * n!$$

where n is the number of columns in the transposition cipher. It is obvious that an exhaustive search would be impossible.

Now using the modified Jakobsen attack and the transposition hill climb attack, a nested hill climb attack can be performed to find the keys for a ciphertext encrypted using simple substitution and transposition ciphers. Algorithm 12 performs the transposition attack, while algorithm 13 performs the nested Jakobsen attack. For each swap performed in the transposition attack, the Jakobsen attack is performed to determine the best simple substitution key for the current putative transposition key.

For this combination attack, the work factor is made up of the transposition attack from the previous chapter and the modified Jakobsen attack from chapter 3.

$$50 \cdot \binom{n}{2} \cdot \binom{26}{2}$$

where n is the transposition key size. For the worst case, $50 \cdot \binom{n}{2}! \cdot \binom{26}{2}!$.

Algorithm 12 SSCT

```
1: Let a = 1 and b = 1.
2: Randomly generate initial transkey.
3: Construct E matrix based on English language text.
4: Construct digram data structure based on description above.
5: Construct D from data structure and transkey.
6: score = SSCTJakobsen(D, subkey).
7: while b ≠ keysize do
8:    if a + b ≤ keysize + 1 then
9:       Let temptranskey = transkey.
10:      Let E' = E
11:      Swap element a and b in temptranskey.
12:      Construct D from data structure and temptranskey.
13:      newscore = SSCTJakobsen(D, tempsubkey).
14:      if newscore < score then
15:         score = newscore
16:         transkey = temptranskey
17:         subkey = tempsubkey
18:         E = E'
19:         a = 1
20:         b = 1
21:      else
22:         tempkey = key
23:         E' = E
24:         increment a
25:      end if
26:   else
27:      a = 1
28:      increment b
29:      tempkey = key
30:      E' = E
31:   end if
32: end while
```

Using this attack as a base, the next step is to extend it to attack homophonic substitution-columnar transposition ciphers.

Algorithm 13 SSCTJakobsen

Input: D matrix

Let $a = 1$ and $b = 1$.
Get initial `key` based on letter frequencies in the ciphertext.
Get decryption key `invkey` by inverting `key`.
Construct E matrix based on English language text encrypted with `key`.
Construct digram data structure based on description above.
Score the putative key with $f(t)$ from equation 1.
while $b \neq 26$ **do**
  if $a + b \leq 26$ **then**
    Let `tempkey` = `invkey`.
    Let $E' = E$
    Swap element a and b in `tempkey`.
    Swap rows and columns a and b in E'.
    Score with E'.
    if `newscore` $<$ `score` **then**
      `invkey` = `tempkey`
      $E = E'$
      $a = 1$
      $b = 1$
    else
      `tempkey` = `invkey`
      $E' = E$
      increment a
    end if
  else
    $a = 1$
    increment b
    `tempkey` = `invkey`
    $E' = E$
  end if
end while

CHAPTER 6

Homophonic Subsitution-Columnar Transposition Attack

An extension of the previous cipher is to substitute simple substitution with the homophonic substitution cipher. By doing so, the key space is

$$2^{4.7n} \times \texttt{keysize}!$$

By using the concepts from the attack on the SSCT cipher, an algorithm can be created to perform in much the same way against the homophonic substitution-columnar transposition cipher, which will be referred to as HSCT in the rest of the paper. If the simple substitution aspect of the attack is substituted with the homophonic attack, the attack would take far too long to complete. The work factor for such an attack, where m is the tranposition cipher key size and n is the homophonic substitution cipher key size, would be

$$4000 \cdot \binom{m}{2}! \cdot \binom{26}{2} \cdot \binom{n}{2}$$

Some changes need to be made to the SSCT attack to make it more efficient for the HSCT attack. First, the nested hill climb is swapped so that the transposition algorithm runs within the homophonic attack. The modified homophonic substitution attack is shown below in algorithm 14, 15, and 16. To speed up the attack, the transposition attack is only run at certain points of the attack rather than every swap in the inner hill climb layer. This is limited to run only when there is no change in score for a given distance between pairs in the inner hill climb. If the score changes at any time during a run through of distance i, then the run is considered a success and the next distance can be proceeded to. However, if there is no change in

score, then the run is repeated using the transposition hill climb attack, as shown in algorithm 17. By doing so, the amount of work can be greatly decreased and make the attack more manageable and efficient. The transposition attack is also modified to not reset after a successful swap. This changes the work factor to $\binom{m}{2}$ in the worst case scenario.

Due to how the algorithm is structured, the use of row and column swaps have been removed. The nature of homophonic keys makes it so that any number of ciphertexts can come from the same plaintext but any of these ciphertexts will always get back the same plaintext. What this means for the algorithm is that if the same method used in the SSCT attack is used here by performing swaps on the E matrix, it is not possible to know what the E matrix is supposed to encrypt to since there are multiple possibilities. So the D matrix is calculated each time a key must be scored. However, since the ciphertext does not need to be decrypted in order to determine this due to the data structure, the I/O operation is removed.

Once a run through the entire attack is complete, the homophonic substitution attack is performed alone without the transposition attack and then transposition attack is run alone without the homophonic attack. When the homophonic substitution attack is run alone, the best transposition key is used for the scoring process. The same process is used for the transposition attack, where the best homophonic substitution key is used in the scoring process.

The work factor for this entire attack is composed of the nested hill climb as well as the additional attacks performed afterwards. In the worst case, the nested hill climb would run for every distance of pairs.

$$4000 \cdot \binom{m}{2} \cdot \binom{26}{2} \cdot \binom{n}{2} + 50 \cdot \binom{n}{2} + 80 \cdot \binom{26}{2} \cdot \binom{n}{2} \approx 4000 \cdot \binom{m}{2} \cdot \binom{26}{2} \cdot \binom{n}{2}$$

Algorithm 14 HSCTOuterHillClimb

global `hsK` = `bestInithsKey` = `besthsKey` = `NULL`.
global `transK` = `bestInittransKey` = `besttransKey` = `NULL`.
parse ciphertext to determine data structure.
Construct E matrix based on English language text.
initialize $n_a, n_b, .., n_z$ as in Table 6.
$(m_1, m_2, ..., m_{26}) = (n_a, n_b, ..., n_z)$
`bestScore` = `RandomKeyLayer`$(m_1, m_2, ..., m_{26})$
`besthsKey` = `bestInithsKey`
`besttransKey` = `bestInittransKey`
for $i = 1$ to 25 **do**
  for $j = 1$ to $26 - i$ **do**
    $(m'_1, m'_2, ..., m'_{26}) = (m_1, m_2, ..., m_{26})$
    `OuterSwap`(m'_j, m'_{j+i})
    `score` = `RandomKeyLayer`$(m'_1, m'_2, ..., m'_{26})$
    if `score` $<$ `bestScore` **then**
      $(m_1, m_2, ..., m_{26}) = (m'_1, m'_2, ..., m'_{26})$
      `bestScore` = `score`
      `besthsKey` = `bestInithsKey`
      `besttransKey` = `bestInittransKey`
    else
      $(m'_1, m'_2, ..., m'_{26}) = (m_1, m_2, ..., m_{26})$
      `OuterSwap`(m'_{j+i}, m'_j)
      `score` = `RandomKeyLayer`$(m'_1, m'_2, ..., m'_{26})$
      if `score` $<$ `bestScore` **then**
        $(m_1, m_2, ..., m_{26}) = (m'_1, m'_2, ..., m'_{26})$
        `bestScore` = `score`
        `besthsKey` = `bestInithsKey`
        `besttransKey` = `bestInittransKey`
      end if
    end if
  end for
end for

Algorithm 15 HSCTRandomKeyLayer

Input: $n_a, n_b, ..., nz$

1: `bestInitScore` $= \infty$.
2: **for** $r = 1$ to R **do**
3: randomly initialize $K = (k_1, k_2, ..., kn)$ satisfying $n_a, n_b, ..., nz$.
4: `initScore` $=$ `InnerHillClimb`(K).
5: **if** `initScore` $<$ `bestScore` **then**
6: $(m_1, m_2, ..., m_{26}) = (m'_1, m'_2, ..., m'_{26})$
7: `bestInitScore` $=$ `initScore`
8: `bestInithsKey` $=$ `hsK`
9: `besttransKey` $=$ `transKey`
10: **end if**
11: **end for**

Algorithm 16 HSCTInnerHillClimb

Input: key K

```
1: innerScore = f(t).
2: for i = 1 to n − 1 do
3:    for j = 1 to n − i do
4:       K′ = K
5:       swap(k′_j, k′_{j+i})
6:       if no change in score for current i then
7:          K′ is previous best key.
8:          score = HSCTTransposition(K′)
9:          if score < innerScore then
10:            innerScore = score
11:            K = K′
12:         end if
13:      else if no change a second time then
14:         for k = 1 to iterations do
15:            K′ is a newly randomized key
16:            score = HSCTTransposition(K′)
17:            if score < innerScore then
18:               innerScore = score
19:               K = K′
20:            end if
21:         end for
22:      else
23:         score = ft
24:         if score < innerScore then
25:            innerScore = score
26:            K = K′
27:         end if
28:      end if
29:   end for
30: end for
```

Algorithm 17 HSCTTransposition

```
1: Let a = 1 and b = 1.
2: Randomly generate initial transkey.
3: Construct D from data structure and transkey.
4: score = f(t).
5: while b ≠ keysize do
6:    if a + b ≤ keysize + 1 then
7:       Let temptranskey = transkey.
8:       Swap element a and b in temptranskey.
9:       Construct D from data structure and temptranskey.
10:      newscore = f(t).
11:      if newscore < score then
12:         score = newscore
13:         transkey = temptranskey
14:      else
15:         tempkey = key
16:         increment a
17:      end if
18:   else
19:      increment b
20:      tempkey = key
21:   end if
22: end while
```

CHAPTER 7

Results

The modified Jakobsen algorithm, transposition hill climb attack, the combined simple substitution-columnar transposition, and finally, the homophonic substitution-columnar transposition algorithms were all tested. They were all written in C in a UNIX-like environment. All of the tests were performed on a 3.4GHz Intel i7 machine with 16GB of RAM.

7.1 Modified Jakobsen Algorithm

This was tested to ensure that the modifications did not negatively affect the results. It was tested over multiple ciphertexts of lengths 100 to 1000 at differences of 100 and 2000 to 10000 at differences of 1000. Based on the percentage of the plaintext recovered, the results are just as good as with the original algorithm, as can be seen in figure 2. This confirmed that the data structure is suitable for this attack.

7.2 Transposition Hill Climb Attack

The tests run on the transposition attack was run over transposition key lengths of 10 to 20 with the same ciphertext lengths used in the previous section. It was also run using 50 and 100 iterations. Scoring of the key is based on groups of columns. Due to the nature of the transposition cipher, if the correct columns are grouped together, then text will be readable as long as there are enough columns within each group. It is just a matter of rearranging these blocks to get the correct key. The percentage of the text recovered is based on this concept. As long as the number of columns is greater than 2, then the text is considered recovered.

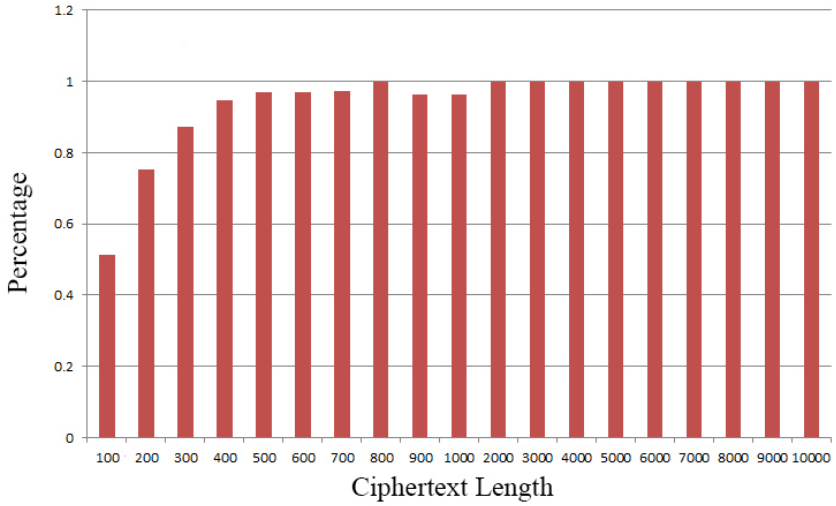
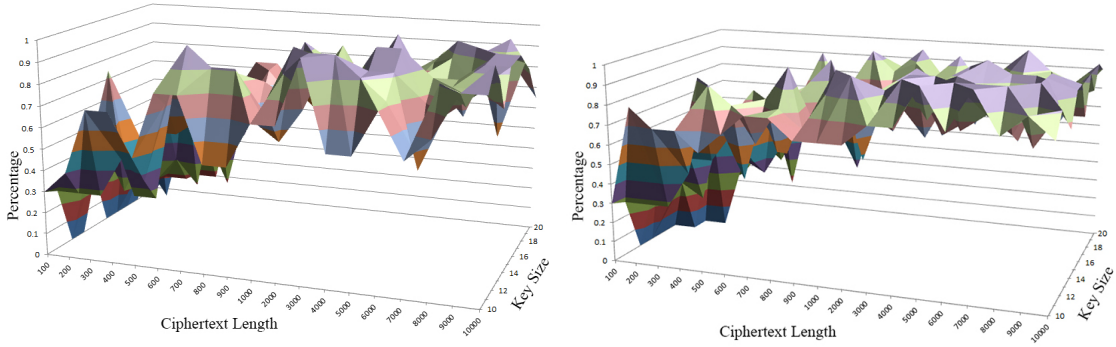


Figure 2: Modified Jakobsen Results

.

Based on the amount of the key recovered, using 100 iterations, figure 3b, produced similar results to using 50 iterations, figure 3a, with only a slight improvement. So based on these results, for the combined attack, only 50 iterations are needed. As can be seen in the graphs, the longer the transposition key to be recovered, the less success there is. Based on testing, with transposition, having at least 60% of the text decrypted generates readable text that can be manually decrypted afterwards.



(a) 50 Iterations Results

(b) 100 Iterations Results

Figure 3: Transposition Hill Climb Results

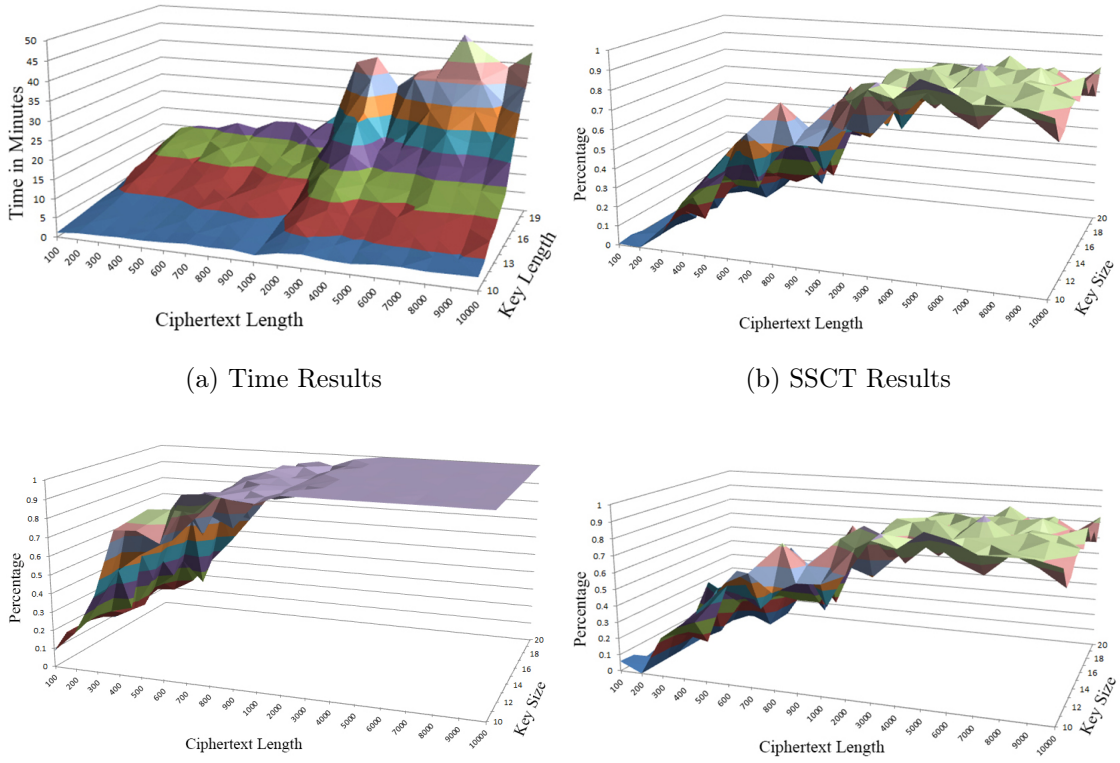
7.3 Simple Substitution-Columnar Transposition

For the SSCT attack, the tests are very similar to the tests done on the transposition attack. In this case, the same amounts of ciphertext were tested along with the same key lengths. The only part added is the fact that the ciphertext is also encrypted with a simple substitution key. The transposition part of the attack is set to 50 iterations based on the results obtained in the previous set of tests. In addition the time to complete was measured to check how much work is involved to execute the attack.

Based on the time results, figure 4a, for short transposition keys, the algorithm can be completed in a few minutes. However, as this increases, the time to complete also sharply increases.

In addition to the time results, the percentage of the text recovered is shown in figure 4b. The percentage is calculated by determining the blocks of columns that are recovered using the same method as with the transposition attack and then the plaintext recovered from these columns. To further break down the results to determine the effectiveness of the attack, the scores for the individual transposition keys in figure 4d and simple substitution keys in figure 4c are examined.

With about 600 to 700 characters of ciphertext, simple substitution keys that decrypt 80% of the ciphertext can be found. And with the same amount of ciphertext, 60% of the text can be decrypted with the transposition keys found. Based on these results, at least 600 or 700 characters of ciphertext is needed to get at least readable text and to be able to recover the remainder of both keys.



(a) Time Results

(b) SSCT Results

(c) Simple Substitution Results

(d) Transposition Results

Figure 4: SSCT Results

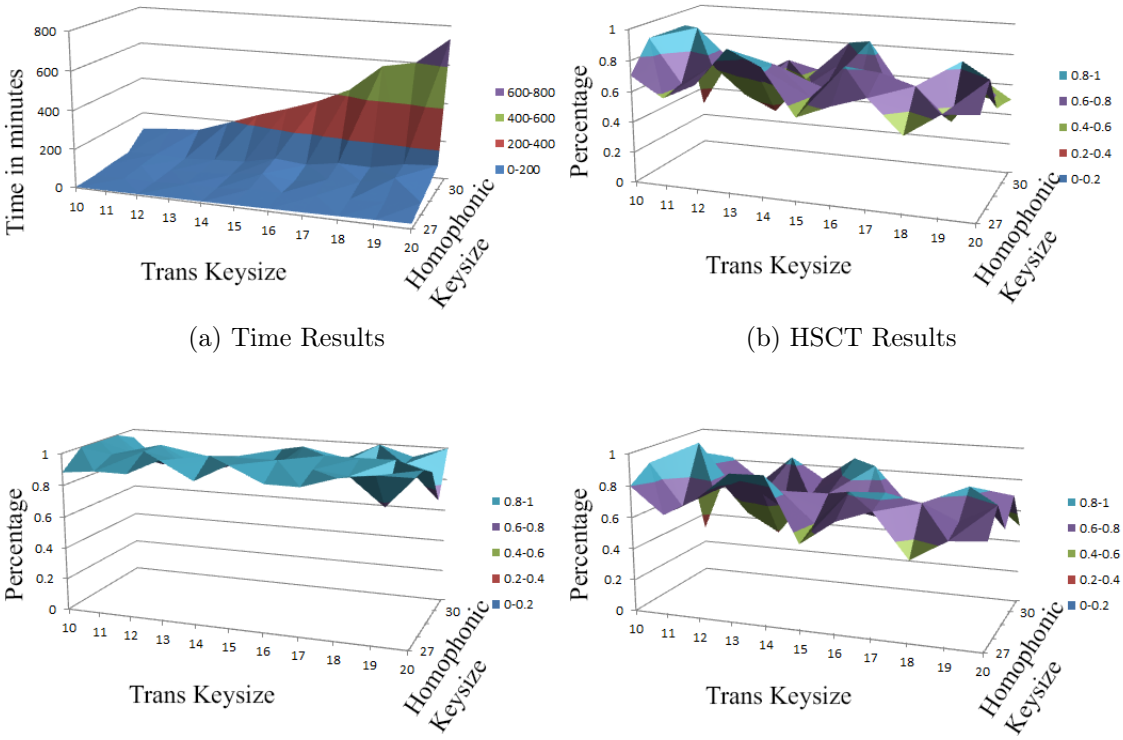
7.4 Homophonic Substitution-Columnar Transposition

The tests performed with the HSCT attack used transposition key sizes of 10 to 20 and homophonic key sizes of 27 to 35. In addition to this, like the SSCT tests, the time was also recorded. The time to complete increases tremendously as the key sizes increase for a ciphertext of length 10000, as shown in figure 5a. So for larger keys, such as the ones used for the Zodiac-340 cipher, the time to complete can take at least a week.

Just as with the SSCT tests, the percentage of the text recovered is scored based on the blocks of columns grouped together and the amount of text that is correctly substituted within these blocks. The results for a ciphertext of 10000 is shown in

figure 5b. In this case, for transposition keys of greater than 18 and homophonic keys greater than 30, the scores dip below 60%, meaning that the text is not readable enough to manual recover the remainder of the keys.

By breaking down the amount of plaintext recovered with the individual keys, the homophonic key is very close to 100%. However, the transposition key only recovers about 60% of the text with transposition keys less than 15 and homophonic key less than 30, which falls in line with the results seen in the combined results.



(a) Time Results

(b) HSCT Results

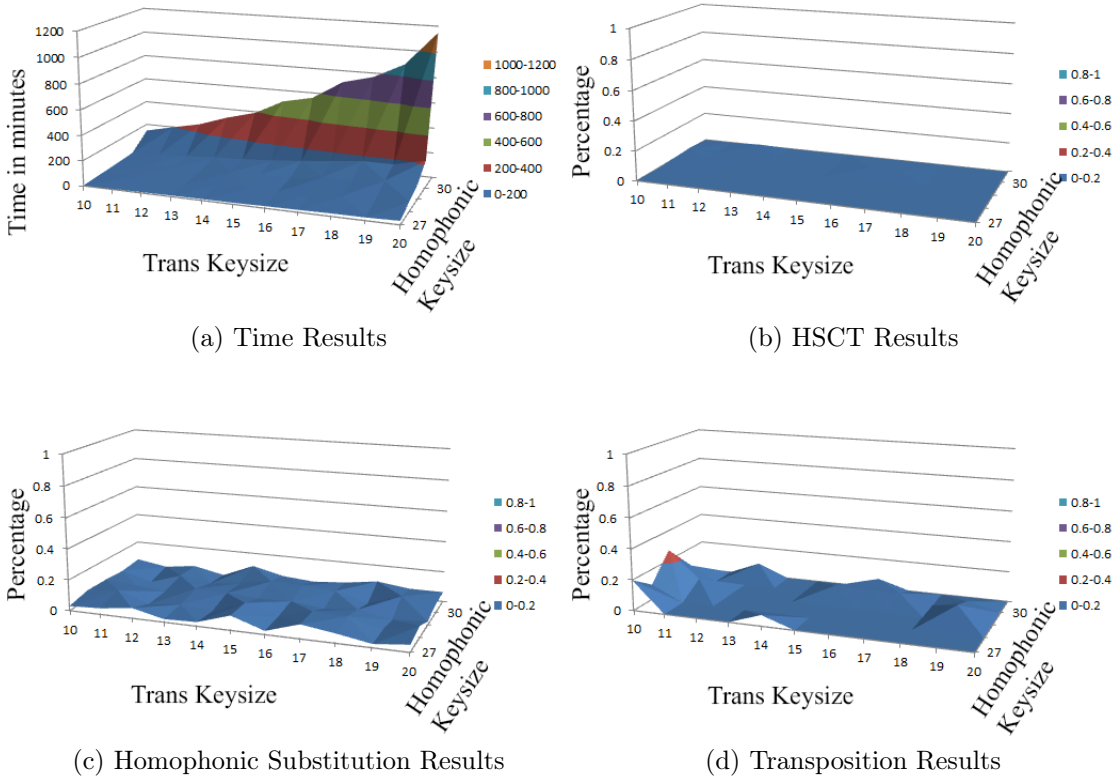
(c) Homophonic Substitution Results

(d) Transposition Results

Figure 5: HSCT Results - 10000 Ciphertext Length

Based on the results from using 100 and 1000 ciphertext lengths, figures 6 and 7 respectively, no readable text is recovered. In the 1000 length case, at the very smallest keys, some text may be recovered but the minimum needed for each key

is not achieved. Based on the combined results, the text would still be unreadable. More testing needs to be done to determine the threshold ciphertext length to be able to recover the homophonic substitution and transposition keys.



(a) Time Results

(b) HSCT Results

(c) Homophonic Substitution Results

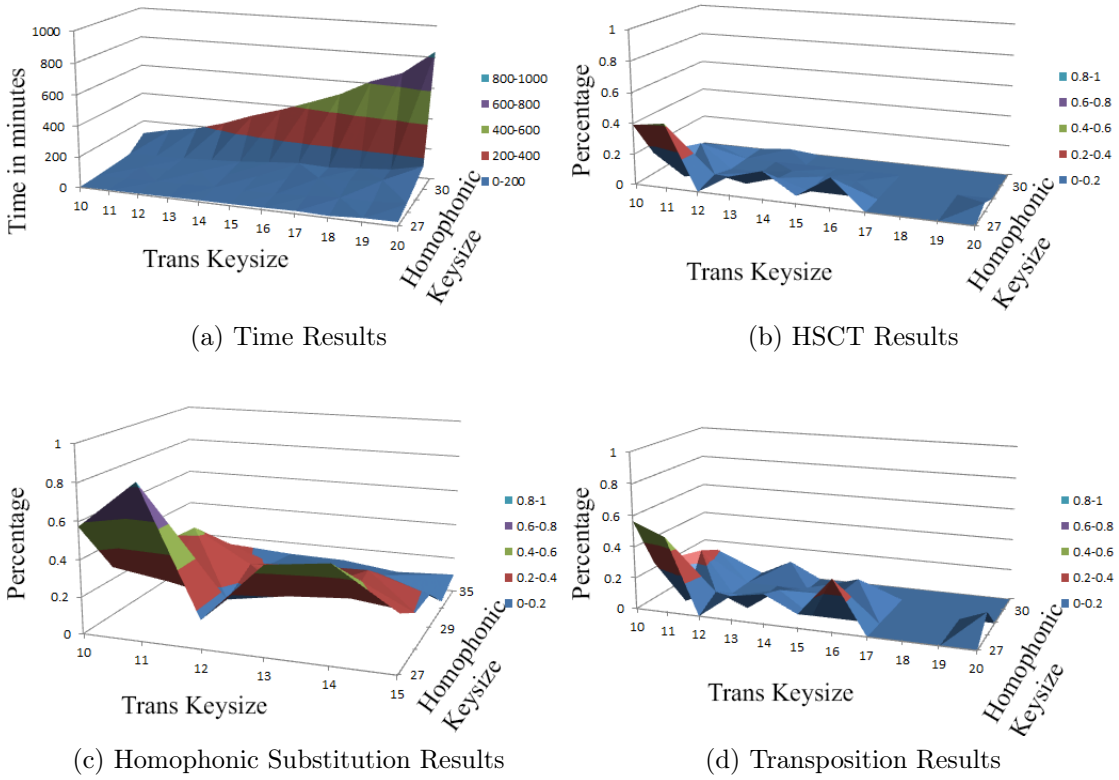
(d) Transposition Results

Figure 6: HSCT Results - 100 Ciphertext Length

7.5 Zodiac-340 Cipher

The Zodiac-340 cipher is an unsolved cipher made up of 62 distinct symbols and organized in rows of 17. Using these as the key sizes for the HSCT attack, the attack is run against the cipher to see if can retrieve any readable text. The ciphertext is shown in Table 8.

The HSCT attack on the cipher took 8.7 days to complete. The homophonic



(a) Time Results

(b) HSCT Results

(c) Homophonic Substitution Results

(d) Transposition Results

Figure 7: HSCT Results - 1000 Ciphertext Length

substitution decryption key found is

`YTAAKCMSIASTQLDAEAKHNETEXNTHGSVIBUNJITDPHONOEIRVORLREFEOEZPJWX.`

The transposition key is

$$(13, 10, 1, 15, 2, 16, 11, 8, 12, 5, 9, 14, 7, 3, 4, 6, 0).$$

With these keys, the ciphertext decrypts to the plaintext in Table 9.

This decryption does not provide at any readable text and the time to perform the attack makes impractical for multiple runs to try to find a better solution. Improvements need to be made to both lower the time it takes to complete the attack and improve the results for lower lengths of ciphertext. Some ideas to do this will be

Table 8: Zodiac-340 Cipher

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17
18	05	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
20	34	35	36	37	19	38	39	15	26	21	33	13	22	40	01	41
42	05	05	43	07	06	44	30	08	45	05	23	19	19	03	31	16
46	47	37	19	40	48	49	17	11	50	51	09	19	09	52	10	53
05	44	03	07	51	06	23	54	30	17	55	10	51	04	16	25	21
22	50	19	31	56	24	57	16	38	36	58	15	08	28	40	13	11
21	15	16	41	32	49	22	23	19	46	18	27	40	19	59	13	47
17	29	37	19	60	19	39	03	16	51	20	36	34	61	62	52	31
54	40	06	38	08	19	07	41	19	23	05	43	29	51	20	34	54
38	19	03	53	50	48	02	11	25	27	20	05	60	14	37	31	23
16	29	36	06	03	41	11	30	50	14	50	37	28	19	09	20	51
40	62	47	42	34	22	19	18	11	50	51	20	36	21	57	44	03
06	15	51	18	07	32	50	16	50	60	28	36	08	50	48	19	19
34	20	58	12	30	35	52	47	55	02	04	08	38	39	50	54	19
11	36	28	45	40	20	31	21	23	05	07	28	32	37	56	15	16
03	36	14	19	13	50	16	55	29	19	51	06	26	20	11	33	13
19	19	33	26	55	40	26	36	09	23	42	01	14	53	21	33	05
11	51	10	17	26	29	43	48	20	46	27	23	20	30	54	55	36
04	37	25	01	18	05	10	42	40	39	23	44	61	11	31	57	19

presented in Chapter 8.

Table 9: Zodiac-340 Putative Plaintext

E	A	K	L	D	A	A	Q	S	S	T	M	I	Y	T	A	C
B	K	N	S	V	N	I	G	E	T	K	T	X	A	H	H	E
H	N	I	E	P	N	Y	Q	D	N	U	T	D	H	B	J	K
A	K	M	K	A	E	V	K	S	K	K	O	S	O	T	N	C
E	I	P	I	R	R	A	K	E	L	R	O	S	I	I	K	V
N	A	L	A	A	E	X	L	F	E	O	T	S	K	A	M	C
S	K	O	H	P	J	Q	S	A	Z	R	E	T	E	D	V	E
R	A	I	K	P	I	Q	P	T	A	D	E	K	N	T	H	O
V	I	J	W	X	L	R	U	A	H	G	D	A	E	J	K	K
F	C	S	L	H	T	U	G	H	K	P	M	K	F	N	T	K
T	A	R	L	I	T	V	J	S	H	K	T	X	T	K	E	V
L	J	A	K	I	L	H	H	S	R	G	S	R	A	I	C	H
A	R	U	N	E	R	O	J	A	L	X	K	S	P	H	O	E
K	L	M	R	V	J	K	S	A	H	D	R	R	C	J	A	I
K	Z	S	D	R	T	F	T	R	A	H	R	E	U	S	T	N
A	H	P	I	O	K	D	I	N	M	J	V	T	S	H	E	H
Q	L	Q	H	S	K	B	N	E	L	J	A	G	A	C	K	R
K	B	E	E	N	T	B	L	J	O	K	N	I	K	Y	N	P
J	A	N	S	F	I	E	H	V	T	L	N	H	S	T	E	G
K	X	A	S	V	D	E	W	O	T	I	A	P	A	O	Y	K

CHAPTER 8

Conclusion

By utilizing two different types of ciphers, ciphertext can gain the attributes of confusion and diffusion. Using a homophonic substitution cipher along with a transposition cipher, it becomes possible to flatten the letter frequencies in the ciphertext as well as permute the characters to mix up the digram frequencies. The goal of this paper is to find a way to crack this type of encryption.

The idea for this attack came from the as of yet unsolved Zodiac-340 cipher. While it has not been completely ruled out as solely a homophonic substitution cipher, looking for another possibility for the cipher is the next logical step. And since it is most likely a homophonic cipher due to the use of over 26 symbols, it must be an extension of this encryption method rather than a completely new cipher.

This was done by first looking at the simpler combination of simple substitution and transposition ciphers. With this obtaining readable text, it was possible to replace simple subsitution with homophonic substitution, albeit with modifications to speed up the algorithm.

Based on the results of the HSCT attack, the amount of ciphertext required is much higher than desired to retrieve the keys. Due to this limitation, performing the attack against the Zodiac-340 cipher results in no readable text. And in addition to this, due to the large amount of symbols used in the cipher results in a very long attack time. While the SSCT attack did provide decent results, the complexity of the homophonic attack causes the attack to take many times longer than the SSCT attack.

While this may not have achieved results that can be applied to lower amounts of ciphertext, future work on the algorithm could improve the speed and effectiveness of the attack. One way may be to increase the number of iterations performed by both the random key layer and the transposition layer. By increasing the number of iterations, better starting points may be obtained. In conjunction with this, the algorithm can take advantage of parallel computing by performing the same two layers in parallel since each key is independent from one another. Previous work was done using the EM algorithm to perform up to a million random starts in a parallel environment against the homophonic substitution cipher [4]. This paper found that using a large number of restarts was able to find the homophonic substitution key for the Zodiac-408 cipher. Using this idea, the algorithm in this paper could also use a million random keys for both the transposition attack and the homophonic attack and utilize a parallel GPU environment to have a better chance of finding the best solution.

LIST OF REFERENCES

[1] Voigt, T. (2007). *340-Symbol Cipher*.
http://www.zodiackiller.com/340Cipher.html

[2] A 'Murder Code' Broken. (1969, August 9). *San Francisco Chronicle*.
http://www.flickr.com/photos/seanutbutter/2462841231/

[3] Basavaraju, P. (2009). Heuristic-search cryptanalysis of the Zodiac 340 cipher, *Master's Projects.*, Paper 56.
http://scholarworks.sjsu.edu/etd_projects/56/

[4] Berg-Kirkpatrick, T., Klein, D. (2013). Decipherment with a Million Random Restarts. *Proceedings of EMNLP 2013*.

[5] Carroll, J. M., Martin, S. (1986). The automated cryptanalysis of substitution ciphers. *Cryptologia, 10*(4), 193–209.

[6] Clark, A. (1994, December). Modern optimisation algorithms for cryptanalysis. *In Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, 258–262.

[7] Dao, T. (2008). Analysis of the Zodiac 340-cipher. *Master's Projects.*, Paper 3570.
http://scholarworks.sjsu.edu/etd_theses/3570/

[8] Dhavare, A., Low, M., Stamp, M. (2011). Efficient Cryptanalysis of Homophonic Substitution Ciphers. *Master's Projects*.
http://www.cs.sjsu.edu/faculty/stamp/RUA/homophonic.pdf

[9] Dimovski, A., Gligoroski, D. (2003). Attacks on the transposition ciphers using optimization heuristics. *Proceedings of ICEST*, 1–4.

[10] Garg, P (2009). Genetic algorithms, tabu search and simulated annealing: A comparison between three approaches for the cryptanalysis of transposition cipher. *Theoretical and Applied Information Technology, 15*(4), 387–392.

[11] Giddy, J. P., Safavi-Naini, R. (1994). Automated cryptanalysis of transposition ciphers. *The Computer Journal, 37*(5), 429–436.

[12] Graham-Cumming, J. (2011, June 14). How The Zodiac enciphered the Zodiac 408 cipher.
http://blog.jgc.org/2011/06/how-zodiac-enciphered-zodiac-408-cipher.html

[13] Jakobsen, T. (1995). A fast method for the cryptanalysis of substitution ciphers, *Cryptologia, Vol. 19*, 265–274.

[14] Lucks, M. (1990, February). A constraint satisfaction algorithm for the automated decryption of simple substitution ciphers. *In Proceedings on Advances in cryptology*, 132–144.

[15] Oranchak, D. (2008). Evolutionary algorithm for decryption of monoalphabetic homophonic substitution ciphers encoded as constraint satisfaction problems. *In Proceedings of the 10th annual conference on Genetic and evolutionary computation* (GECCO '08), Maarten Keijzer (Ed.). ACM, New York, NY, USA, 1717–1718.

[16] Ravi, S., Knight, K. (2011). Bayesian inference for Zodiac and other homophonic ciphers. *In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1* (HLT '11), Vol. 1. Association for Computational Linguistics, Stroudsburg, PA, USA, 239–247.

[17] Russell, M. D., Clark, J. A., Stepney, S. (2003, December). Making the most of two heuristics: Breaking transposition ciphers with ants. *In Evolutionary Computation, 2003, Vol. 4* (CEC'03), 2653–2658.

[18] Spillman, R., Janssen, M., Nelson, B., Kepner, M. (1993). Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia, 17*(1), 31–44.

[19] Stamp, M., Low, R. M. (2007). *Applied cryptanalysis: Breaking ciphers in the real world*. Hoboken, N.J: Wiley-Interscience.

[20] Toemeh, R., Arumugam, S. (2007). Breaking Transposition Cipher with Genetic Algorithm. *Electronics and Electrical Engineering*, (79), 75.

[21] Uddin, M. F., Youssef, A. M. (2006, July). Cryptanalysis of simple substitution ciphers using particle swarm optimization. *In Evolutionary Computation, 2006* (CEC 2006), 677–680.