CRYPTANALYSIS OF THE SIGABA CIPHER

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Heather Ellie Kwong

December 2008

SAN JOSE STATE UNIVERSITY

The Undersigned Thesis Committee Approves the Thesis Titled

CRYPTANALYSIS OF THE SIGABA CIPHER

by

Heather Ellie Kwong

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Mark Stamp, Department of Computer Science   Date

Dr. Sami Khuri, Department of Computer Science   Date

Dr. Robert Chun, Department of Computer Science   Date

APPROVED FOR THE UNIVERSITY

Associate Dean   Date

ABSTRACT

CRYPTANALYSIS OF THE SIGABA CIPHER

By Heather Ellie Kwong

SIGABA is a rotor-based cipher machine that is famous for its service during World War II by the United States. Compared to other ciphers used in World War II, such as the German Enigma or the Japanese Purple, SIGABA's security was undefeatable, as it was the only cipher to withstand all cryptanalytic attacks in the course of its usage.

This thesis covers the history of SIGABA's development, how SIGABA works, and a cryptanalytic attack on SIGABA. The attack covered in this thesis has never been implemented before and is divided into a primary phase and secondary phase. The attack recovers SIGABA's keyspace by targeting SIGABA's rotor banks separately, while demonstrating SIGABA's strength in design that separates it from other ciphers.

ACKNOWLEDGEMENTS

To my parents who taught me the value of an education and raised me to be the person I am today.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

1.1. Introduction to Computer Security

As technology continues to thrive and grow, so does computer hacking. Hackers are drawn to expose security gaps for a variety of reasons: greed, ill intent, a hobby, or gaining the bragging rights to have defeated a system. Though this can be seen as unfortunate, there is a silver lining. In order to avoid computer security breaches, stronger code is developed with the following security goals and protocol in mind: confidentiality, integrity, and authentication.

An important security goal is confidentiality, which prevents any unauthorized reading of information so that only those who are authorized can access the data. Integrity, the second security goal, prevents any unauthorized writing of information to protect the data's validity. Authentication protocols are used to confirm a person's identity to protect a computer system or resource [8]. Technology would not be where it is today without these three cornerstones, and these three cornerstones would not be possible without cryptography.

1.2. Introduction to Cryptography Terminology

Cryptography is the study and practice of encryption, the hiding of information. The root of the word is derived from the Greek words *krypto* and *grafo*, which mean "hidden" and "to write," respectively. Today, the word cryptography holds a double meaning that pertains to the study of mathematics and computer science [12].

Encryption involves converting original data, called plaintext, to something incomprehensible, called ciphertext. The reverse process of encryption is decryption, the conversion of ciphertext back to its original plaintext.

A cipher is a machine that performs both encryption and decryption. Each cipher contains a specific algorithm for both encryption and decryption based on a secret key. Without the same key that was used to encrypt the plaintext in symmetric ciphers, the ciphertext cannot decrypt to its original state [9].

The analysis and breaking of ciphers is called cryptanalysis. The term cryptology refers to both cryptography and cryptanalysis [9].

1.3. Project Overview

Cipher systems can be divided into two groups: classic and modern. This thesis will focus on the classic cipher SIGABA, which is a symmetric cipher. We will discuss SIGABA's history, its internal hardware design, and a modern cryptanalytic attack on SIGABA that is developed in *SIGABA: Cryptanalysis of the Full Keyspace* [10], which I implement for the first time. We will refer to this paper as *Cryptologia* from this point on.

2. Rotor Machines

2.1. Simple Substitutions

2.1.1. Simple Substitution Definition

A simple substitution is a fixed one-to-one mapping of a plaintext letter to one ciphertext letter. Consider Table 1, which contains a key for a simple substitution cipher.

Table 1. An example key for a simple substitution.

Plaintext	A	B	C	D	E	F	G	H	U	J	K	L	M
Ciphertext	i	r	d	e	j	u	a	n	f	c	q	t	o

Plaintext	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext	h	m	w	v	g	k	b	x	s	y	l	z	p

Using this key, the plaintext letter *a* maps to *i*, *t* to *b*, and so on. Therefore, the plaintext message

AttackAtDawn

will always encrypt to

ibbidqibeiyh.

2.1.2. The Cryptanalysis of Simple Substitutions

Simple substitutions are inherently weak ciphers, because their keys can easily be deduced by studying the frequency of letters for any intercepted ciphertext. This study of

3

frequencies is known as frequency analysis. Figure 1, reprinted from Wikipedia, illustrates a frequency analysis for the English language. Notice that the letter *e* is most common and *z* is the least common, which explains why contestants must pay extra for an *e* in Wheel of Fortune.
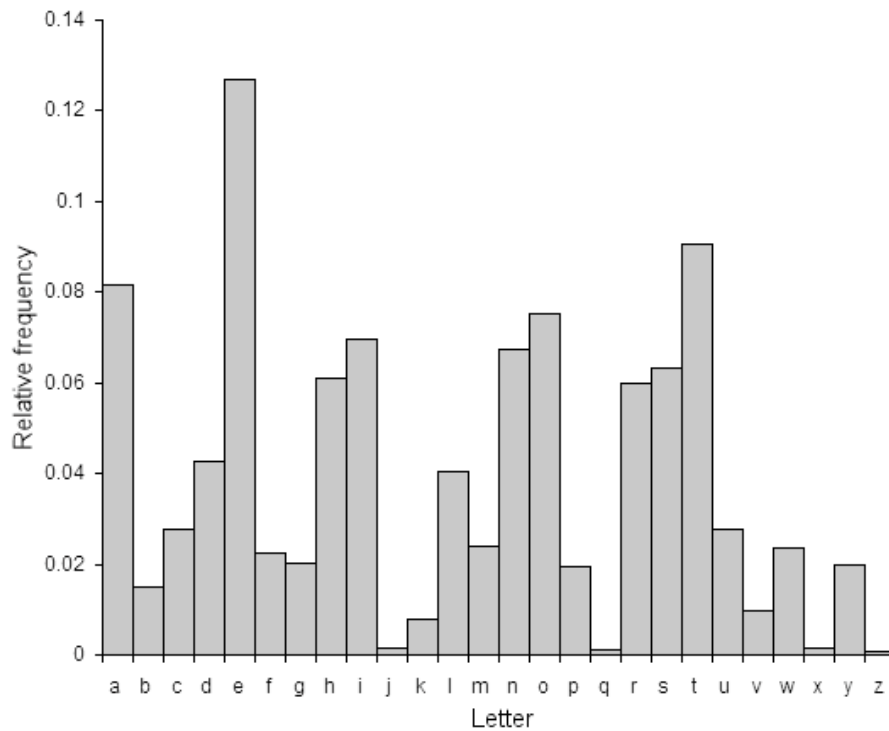
Figure 1. A typical distribution of English letters.

Frequency analysis can also be applied to groups of letters, known as digrams (groups of two letters) and trigraphs (groups of three letters). Table 2 lists the most common digrams in the English language and the expected number of occurrences for each given 2,000 letters [13]. The most common trigrams of the English language are listed in Table 3 [8].

Table 2. Digrams for the English language per 2,000 letters.

TH	50	AT	25	ST	20
ER	40	EN	25	IO	18
ON	39	ES	25	LE	18
AN	38	OF	25	IS	17
RE	36	OR	25	OU	17
HE	33	NT	24	AR	16
IN	31	EA	22	AS	16
ED	30	TI	22	DE	16
ND	30	TO	22	RT	16
HA	26	IT	20	VE	16

Table 3. Common trigams for the English language.

the	for	res
and	tio	ter
tha	has	con
hat	edt	ing
ent	tis	men
ion	ers	tho

Consequently, recovering the key to a simple substitution is not an impossible or even a difficult task.

2.2. Poly-Alphabetic Substitutions

A poly-alphabetic substitution is similar to a simple substitution only multiple substitution alphabets are used. The first cipher to use a poly-alphabetic substitution was invented by Leon Battista Alberti (1404 – 1472). Alberti's cipher consisted of two cipher wheels, each with the alphabet printed along the perimeter. The two wheels were positioned so that one was inside the other [4].   Figure 2 illustrates an Alberti cipher reprinted from Wikipedia [12].

Figure 2. An Alberti cipher.

The inner wheel was allowed to rotate per input so that multiple alignments were created between the two wheels, and with each new alignment, a new simple substitution was generated [4].

A more famous poly-alphabetic cipher is the aVigenère cipher. The aVigenère cipher is a classic poly-alphabetic substitution cipher, whereas the World War II ciphers

are more recent poly-alphabetic substitution ciphers. In the aVigenère cipher, the key used to encrypt and decrypt messages is of the form K = $(k_0, k_1, \ldots, k_{n-1})$, where each $k_i \in \{0, 1, \ldots, 25\}$ and represents a particular shift in the alphabet. Equation (1) encrypts and Equation (2) decrypts a letter using the aVigenère cipher [4].

$$c_i = p_i + k_i \ (\text{mod } n) \ (\text{mod } 26) \tag{1}$$

$$p_i = c_i - k_i \ (\text{mod } n) \ (\text{mod } 26) \tag{2}$$

where

c_i is the ith ciphertext letter

p_i is the ith plaintext letter

Another example of a poly-alphabetic substitution is a cipher with multiple wheels that have an odometer effect. The most famous cipher known to follow an odometer stepping pattern is the Enigma cipher that was used by the Germans in World War II [4]. By using an odometer stepping pattern, the rightmost wheel, which we will call F (for fast), advances one step for every input. The wheel directly to the left, M (for medium), will advance one step for every complete rotation F makes. Likewise, the leftmost wheel, S (for slow), will advance one step once M makes a complete rotation. The order in which the wheels F, S, and M are placed is illustrated in Figure 3.
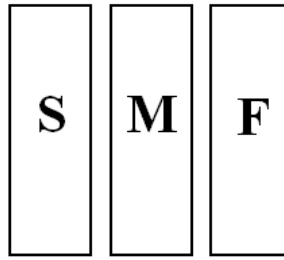
Figure 3. Wheels for an odometer setup.

In this example, there are only three wheels. However, this stepping pattern is not limited to three wheels and may be repeated for as many additional wheels that follow to the left. Although ciphers stepping to the pattern of an odometer are significantly stronger compared to a simple substitution cipher, it is still weak because the steps are predictable. This ultimately led to Enigma's downfall [4]. If Enigma's designers had designated either the leftmost or middle wheel to be the fast wheel, replicating an almost odometer-like pattern, Enigma's steps would have been irregular. Consequently, it would have been much more difficult to break Enigma [4]. This almost odometer pattern is used in SIGABA [4].

2.3. Introduction to Rotor-Based Machines

SIGABA is a rotor-based cipher, a type of electro-mechanical machine [4]. Before World War II, developing rotor-based machines received little attention within the United States. However, interest in cipher machines increased drastically during World War II when the need to develop secure methods of communication involving relatively large amounts of data became a priority. While building secure ciphers was considered important, scientists devoted far more time and research to breaking ciphers belonging to the Axis Powers, many of which were rotor cipher machines. The popularity of rotor-based ciphers lasted into the 1950s.

As indicated by its name, the main components of a rotor-based cipher are its rotors. A rotor is essentially a wired wheel. Each rotor is typically the shape and size of a hockey puck. Along the perimeter of each rotor are a variable amount of evenly spaced electrical contacts. This small array of electrical contacts lies on either side of the rotor and functions in a simple substitution of letters. Figure 4, reprinted from Wikipedia [16], illustrates what a typical rotor looks like.

Figure 4. A rotor.

Rotors can represent letters or numbers. If the rotor represents the English alphabet, the rotor will have 26 contacts — one contact for each letter of the alphabet. When there are 26 contacts, there are 26 different electrical signals, where each signal is mapped to a specific contact on the rotor. When a signal is passed from one face to the opposite face of the rotor, it is permuted into a different character. In Figure 5, reprinted from Hellman [3], an input letter of R is permuted to output letter B. A bank of rotors is a set of rotors that are connected via the electrical contacts that are shown in Figure 6, reprinted from Hellman [3].
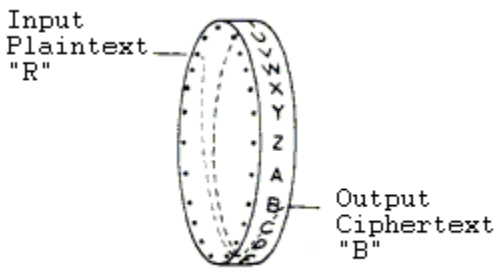
Figure 5. A rotor for the English Alphabet.
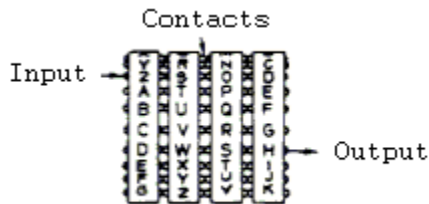
Figure 6. A bank of rotors.

When rotors are connected in such a manner, an electrical input entering one rotor continues to be permuted by all subsequent rotors. However, without applying any stepping motion to any of the rotors, a bank of four non-stepping rotors is equivalent to a simple substitution cipher, making it just as weak. In order to build a strong and secure cipher, it is essential to change the permutation for at least one of the rotors per electrical input. If rotors are given the ability to step, the number of possible permutations and level of security generated from the bank increases dramatically. Recall that it is best for a bank of rotors to have an unpredictable stepping pattern.

3. SIGABA the Cipher

3.1. A Historical Background of SIGABA

SIGABA, shown in Figure 7, reprinted from the National Security Agency [7], is a type of rotor-based cipher machine that was used by the United States during World War II. The development of this cipher is credited to the director of the US Army's Signals Intelligence Service, William Friedman, and his associate, Frank Rowlett. What separated SIGABA from all other ciphers used in World War II was its unique design that made it able to withstand all crypt-analytical attacks in the course of its usage.

Figure 7. SIGABA the cipher.

3.2. SIGABA's Developmental Phases

3.2.1. Pre-development phase

Friedman's first version of SIGABA was designed with the intent to fix the inherit weakness of single stepping rotor machines. A single stepping machine consisted of only one rotor so that with every input, the rotor advanced one step only to change the permutation slightly. Friedman's solution incorporated randomized stepping motions for the rotors that were controlled by a paper tape. This paper tape is similar to the one found in a teleprinter, which is now an obsolete electro-mechanical typewriter used to communicate typed messages via simple electrical connections. Electric signals were established when electricity was able to pass through the pattern of holes punched throughout the tape that determined the stepping motion for the rotors. For every rotor that advances, the tape will follow by advancing a step as well [17].

3.2.2. M-134

The M-134 is the first of Friedman's design that went into a limited production. M-134's key size depended on the pattern of the holes punched throughout the paper tape and the plugboard's setting that determined which holes were connected to which rotors. One significant disadvantage to the M-134 was the fragility of the paper tapes that often broke under harsh field conditions [17].

3.2.3. M-134-C

In response to M-134's weakness caused by the paper tape, Friedman and Rowlett replaced the tape with another set of rotors. However, because there was a lack of funds to develop an entirely new cipher, Friedman and Rowlett created an additional external

device to work concurrently with M-134. Friedman and Rowlett gave this new external device the name SIGGOO or M-229, which was a box that contained three rotors.

The SIGGOO or M-229 took five active inputs for every letter typed into the cipher and had a maximum of five active outputs that determined stepping motion of the rotors in M-134. The combination of M-134 and SIGGOO or M-229 is known as M-134-C [17].

3.2.4. ECM Mark II

In 1937, Navy Commander Laurance Safford, Friedman's counterpart in the Navy's Office of Naval Intelligence, saw great potential in the M-134-C. He and Commander Seiler made further improvements to the M-134-C that resulted in an easier method to build and transport the cipher. Their new generation of the machine was given a new name as well – the Electric Code Machine (ECM) Mark II, also known as CSP-888 or 889 [17].

A separate version of ECM Mark II, known as POTUS-PRIME, was developed specifically to provide communication between the President of the United States and the Prime Minister of the United Kingdom. All rotors were set by hand, in which the settings were specified in a codebook known as Message Indicators [1].

3.2.5. SIGABA

In the early 1940s, the Army learned of the Navy's usage of ECM Mark II. In 1941, the Army and the Navy formed a joint cryptographic system based on ECM Mark II, which became famously known as SIGABA within the Army.

Figure 8, reprinted from the USPTO Databases [11], is a diagram of SIGABA found from the original secret U.S. patent that was issued in 1944. It was not until 2001 that the patent was made available for the public [11].
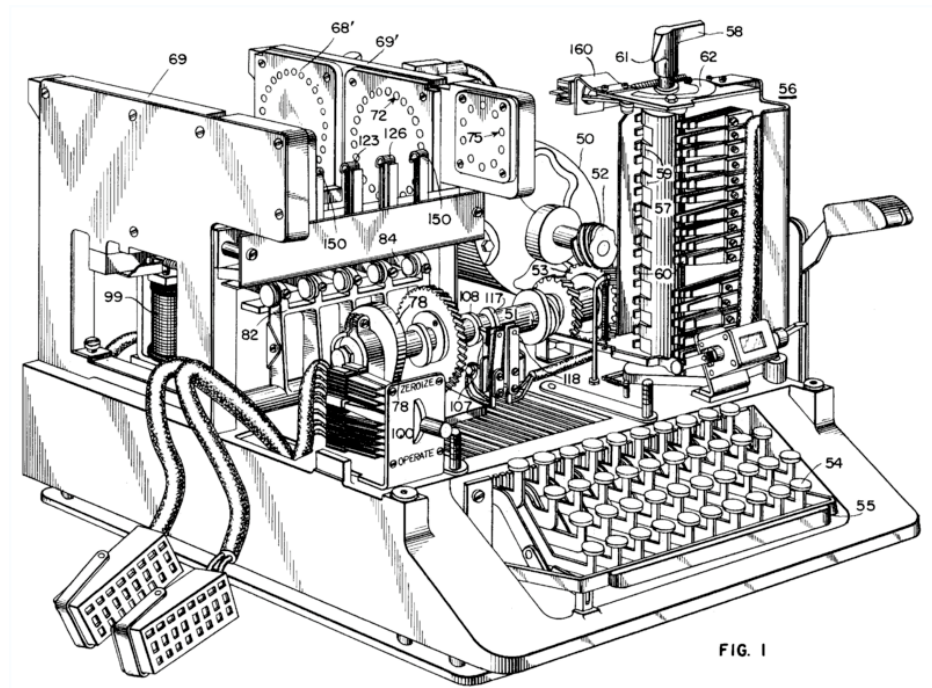
Figure 8. SIGABA.

3.3. A Technical Description of SIGABA

3.3.1. Overview

SIGABA consists a total of a total of fifteen rotors – three times the amount of the German Enigma. Out of SIGABA's 15 rotors, five are cipher rotors, five are control rotors, and five are index rotors. The cipher rotors are used to permute SIGABA's inputs while the control and index rotors are used to control the cipher's stepping pattern.

Figure 9, reprinted with permission from Chan [2], outlines the general layout of all 15 rotors. Notice that the order in which the electrical input reaches the rotors is control, index, and lastly cipher.
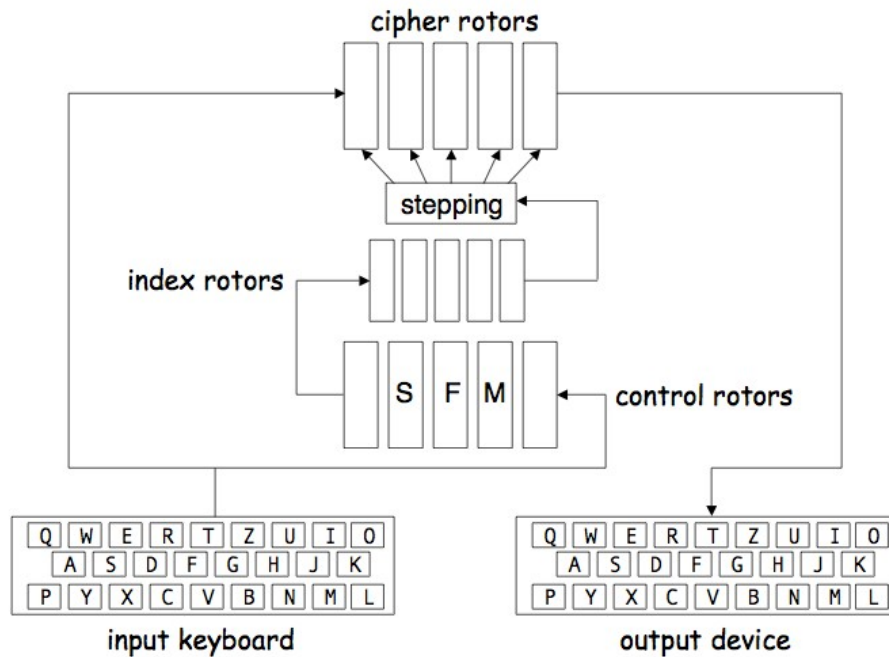
Figure 9. SIGABA's 15-rotor layout.

3.3.2. Cipher and Control Rotors

Both the cipher and control rotors are identical in appearance and function, making them interchangeable. Therefore, there are a total of 10 rotors available to form two banks of five rotors, one for each the cipher and control rotor banks. Each of the rotors is a 26-contact rotor with the alphabet letters A through Z printed on the outer edge as shown in Figure 4 and Figure 5.

With the exception of the appearance of the letters along the outer edge on both the cipher and control rotors, the left face is identical to the right face. Because of this property, all cipher and control rotors can be inserted backwards to operate in the reverse orientation.

If a rotor is operating in forward orientation, the stepping of the rotor is in an upward direction. Figure 10, reprinted with permission from Chan [2], illustrates either a control or cipher rotor advancing two steps in forward orientation. Note that the initial setting O changes to N with the first step, and changes to M on the second step.
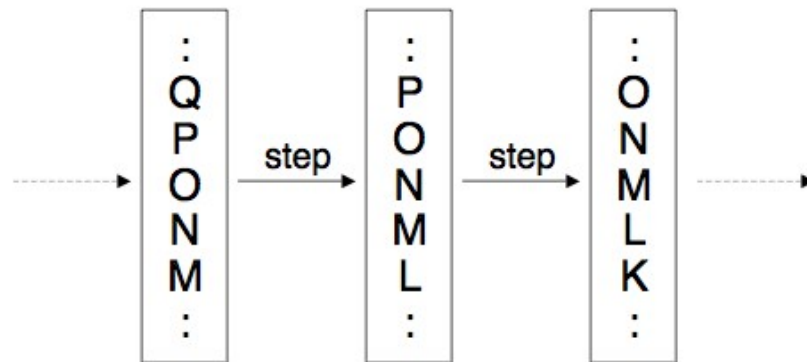
Figure 10. Stepping in forward orientation.

If the same rotor is operating in reverse orientation, the rotors will physically appear upside down on the machine and step in a downward direction. Notice how in Figure 11, reprinted with permission from Chan [2], the initial position of O steps to P, which steps to Q.
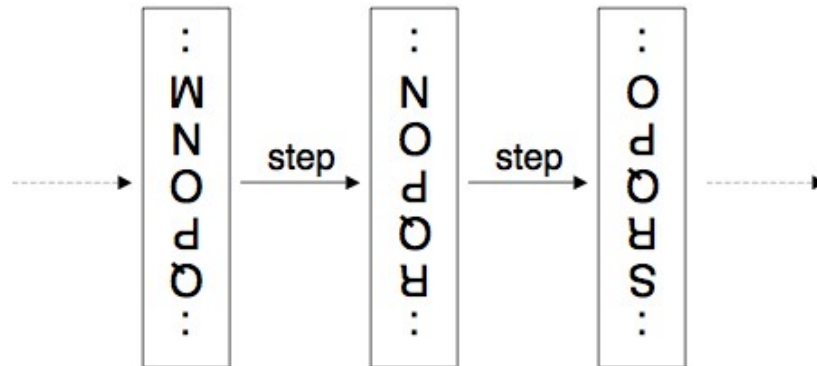
Figure 11. Stepping in reverse orientation.

3.3.3. Index Rotors

Index rotors are similar in design to the cipher and control rotors, except index rotors are 10-contact rotors instead of 26. Each of SIGABA's index rotors permutes integers ranging from zero to nine. Index rotors also do not step like the cipher and control rotors do. Although they may be placed backwards on the machine, setting index rotors in reverse orientation does not affect the cryptanalysis of SIGABA so this feature will be ignored.   The following picture in Figure 12, reprinted from Maritime Park Association [5], is of an actual cipher or control rotor and an index rotor. The smaller of the two is the index rotor.
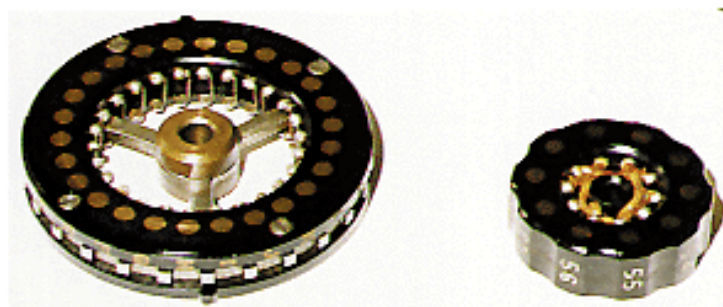
Figure 12. SIGABA's rotors.

3.4. Stepping SIGABA

The stepping of the rotors in encryption mode start with the control rotors, which we will denote as C_0, C_1, C_2, C_3, and C_4 from left to right. These rotors are illustrated in Figure 13.
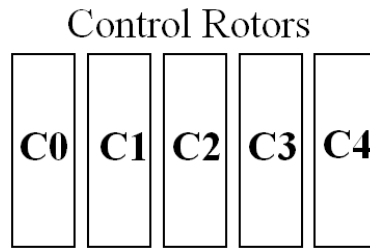
Control Rotors

Figure 13. Control rotors C0 through C4.

Rotors C_1, C_2, and C_3 step in an almost odometer-like fashion: C_2 steps for every input, C_3 steps once for every complete rotation of C_2 (26 steps), and C_1 steps once for every complete rotation of C_3 (26 steps). Essentially, C_2 is the fast rotor, C_3 is the medium rotor, and C_1 is the slow rotor. Rotors C_0 and C_4, however, do not step and remain in their initial setting positions.

For every input into SIGABA, C_4 receives four concurrent active inputs we assume to be *F*, *G*, *H*, and *I*. Based on the control rotors' permutations, four new letters are outputted from control rotor C_0. Before the output signals are sent to the index rotors, they are combined by the following logical operations listed under Equation (3).

$$
\begin{aligned}
I_1 &= B & I_4 &= F \vee G \vee H & I_7 &= P \vee Q \vee R \vee S \vee T \\
I_2 &= C & I_5 &= I \vee J \vee K & I_8 &= U \vee V \vee W \vee X \vee Y \vee Z \qquad (3) \\
I_3 &= D \vee E & I_6 &= L \vee M \vee N \vee O & I_9 &= A
\end{aligned}
$$

where

I_i is the ith input into the index rotors, and $i \in \{0,1,\ldots,9\}$.

For example, I_4 will be active if any of the four outputs from the control rotors include *F*, *G*, or *H*. Although there is an input I_0, it is never an active signal. For this reason, I_0 is disregarded in Equation (3).

At least one and at most four of the index inputs and outputs will be active. This means that at least one and at most four of the cipher rotors will step. The cipher rotors step according to the logical equations listed in Equation (4). Recall that the index rotors do not step themselves. For example, cipher rotor C_4 will step if either O_1 or O_2 is an active output from the index rotors.

$$C_4 = O_1 \vee O_2 \qquad\qquad (4)$$
$$C_3 = O_3 \vee O_4$$
$$C_2 = O_5 \vee O_6$$
$$C_1 = O_7 \vee O_8$$
$$C_0 = O_0 \vee O_9$$

where

O_i is the ith contact or output of the index rotor, and $i \in \{0,1,\ldots,9\}$

C_j is the jth cipher rotor, and $j \in \{0,1,2,3,4\}$

When SIGABA is set to decryption mode, all but the cipher rotors are set and

function exactly the same as in encryption mode. When SIGABA is in encryption mode, the electric signals pass through the cipher rotors in the direction from left to right. When decrypting, the signals are sent in the direction from right to left, which is the equivalent to using the inverse cipher permutations.

3.5. SIGABA's Keyspace

The number of possible settings for the cipher, control and index rotors determines SIGABA's keyspace. Each choice of setting includes the choice of rotors, the initial settings, and the orientation of rotors for those that apply. Equation (5) calculates the maximum possible keyspace for SIGABA.

$$(26!)^{10} \bullet (10!)^5 \approx 2^{884} \bullet 2^{110} \approx 2^{994} \tag{5}$$

where

$(26!)^{10}$ is the total number of permutations for the cipher and control rotors
$(10!)^5$ is the total number of permutations for the index rotors

However, due to the index rotors' inability to step, only 10! out of the $(10!)^5$ permutations are distinct. This reduces SIGABA's keyspace to 2^{906} (see Equation (6)). This is still a considerably large number, but thankfully the actual keyspace for SIGABA is much less due to several restrictions applied to the rotors.

$$(26!)^{10} \bullet 10! \approx 2^{884} \bullet 2^{22} \approx 2^{906} \tag{6}$$

Historically, there were only 10 rotors available to form both the control and cipher rotor banks and 5 rotors available to form index rotor bank. Recall that the cipher

and control rotors can also be inserted in either forward or reverse orientation. Although each of the cipher and control rotors can be set to any of the 26 possible positions and the index rotors can be set to any of the 10 positions, they are usually set to known default settings specified in a Message Indicator.

Two examples of message indicators are in the Table 4, reprinted from the USPTO Databases [11], and Table 5, reprinted from the USPTO Databases, for secret, confidential, and restricted messages. For each of these three types of messages, the message indicator contains the following information:

- The order of cipher, control and index rotors for each day of the month.
- The orientation of cipher and control rotors, where R indicates reverse.
- 26 to 30 check groups to ensure all rotors are placed and positioned correctly.

Table 4. Message indicator example for secret messages.

```
Day    |       ROTOR ARRANGEMENT        |              SECRET
of     |     (for all classifications)  |                    | 26-30
Month  | Stepping Control | Alphabet    |  Index(Front)      | Check
       |      (Middle)    |  (Rear)     |   Alignment        | Group

  1    | 0R 4  6  2R 7    | 1 8  5 9 3R | 10 23 31 49 5      | R N H V C
  2    | 2  3R 9R 1  5    | 6 4R 8 7 0  | 14 25 33 46 59     | S E M N O
```

Table 5. Message indicator for confidential and restricted messages.

```
Day    |         CONFIDENTIAL           |            RESTRICTED
of     |                    | 26-30     |                    | 26-30
Month  |  Index(Front)      | Check     |  Index(Front)      | Check
       |   Alignment        | Group     |   Alignment        | Group

  1    | 12 28 31 44 53     | P W V M T | 17 25 36 43 58     | M C S D T
  2    | 15 20 32 48 56     | E H E W B | 10 27 34 42 56     | R S T H H
```

If the Message Indicator is known, SIGABA's keyspace is reduced to $2^{48.4}$ (see Equation (7)). For the attack implemented in this thesis, we assume the Message Indicator is unknown.

$$10! \bullet 2^{10} \bullet 10^5 \approx 2^{48.4}. \tag{7}$$

4. The Cryptanalysis of SIGABA

4.1. Overview

This attack is divided into two parts: the primary phase and the secondary phase. For both these phases, we require a certain amount of known plaintext and the corresponding ciphertext. The primary phase exhausts all possible cipher settings. Recall that each setting includes the choice of rotors, initial settings, and orientation for each rotor. Any cipher setting that is consistent to the known plaintext, we deem the setting causal. All other inconsistent settings are deemed random. For each causal setting we require a secondary phase.

The secondary phase exhausts all possible control and index settings. Recall that the option for rotors to operate in reverse orientation does not apply to index rotors. For each combination of causal, control, and index setting that is consistent with the known plaintext, we recover the key.

Dividing the attack into two phases reduces the work to recover the key by first eliminating random cipher settings and for causal cases only do we exhaust all possible control and index rotor settings. For both primary and secondary phases, we make the following assumptions:

- All three rotor banks are set independently.
- There are only five rotors available to form the set of index rotors.
- There are a total of 10 rotors to form both the cipher and control rotors.
- Only the cipher and control rotors can operate in reverse orientation.
- The inner workings of SIGABA are known.

Taking all the assumptions into consideration, SIGABA's keyspace is

$$10! \bullet 2^{10} \bullet 26^{10} \bullet 5! \bullet 10^5 = 2^{102.3}. \tag{8}$$

where

10!   is the number of permutations for choosing the order of cipher and control rotors since $\binom{10}{5} \bullet 5! \bullet 5! = 10!$

2^{10} is the number of possible orientations for both cipher and control rotors

26^{10} is the number of possible initial settings for cipher and control rotors

5!   is the number of permutations for setting the index rotors

10^5 is the number of possible initial settings for the index rotors

However, recall that only 10! out of the total $5! \bullet 10^5$ index settings are distinct since index rotors do not step. This reduces Equation (8) to

$$10! \bullet 2^{10} \bullet 26^{10} \bullet 10! = 2^{100.6}. \tag{9}$$

Further deductions to the keyspace can be made due to the fact that the outputs of the index rotors are ORed together. Recall from Equation (4) that the index outputs are combined by the following logical equations.

$$C_4 = O_1 \vee O_2$$
$$C_3 = O_3 \vee O_4$$
$$C_2 = O_5 \vee O_6$$

$$C_1 = O_7 \vee O_8$$

$$C_0 = O_0 \vee O_9$$

Based on these five equations, each permutation has 32 equivalent permutations. Take for example the following permutation of (5, 4, 7, 9, 3, 8, 1, 0, 2, 6). This means an input of 0 is be mapped to an output of 5, an input of 2 is mapped to 4, and so on. Now, let us generate the 32 equivalent permutations. From Equation (4), cipher rotor C_4 will step if either output O_1 or O_2 is active. If C_4 steps using the above index permutation example, C_4 will also step if the index permutation is (5, 4, 7, 9, 3, 8, 2, 0, 1, 6). Table 6 consists of all 32 equivalent permutations for the above example.

Table 6. 32 equivalent permutations.

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

Therefore, the number of actual distinguishable index permutations is $2^{16.8}$ (see Equation (10)). SIGABA's resulting total keyspace is reduced to $2^{95.6}$ (See Equation (11)).

$$10!/32 = 113,400 \approx 2^{16.8} \tag{10}$$

$$10! \bullet 2^{10} \bullet 26^{10} \bullet 2^{16.8} = 2^{95.6} \qquad (11)$$

4.2. Primary Phase

4.2.1. Review

The purpose of the primary phase is to eliminate all random settings and to keep only the causal settings. Recall that the primary phase exhausts all possible cipher settings, which includes the choice of the rotors, orientations, and initial settings. Only the settings that are consistent to the known plaintext are causal.

4.2.2. Keyspace for Primary Phase

The number of possible cipher settings determines the keyspace for the primary phase. Each setting includes the following factors:

- The choice of five rotors from 10 available rotors.
- The option for each cipher rotors to operate in forward or reverse orientation.
- A total of 26 possible initial positions for each cipher rotor.

Taking all these factors into account, there are $2^{43.4}$ possible cipher settings for the primary phase (see Equation (12)).

$$\binom{10}{5} \bullet 5! \bullet 2^5 \bullet 26^5 = 2^{43.4} \qquad (12)$$

26

4.2.3. Stepping Through the Primary Phase

Recall that for each letter typed into the cipher, a range of one to four cipher rotors can step. Suppose we have chosen our choice of cipher rotors, and the orientation and initial setting for each rotor has been set so that the first known plaintext letter encrypts to the first ciphertext letter. Before we determine if the second known plaintext letter encrypts to the second ciphertext letter, we simulate all possible steps the cipher rotors can take from the initial setting. The total number of possible steps is 30 since

$$\binom{5}{1}+\binom{5}{2}+\binom{5}{3}+\binom{5}{4}=30\,.$$

Each of these 30 new steps will give us a new cipher setting or permutation that we will use to encrypt the second known plaintext. Any of the new 30 settings that are consistent with the second pair of plaintext and ciphertext letters are kept, while the rest that are inconsistent are discarded. We will make each consistent setting the current cipher setting and generate an additional 30 steps each setting can take. Again, we keep each of the new 30 settings that are consistent with the third pair of plaintext and ciphertext letters and discard the rest. This process is repeated until all pairs of plaintext and ciphertext letters are tested.

Once all pairs of plaintext and ciphertext letters have been tested, we keep all initial cipher settings that have generated the cipher setting that is consistent with the last pair of plaintext and ciphertext letters. Recall that these cipher settings are what we call causal. If it is the case that the last pair of plaintext and ciphertext has not been reached and none of the new 30 cipher settings are consistent with the current pair of plaintext and ciphertext, we know the initial setting is random and discard it.

Note that the primary phase creates a treelike structure where the initial setting is

27

the root of the tree and each child node of the tree is every subsequent consistent cipher setting.

Assuming that the cipher permutations are uniformly random, the number of matches of plaintext to ciphertext follow a binomial distribution of $p = 1/26$ and $n = 30$, meaning the number of expected matches per step is $30/26 = 1.154$. Therefore, the number of paths is expected to increase at any given step. This branching effect may seem daunting, but keep in mind that we do not track the intermediate steps and the primary phase also merges settings of equal value. Consider the following example.

Suppose we have chosen our cipher settings so that the first plaintext encrypts to the first ciphertext and the initial position of the cipher rotors is set to AAAAA. Recall that because the initial setting yields a match between the first plaintext and ciphertext, the next step involves generating all 30 possible subsequent settings that can occur where the expected number of matches is 1.154. Taking this factor into account, suppose that out of the 30 new cipher settings, only two yield consistent results with the second pair of plaintext and ciphertext. Assume these cipher settings to be BBABA and ABABA as indicated in Figure 14, reprinted with permission from Chan [2].

Figure 14. First round of primary phase.

Settings BBABA and ABABA now become the current cipher settings and 30 possible cipher steps are generated for each. As indicated in Figure 15, reprinted with permission from Chan [2], only one setting from BBABA is consistent and two from ABABA are consistent with the third pair of plaintext and ciphertext. These three

settings are BBBBA, BBBBA, and ACBBA.
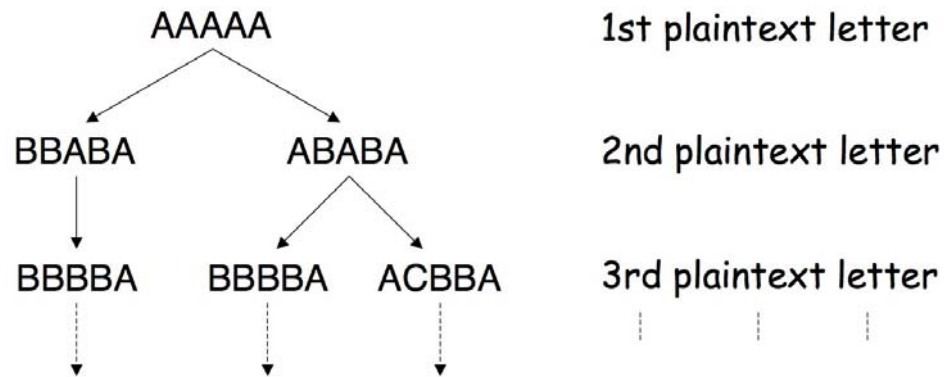
Figure 15. Second round of primary phase.

Notice that in the third round, two settings, BBBBA, are identical. In such situations, we can merge these two settings together. By merging these settings, rather than generating two sets of 30 new settings for each setting of BBBBA, we generate it only for one.

Applying the merging method does not cause us to lose accuracy to the primary phase since the primary phase is only concerned with the initial cipher setting and not the subsequent settings. The merging method is illustrated in Figure 16, reprinted with permission from Chan [2].

```
AAAAA                          1st plaintext letter

BBABA        ABABA             2nd plaintext letter

    BBBBA    ACBBA             3rd plaintext letter
```
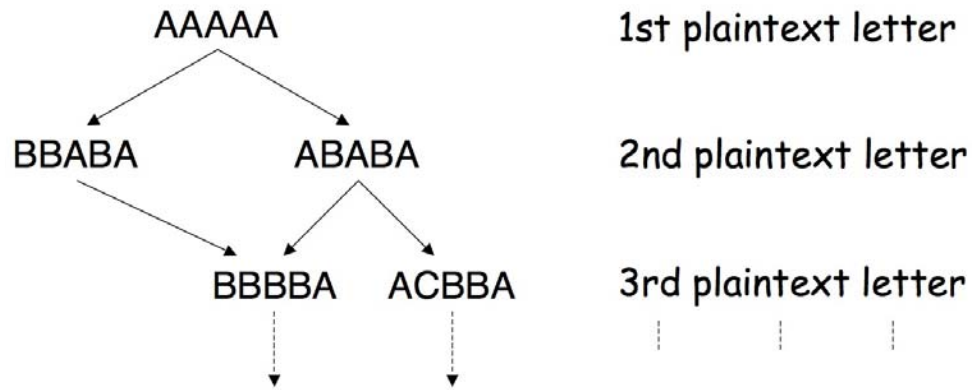
Figure 16. Third round of primary phase.

This process of generating 30 possible steps for each causal cipher setting while merging identical settings is repeated until all known plaintexts are tested or if none of the 30 subsequent permutations yield any consistent results.

If for this particular example there are only three known plaintexts available, then the primary phase will be complete and we apply the secondary phase to this particular causal setting.

4.2.4. Primary Phase Analysis

Let us consider the statistics from the primary phase for both causal and random settings. Given a set of known plaintexts that is indicated in the column labeled "Steps" in Table 7 and Table 8, a variable amount of "Tests" were conducted for each. Each "Test" consisted of generating either a random or causal cipher setting for the respective tables.

For each causal or random setting, the first known plaintext letter is encrypted and compared with the first ciphertext letter. If the first plaintext letter encrypts to the first ciphertext, then all subsequent 30 possible cipher steps are generated, where any of the 30 new settings that is consistent to second plaintext and ciphertext letters are saved. In the case for all random settings, the probability for it to survive or be consistent with the

plaintext is 1/26.   This process is repeated until all plaintext letters have been tested, if none of the 30 generated settings are consistent with the known plaintext, or if the maximum number of steps has been reached.

For each set of tests listed in Table 7, reprinted with permission from Chan [2], and Table 8, reprinted with permission from Chan [2], we count the number of settings to survive the primary phase along with the number of merged paths. For Table 7, we conduct 10^5 tests for each different case, where the number of random settings expected to survive out of the total 10^5 settings is denoted as "non-zero settings," and the number of merged paths to have been generated for each of the surviving settings is denoted as "Avg. per non-zero."

For example, we see in Table 7 that in the case of using 50 consecutive known plaintext letters, 290 out of 100,000 random settings are expected to survive with an average of 28.4 merged paths to have occurred with a maximum value of 194.

Looking at the same case in Table 8 for the causal case, we see that using 50 known plaintexts, each causal setting is expected to generate 54.1 consistent branches. Out of the 100,000 tests generated, the numbers of consistent paths range from one to 404.

Table 7. Random case.

Steps	Tests	Non-zero settings	Avg. per non-zero	Maximum
10	10^5	763	6.5	27
20	10^5	516	11.8	56
30	10^5	427	16.5	84
40	10^5	324	20.8	105
50	10^5	290	28.4	194
60	10^5	275	38.8	163
70	10^5	269	47.1	415
80	10^5	212	71.3	524
90	10^5	216	77.6	486
100	10^5	203	100.5	1005

Table 8. Causal case.

Steps	Average	Maximum	Minimum	Tests
10	10.2	51	1	10,000
20	19.6	94	1	10,000
30	29.6	151	1	10,000
40	40.1	237	1	10,000
50	54.1	404	1	10,000
60	69.2	566	1	10,000
70	85.0	689	1	5,000
80	105.0	829	2	5,000
90	130.4	1152	1	3,000
100	161.1	1926	1	3,000

The results of Table 7 are favorable, as we can see that most of the random settings are eliminated. For this attack, this is all we need to concern ourselves with. However, for the secondary phase refinement (see 6.1), the results of Table 7 can be a negative aspect of the primary phase. The secondary phase refinement deals with using all surviving merged paths, and we can see that as more plaintext letters are used, the number of total merged paths increases as well, thereby increasing the work.

We can further reduce the number of random settings by only keeping random settings that lie above the mean found in Table 8 for each respective causal case. However, this comes with the risk of eliminating causal settings and thereby decreasing the success rate of this attack. For this reason, we will not apply this checkpoint for the cipher settings.

4.3. Secondary Phase

4.3.1. Description

The secondary phase exhaustively permutes through all possible control and index settings for every causal cipher setting from the primary phase. If the combination of all

three-rotor settings encrypts all plaintext letters to its respective ciphertext, we have recovered the key. If none of the exhausted control and index settings gives consistent results with the causal cipher setting, then the cipher setting has survived the primary phase by chance and is discarded. This is the simpler method of two possible secondary phases.

The alternative option is called the refined the secondary phase (see 6.1). Although the refinement has not been fully implemented, analysis has been done for selected issues mentioned in the refinement that prove its correctness [10]. The work I implement in the secondary phase also adds to the analysis already done in *Cryptologia*.

4.3.2. Keyspace for Secondary Phase

Although there are 10 rotors that can be used to construct the set of control rotors, five have already been designated for the set of cipher rotors. This leaves a total of 5! Possible rotor choices, each with 26 possible initial positions, and each with the ability to operate in reverse orientation. With only five index rotors, there are also 5! Possible index rotor choices, with 10 initial possible initial positions each. Recall that the index rotors do not operate in reverse orientation. This gives the secondary phase a total of

$$5! \bullet 2^5 \bullet 26^5 \bullet 5! \bullet 10^5 \approx 2^{58.9} \tag{13}$$

possible settings.

This workload is considerable since it is applied to each causal cipher setting that survives the primary phase. However, recall from Equation (10) that there are only $2^{16.8}$ distinguishable index permutations. This reduces the secondary phase keyspace to

$$2^{16.8} \bullet 5! \bullet 2^5 \bullet 26^5 \approx 2^{52.2} \tag{14}$$

that is applied to each causal setting.

5. Implementation

5.1. Overview

The cryptanalysis, which includes both the primary phase and the simpler secondary phase, is developed using Microsoft Visual C++ .Net version 7.1. The cryptanalysis project is named SigabaCryptanalysis. The keyspace for the primary and secondary phase under the worst-case scenario is

$$2^{43.4} \bullet 2^{53.2} \approx 2^{96.6}. \tag{15}$$

That is, if every cipher setting survives the primary phase, the attack will further permute $2^{53.2}$ settings to recover control and index settings in the secondary phase. Because applying the attack to SIGABA's full keyspace still takes a considerable amount of time to compute, (see 5.4), I ease the process by allowing the user to specify a range of cipher and control rotors to target.

I initially created two side projects in Microsoft Visual C++ version 6.0 to create data files for SigabaCryptanalysis to use. These two projects are named distinguishablePermutations and distinguishableSettings.

The project distinguishablePermutations generates all 10!/32 (113,400) distinguishable index permutations and writes them into a text file, distinguishablePermutations.txt (see pocket).

The project distinguishableSettings determines which index setting (choice of index rotors and initial settings) generates each of the distinguishable index permutations. The results are written into distinguishableSettings.txt (see pocket). There are four

columns in distinguishableSettings.txt: the row number the permutation is located in distinguishablePermutations.txt, the index permutation, the index order, and initial setting in order from left to right.

What I have found is that not all index permutations can be generated from the index rotors. I created a third project called distinctPermutations to exhaust all index settings and find all distinct permutations. However, the end results showed only 1,811,873 permutations out of the expected 10! are able to be generated from the index rotors. These permutations are in distinctPermutations.txt (see pocket). This is a factor I had to take into account while generating all 10!/34 index permutations for distinguishablePermutations (see 5.2.2). Without taking this fact into consideration, only 56,567 out of the 113,400 distinguishable permutations ended up having an equivalent index setting.

5.2. Pseudocode

5.2.1. SigabaCryptanalysis

Main( )
{

      FOR each possible cipher setting

            Apply the primary phase

            IF primary phase returns value > 0

                  Keep current causal cipher setting

                  Determine which rotors are available to make the control rotors

                  FOR each possible control setting

                        FOR each distinguishable index setting

                              Apply secondary phase

                              IF secondary phase returns value of 1

Keep the current setting for all ciphers

ELSE

Discard current setting and keep permuting

ELSE cipher setting is random

}

int Primary Phase( )

{

Place cipher rotors in correct order

Set initial positions for all cipher rotors

Set orientation of cipher rotors

Save current cipher rotor offsets before any rotor steps

IF first plaintext encrypts to first ciphertext

WHILE there are plaintexts to test

IF there are no saved cipher rotor offset

Break

FOR each saved cipher rotor offset

Set the cipher rotors configuration to saved offsets

FOR each of 30 possible cipher rotor step

IF next plaintext encrypts to next ciphertext

IF offset is not the same as any offset

generated by one of the 30 steps

Save the offset

Delete all previous offsets

Keep only the new offsets generated from the 30 new steps

that are consistent to known plaintext and ciphertext

ELSE

Do nothing

RETURN number of saved offsets

```
}


int secondaryPhase( )

{

        Place cipher rotors from primary phase in correct order

        Set initial positions for all cipher rotors

        Set orientation of cipher rotors

        Place control rotors in correct order

        Set initial positions for all control rotors

        Set orientation of control rotors

        Place cipher rotors in correct order

        Set initial positions for all cipher rotors

        Set orientation of cipher rotors

        Place cipher rotors in correct order

        Set initial positions for all cipher rotors

        Set orientation of cipher rotors


        Set Boolean variable match to true to indicate whether encrypted plaintext
        encrypts to ciphertext


        WHILE there are plaintexts to test
                Send plaintext through control rotors
                Use control outputs as index rotor inputs
                Step cipher rotors based on index rotor outputs
                Output encrypted plaintext
                IF encrypted plaintext is not equal to ciphertext
                        Set match to false
        RETURN match

}
```

5.2.2. distinguishablePermutations

main( )

{

      Create an array to hold all distinct permutations

      Generate all 3,628,800 permutations for "0123456789"

      Create a map structure so that each permutation is a key paired with the value

      FALSE

      FOR each of the 3,628,800 permutations

            IF permutation can be generated by index rotors

                  IF map[permutation] is **FALSE**

                        Generate32Equivalents(array, permutation, map)

      Write all contents of array to distinguishablePermutations.txt.

}

void Generate32Equivalents(array, permutation, map)

{

      FOR 32 rounds

            Swap the all indices that are mapped together via Equation (4)

      FOR each of the 32 equivalent permutations

            Flag each permutation in the map to **TRUE**

}

5.2.3. distinguishableSettings

main( )

{

      Create a map structure so that each key is a distinct index permutation in

      indexPermutation.txt paired with the row index number (map1)

Create a map structure so that each key is an index ranging from 0 to 113,400 paired with the value FALSE (map2)

FOR each possible index setting

      Place index rotors in correct order

      Set initial positions for all index rotors

      Generate the permutation from the index rotors

      IF the permutation is one of the distinct permutations (map1[permutation])

            Get the index of the permutation in indexPermutations.txt

            IF an equivalent permutation has not already been written (!map2[index])

                  Write the index setting to distinguishableSettings.txt

}

5.3. Executable

5.3.1. Description

Figure 17 is the GUI developed for SIGABA's cryptanalytic attack. There are inputs to specify both cipher and control settings, each of which include the choice and order of rotors, initial settings, and orientation for each rotor. The left input for each of these settings indicates the start of the target range and the input on the right indicates the end of the target range. If the start of the range is specified without an ending value, this will indicate to target the starting range value only. If neither start nor ending range is specified for any setting input, this will indicate to the program to exhaust all possible settings.

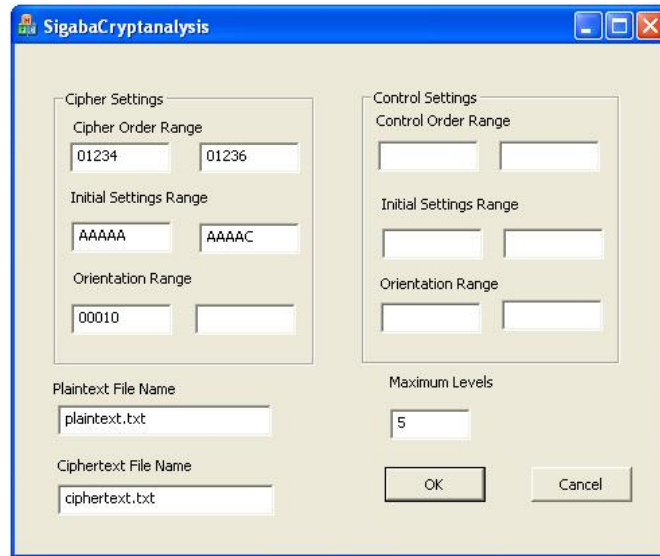Additional inputs include the filenames containing the known plaintext and ciphertext, and the maximum number of letters of the known plaintext to test.
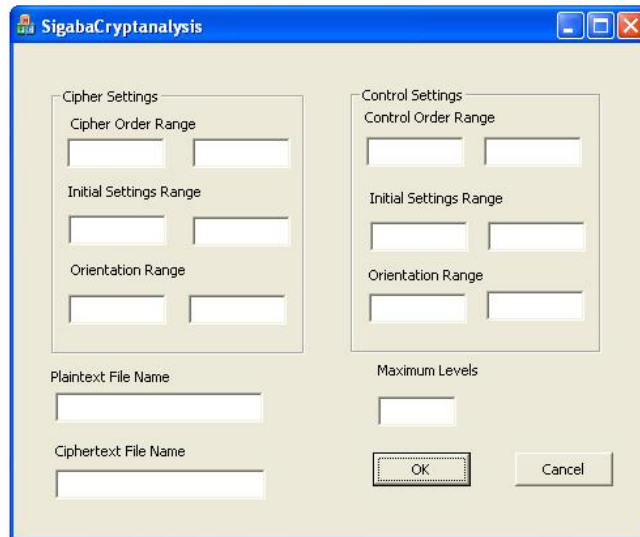
Figure 17. GUI for SIGABA's cryptanalysis.

5.3.2. Example Usage

Consider the following example shown in Figure 18 in which we specify a targeted range for SIGABA's cipher rotors.

Figure 18. Example GUI use.

Based on these input values in Figure 18, the primary phase will be applied for all nine cipher settings specified in Table 9.

Table 9. Example cipher settings to primary phase.

Call to Primary	Cipher Order	Cipher Init	Cipher Orient
1	01234	AAAAA	00010
2	01234	AAAAB	00010
3	01234	AAAAC	00010
4	01235	AAAAA	00010
5	01235	AAAAB	00010
6	01235	AAAAC	00010
7	01236	AAAAA	00010
8	01236	AAAAB	00010
9	01236	AAAAC	00010

5.4. Time Analysis

Table 10 and Table 11 lists specific tasks performed in this attack along with the amount of time it takes to compute.

Table 10 shows the amount of time it takes to permute through all possible rotor settings in this attack, and does not include any function calls to the primary or secondary phase, which are separately indicated in Table 11. Recall that permuting through the cipher settings are nested within each other (see 5.2), similar to the following pseudocode:

FOR all possible cipher order settings
    FOR all possible initial cipher settings
        FOR all possible cipher orientations

Likewise, permuting through all control and index rotors for the secondary phase follows the structure of the following pseudocode:

FOR all possible control order settings
    FOR all possible initial control settings
        FOR all possible control orientations
            FOR all possible distinct index permutations

Table 10. Time analysis to permute through all possible settings.

Rotor Setting to Permute	Total Iterations	Time (milliseconds)
Cipher Order	30,240	0.708
Initial Cipher Setting	26^5	6017.025
Cipher Orientations	32	0.0
Control Order	5!	0.099
Initial Control Setting	26^5	6115.415
Control Orientations	32	0.0
Index Permutations	113,400	0.0

The results of Table 10 indicate that the amount of time it takes to compute all possible cipher settings in the primary phase is about

$$0.708 + 30,240 \bullet (\ 6{,}017.025 + 26^5 \bullet 0.0\ ) \approx 250{,}939{,}057.786 \text{ ms}$$
$$= 250{,}939.058 \text{ sec}$$
$$= 4182.318 \text{ min}$$
$$= 69.705 \text{ hours.}$$

The amount of time to compute all control and index settings in the secondary phase for each surviving cipher setting is about

$$0.099 + 5! \bullet (\ 6115.415 + 26^5 \bullet (\ 0.0 + 32 \bullet 0.0\ )\ )$$
$$\approx 9220003.939 \text{ ms}$$
$$= 9220.004 \text{ sec}$$
$$= 153.667 \text{ min}$$

$$= 2.561 \text{ hours.}$$

Table 11 shows the different times it takes to calculate the primary phase for the same set of plaintext given different numbers of plaintext letters to test ("Max Value" column).

Table 11. Time analysis for primary phase.

Max Value	5	10	15	20	25
Time (ms)	1.623	2.925	3.801	5.418	6.586

Max Value	50	75	100	125	150
Time (ms)	28.356	59.805	103.571	142.282	171.788

6. Future Work

6.1. Secondary Phase Improvements

6.1.1. Description

Further work can be done to the secondary phase to improve the workload by using all the merged paths that survived the primary phase versus just the initial cipher settings. However, recall from Table 7 and Table 8, this can be a disadvantage since the numbers of merged paths tend to increase as more plaintext letters are tested.

6.1.2. Inner Workings of Control and Index Rotors

Before discussing the details of the secondary phase refinement, let us first review how the control and index rotors operate. For every letter typed into SIGABA, the inputs F, G, H, and I are activated into the control rotors simultaneously. Based on these active signals and the control rotors' permutation, one to four of the control rotor outputs or index rotor inputs will be active. After these signals are permuted through the index rotors, one to four of the index outputs will be active. The index outputs are combined in pairs as indicated in Figure 19, reprinted with permission from Chan [2], and Equation (4). Depending on which index outputs are active, one to four of the cipher rotors will step.
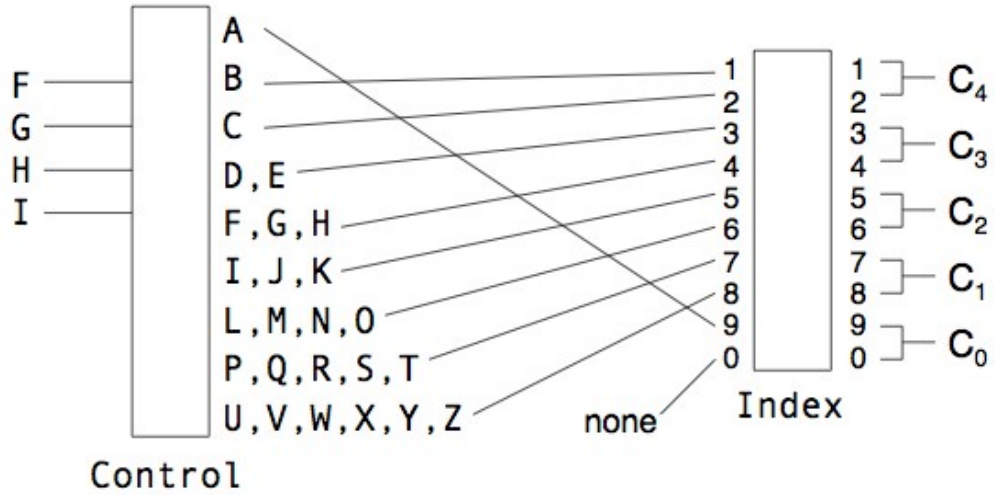
Figure 19. Control to index rotor mapping.

For example, if the outputs of the control rotors are *A*, *M*, and *N*, index inputs 9 and 6 will be active. Which of the cipher rotors will step depends on the index permutation. Consider the following example to illustrate this point.

Suppose the index inputs (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) are mapped to outputs (5, 4, 7, 9, 3, 8, 1, 0, 2, 6) accordingly. Therefore, under this permutation, if index input 0 is active, output 5 will be active, which results in cipher rotor C_2 taking a step.

For each input into SIGABA, we assume all $\binom{26}{4}$ outputs from the control rotors are uniformly random. However, notice the control rotor outputs are not grouped uniformly. For example, the probability for index input 8 to be active is much higher than the probability for index inputs 1, 2, and 9 to be active. Therefore, the probability for each cipher rotor to step is directly related to which group of control rotor outputs it is mapped to via the index permutation. Table 12, reprinted with permission from Chan [2], illustrates this point for the current index permutation of (5, 4, 7, 9, 3, 8, 1, 0, 2, 6).

Table 12. Index to cipher example.

	Cipher rotor				
	C_4	C_3	C_2	C_1	C_0
index rotor outputs	(1,2)	(3,4)	(5,6)	(7,8)	(9,0)
index rotor inputs	(6,8)	(4,1)	(0,9)	(2,5)	(3,7)
control rotor count	10	4	1	4	7

From Table 12, cipher rotor C_4 will step if index rotor output 1 or 2 is active. From the above permutation, index output 1 or 2 is active if input 6 or 8 is active. Index input 6 or 8 is active if the control rotor outputs belongs in the groups L, M, N, O or U, V, W, X, Y, Z respectively, which gives "control rotor count" in Table 12 a value of 10. In comparison, we see cipher rotor C_4 steps far more than cipher rotor C_2, which has a "control rotor count" value of 1. Therefore, the outputs from the index rotors are not evenly distributed, meaning the probability for each cipher rotor to step is not equally likely.

6.1.3. Secondary Phase Algorithm

The goal of this refinement is to reduce the number of index permutations by using the information gained in the primary phase. From the primary phase, we know the initial and ending positions for the cipher rotors that encrypt the plaintext to the ciphertext. By knowing these two positions, we can determine how much each cipher rotor steps. For example, if the five initial positions are set to AAAAA and ends at positions DCBAA, we can infer cipher rotor C_0 stepped three times, C_1 stepped twice, C_2 stepped once, and C_1 and C_0 did not step at all.

Table 13, reprinted with permission from Chan [2], lists all 45 possible input pairs into the index rotors. Associated with each input pair is the number of control rotor outputs that are directly connected ("Letters").

Table 13. Index input

Letters	Count				Pairs			
1	3	(0,1)	(0,2)	(0,9)				
2	4	(0,3)	(1,2)	(1,9)	(2,9)			
3	5	(0,4)	(0,5)	(1,3)	(2,3)	(3,9)		
4	7	(0,6)	(1,5)	(2,5)	(5,9)	(1,4)	(2,4)	(4,9)
5	6	(0,7)	(1,6)	(2,6)	(6,9)	(3,4)	(3,5)	
6	6	(0,8)	(1,7)	(2,7)	(7,9)	(3,6)	(4,5)	
7	6	(1,8)	(2,8)	(8,9)	(3,7)	(4,6)	(5,6)	
8	3	(3,8)	(4,7)	(5,7)				
9	3	(4,8)	(5,8)	(6,7)				
10	1	(6,8)						
11	1	(7,8)						

pairs.

By knowing the number of times each cipher rotor steps, we can gain information related to the "count" column of Table 13. From the "pairs" column, we can then infer index permutation restrictions.

The data in Table 14, reprinted with permission from Chan [2], lists the "stepping ratio" for a cipher rotor when it is connected to any possible number of control rotor outputs, which range from one to 11. To compute these ratios, we generate all possible $\binom{26}{4} = 14{,}950$ outputs from the control rotors. Recall that each of these outputs is assumed to be equally likely. For each set of generated control outputs, we count the number of times each pair of index inputs occur, which is indicated in the "step count" column. The ratios are then calculated by dividing each step count by 14,950.

Table 14. Stepping Ratios.

letters	example pair	step count	step ratio
1	(0,1)	2,300	0.153846
2	(0,3)	4,324	0.289231
3	(0,4)	6,095	0.407692
4	(0,6)	7,635	0.510702
5	(0,7)	8,965	0.599666
6	(0,8)	10,105	0.675920
7	(1,8)	11,074	0.740736
8	(3,8)	11,890	0.795318
9	(4,8)	12,570	0.840803
10	(6,8)	13,130	0.878261
11	(7,8)	13,585	0.908696

By combining the information gained in Table 13 and Table 14 with the putative cipher stepping counts, we can determine which control outputs are most likely mapped to which cipher rotor. Since the mapping between control outputs and cipher rotors are dependent on the index permutation, we obtain restrictions for the index permutations.

Note for an index permutation to be valid, it is required to contain five pairs of index inputs listed in Table 13 in which the numbers zero through nine appear only once. Another requirement is that the associated number of control outputs sum to 26, since all 26 letters must be connected.

In the simpler secondary phase, 10!/32 distinguishable index permutations are tested against the plaintext for each control setting. This refinement further reduces this number to 2^8, which is significantly smaller [10]. Equation (16) calculates the new work factor for the secondary phase.

$$2^8 \bullet 5! \bullet 2^5 \bullet 26^5 \approx 2^{43.4} \qquad (16)$$

Compared to the work factor for the simpler secondary phase computed in Equation (14), this new work factor is significantly reduced. In addition to reducing the number of index permutations, further work can be done in the primary phase to reduce the number of merged paths.

I created my project to have a modular design by creating a separate function for each task I needed to complete. In doing so, the simpler secondary phase is its own function that is independent of everything else, which includes the primary phase and rotor settings set up. This way, future students who develop the refined secondary phase can take the work I have already done and simply substitute the simpler secondary phase function with their own.

6.2. CrypTool

6.2.1. Introduction

CrypTool is a free interactive learning tool built for Windows that is available worldwide for education or training purposes in universities and industries. CrypTool is available in three different languages including English, German and Polish, and focuses on cryptology. Users are allowed to run and analyze both classic and modern ciphers available in CrypTool. Some of these ciphers include the Enigma, Caesar, RSA and AES. CrypTool also includes cryptographic protocols such as the Diffie-Hellman key exchange, and number theories like the Chinese Remainder Theorem that are useful for other cryptosystems.

In addition to cipher simulations, CrypTool also includes applications for cryptanalytic attacks for various ciphers. Since most of these attacks are not intuitive, CrypTool also provides documentation outlining and explaining the attacks in full detail.

CrypTool is also an open source project that allows users to enhance its features by adding new applications. Because CrypTool does not currently include any application to simulate SIGABA cipher or any cryptanalysis pertaining to the cipher, future work can be done to integrate my thesis into CrypTool. In doing so, users can be further educated on what makes a rotor-based cipher strong and secure. For users who are up to the challenge, they can either develop and add the secondary phase refinement into CrypTool or develop additional refinements to the attack that are not already mentioned in this thesis.

6.2.2. Usage

CrypTool targets users of all backgrounds: people with little or no knowledge on cryptology to people with a comprehensive background. CrypTool's intuitive design makes it easy for users to understand and use and also includes an extensive help feature that comes equipped with several step-by-step demonstrations and explanations to help users become familiar with the application as well as more knowledgeable on the cryptographic method.

One of the positive attributes of CrypTool is the interactive feature. Figure 20 and Figure 21 illustrate activating an interactive application of the Enigma cipher in CrypTool.
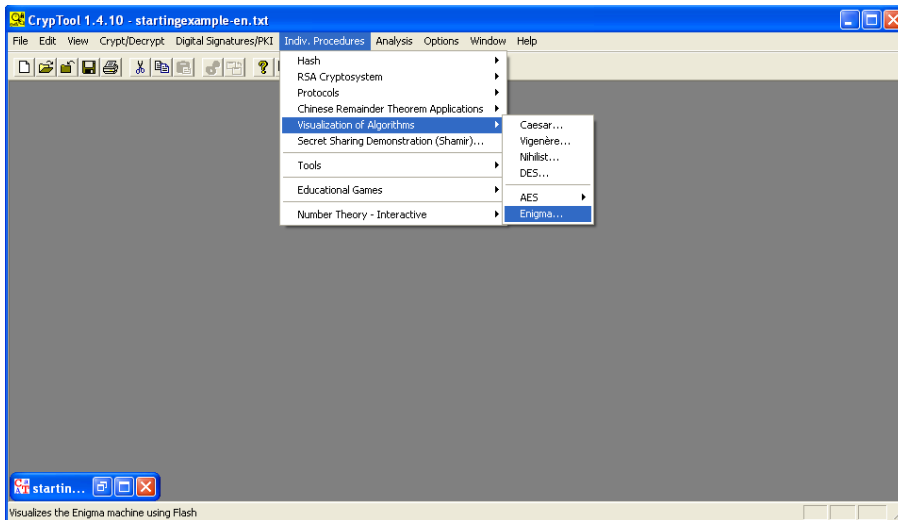
Figure 20. CrypTool application.
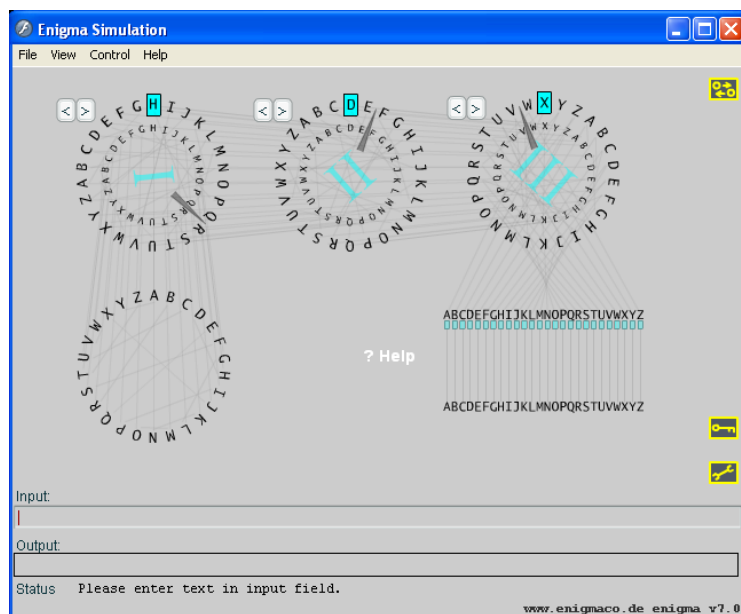
Figure 21. Enigma cipher application.

7. Conclusion

This thesis covers SIGABA: what it is, how it was made, and the significance it has played in history. This thesis also covers a modern day attack on SIGABA, which was developed in *Cryptologia*, but I implemented for the first time.

SIGABA's total keyspace of 2^{906} is infeasible to exhaustively search. Even under the assumptions that either the message indicator was intercepted or that the attacker knows all the available 15 rotors and the internal workings of the cipher, the respective keyspaces of $2^{48.4}$ and $2^{95.6}$ would still have been impossible to break in World War II (see 5.4), given the limited computing power available in the 1940s.

To reduce the workload of recovering the key, the cryptanalytic attack on SIGABA is divided into two phases, the primary and secondary phase. Both of these phases target SIGABA's rotor banks separately, ultimately resulting in recovering the key.

The primary phase exhausts all possible cipher settings and keeps every setting that is consistent with the known plaintext, known as causal settings. For each causal setting we require the secondary phase. The secondary phase exhausts all control and 10!/32 distinct index settings. If the combination of the causal setting from the primary phase, the index permutation, and control settings is consistent with the known plaintext, we have recovered the key.

There is still a considerable amount of work that can be done to refine this attack. Possibilities lie in either reducing the number of merged paths in the primary phase or reducing the number of index permutations used in the secondary phase. Though SIGABA is now obsolete and considered broken, the fact that so much research can still be done to break this cipher proves Friedman and Rowlett's design remains amazing even

to this day.

References

[1] A Cryptographic Compendium. "The ECM Mark II, also known as SIGABA, M-134-C, and CSP-889." Electrical and Mechanical Cipher Machines, http://www.quadibloc.com/crypto/ro0205.htm (accessed September 1, 2008).

[2] Chan, Wing. "Cryptanalysis of SIGABA." MS diss., San Jose State University, 2007.

[3] Hellman, Martin E. "Privacy and Authentication: An Introduction to Cryptography." Proceedings of the IEEE 67, no. 3 (March 3, 1979), http://www.cs.berkeley.edu/~prabal/resources/osprelim/DH79.pdf

[4] Low, Richard, and Stamp, Mark. Applied Cryptanalysis. Hoboken, New Jersey: John Wiley and Sons, Inc., 2007

[5] Maritime Park Association. "Electronic Cipher Machine Mark II." USS Pampanito, http://www.maritime.org/ecm2.htm (accessed June 10, 2008).

[6] Maritime Park Association. "Operating Instructions for ASAM 1 (a.k.a. ECM Mark II)". USS Pampanito, http://www.maritime.org/ecminst.htm (accessed November 1, 2008).

[7] National Security Agency Central Security Service. "Photo Gallery." National Security Agency, http://www.nsa.gov/public/publi00007.cfm (accessed June 10, 2008).

[8] Oracle Think Quest Library. "Letter Frequency Analysis." Think Quest, http://library.thinkquest.org/28005/flashed/thelab/cryptograms/frequency.shtml (accessed September 1, 2008).

[9] Stamp, Mark. *Information Security Principles and Practice*. Hoboken, New Jersey: John Wiley and Sons, Inc., 2006.

[10] Stamp, Mark and Chan, Wing. "SIGABA: Cryptanalysis of the Full Keyspace." *Cryptologia* 31 (2007): 201 – 222.

[11] United States Patent and Trademark Office. "USPTO Patent Full Text and Image Database." United States Patent, http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrc hnum.htm&r=1&f=G&l=50&s1=6175625.PN.&OS=PN/6175625&RS=PN/6175625 (accessed October 10, 2008).

[12] Wikipedia The Free Encyclopedia. "Alberti Cipher Disk." Wikipedia, http://en.wikipedia.org/wiki/Alberti_Cipher_Disk (accessed November 25, 2008).

[13] Wikipedia The Free Encyclopedia. "Bigram." Wikipedia, http://en.wikipedia.org/wiki/Bigram (accessed September 1, 2008).

[14] Wikipedia The Free Encyclopedia. "Cryptography." Wikipedia, http://en.wikipedia.org/wiki/Cryptography (accessed July 10, 2008).

[15] Wikipedia The Free Encyclopedia. "Frequency Analysis." Wikipedia, http://en.wikipedia.org/wiki/Frequency_analysis (accessed September 1, 2008).

[16] Wikipedia The Free Encyclopedia. "Image: K17 rotor 1.jpg." Wikipedia, http://commons.wikimedia.org/wiki/Image:Kl7_rotor_1.jpg (accessed July 10, 2008).

[17] Wikipedia The Free Encyclopedia. "SIGABA." Wikipedia,
http://en.wikipedia.org/wiki/SIGABA (accessed September 1, 2008).