CRYPTANALYSIS OF SIGABA

A Project Report

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 Computer Science

By

Wing On Chan

May 2007

© 2007

Wing On Chan

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Mark Stamp

Dr. John Avila

Dr. Robert Chun

APPROVED FOR THE UNIVERSITY

ABSTRACT

CRYPTANALYSIS OF SIGABA

by Wing On Chan

SIGABA is a World War II cipher machine used by the United States. Both the United States Army and the United States Navy used it for tactical communication. In this paper, we consider an attack on SIGABA using the largest practical keyspace for the machine. This attack will highlight the strengths and weaknesses of the machine, as well as provide an insight into the strength of the security provided by the cipher.

Table Of Contents

ABSTRACT	4
List of Figures	6
List of Tables	6
1. Introduction.	7
2. SIGABA Machine	8
2.1 Rotors	8
2.2 Encryption	10
2.3 Decryption	13
2.4 Physical Security & Operation Guidelines	14
2.5 Theoretical Keyspace	15
2.6 Practical Keyspace	16
3. Attacks On SIGABA	18
3.1 Previous Work	18
3.2 SIGABA Attack	18
3.3 Phase 1	19
3.4 Phase 2	23
4. Attack Refinements.	24
5. Attack Comparisons	30
6. Simulator	32
7. Conclusion	33
8. References	34
Appendix A: Rotor Permutations	35
Appendix B: Simulator Commands	36
Appendix C: Glossary	37

List of Figures

Figure 1: ECM Mark II, CSP 889/2900 [9]	7
Figure 2: ECM Mark II - Cipher/Control Rotor & Index Rotor [9]	8
Figure 3: SIGABA Cipher/Control Rotors - Normal Orientation [1]	9
Figure 4: SIGABA Cipher/Control Rotors - Reverse Orientation [1]	9
Figure 5: SIGABA Index Rotor [1]	9
Figure 6: ECM Mark II Rotor Cage [9]	
Figure 7: SIGABA Encryption [1]	
Figure 8: Control & Index Rotor ORing [1]	
Figure 9: SIGABA Decryption [1].	13
Figure 10: Type 8 Safe Locker [5]	15
Figure 11: Army Field Safe CH 76 [5]	
Figure 12: Tree Before Branch Collapsing and Pruning [1]	
Figure 13: Tree After Branch Collapsing and Pruning [1]	

List of Tables

11
12
13
22
22
25
26
26
27
27
29
29
31

1. Introduction

The ECM (Electronic Cipher Machine) Mk II is a cipher machine used by the United States (U.S.) during World War II and into the 1950s. The ECM Mk II was also known by several other names, depending on which branch of the United States military was using it. The U.S. Army called the machine the SIGABA or Converter M-134. The U.S. Navy called the machine the CSP-888/889 [6]. For this paper, we will use the Army designation of SIGABA for the machine. SIGABA was created out of the need for a better rotor cipher machine since U.S. cryptographers were aware of the susceptibility of single stepping rotor machines. William Friedman, the directory of the U.S. Army's Signals Intelligence Service, and his associate, Frank Rowlett were the ones who developed the SIGABA. Friedman developed a system to randomize rotor movement, while Rowlett came up with a way to advance rotors with other rotors. The strength of SIGABA was proven during its service lifetime, as there is no record of a successful cyptological attack on the machine. During the war, it is said that the Germans were never able to break SIGABA. It is also said that the Japanese gave up on breaking SIGABA due to the seemingly random nature of the stepping [3].



Figure 1: ECM Mark II, CSP 889/2900 [9]

2. SIGABA Machine

The SIGABA cipher machine is a rotor-based machine that uses rotating, wired rotor wheels that are removable and interchangeable. SIGABA is similar to the Enigma machine, except that SIGABA uses 15 rotors to encrypt a message compared to the Enigma's three rotors [2]. For the 15 rotors, there are three groups of rotors, five cipher rotors, five control rotors, and five index rotors. The input is from a typewriter-style keyboard and the cipher produces output on an output device, usually a paper tape. The SIGABA machine has a rotor cage that holds the 15 rotors. The cage holds three banks of rotors, with a bank for each type of rotor: cipher, control and index.

2.1 Rotors

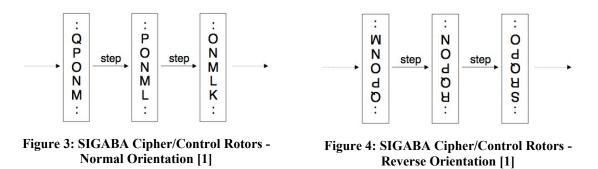
The cipher and control rotors each have 26 contacts on the two faces of the rotor, while the index rotors have 10 contacts on each face. The cipher and control rotors permute one letter of the alphabet to another letter, while the index rotors permute one digit to another digit. The rotors used for the cipher and control rotors could be interchanged to serve as either a cipher rotor or a control rotor. They could also be inserted into the rotor cage in a reversed orientation. The index rotors are inserted only in the forward orientation and are not inserted in a reversed orientation. There is nothing to prevent the rotor from being inserted in a reversed orientation. Before June 1945, the operation instructions allowed the index rotors to be inserted in reverse. However, after June 1945, when new instructions were released, the index rotors were left in the normal position [4]. For the purposes of this paper, whether this rotor is reversible is not important, as will be explained in a later section. Therefore, we will assume that the index rotors are only inserted in a forward orientation for analysis and comparison purposes.



Figure 2: ECM Mark II - Cipher/Control Rotor & Index Rotor [9]

When the cipher and control rotors are inserted in the normal orientation, the current position is shown to the user as a right side up letter that steps in reverse alphabetic order (assuming the rotor is in a position that will step). For example, in Figure 3, a rotor inserted in the normal position at letter O will have the letter P above it and the letter N below it. If the rotor steps, the current position will change to N with the letter O above it and the letter

M below it. If the rotor is inserted in the reversed orientation, the current position is shown upside down. Using the same example for the normal orientation, except with the rotor inserted in the reverse orientation (Figure 4), the user would see an upside down O as the current position, with an upside down N above it and an upside down P below it. If the rotor steps, the current position will be an upside down P with an upside down O above it and an upside down Q below it.



The index rotors are inserted in only the normal orientation (though as previously mentioned, they could be inserted in a reversed orientation), with the current position appearing as a right side up digit. However, unlike the cipher and control rotors, which appear in a decreasing order in the normal orientation, the index rotors appear in an increasing order in the normal orientation.



Figure 5: SIGABA Index Rotor [1]

The machine is initialized by inserting the rotors into their respective banks within the rotor cage. However, before the rotors can be inserted into the rotor cage, certain decisions must be made regarding the key. First, the five rotors that will be used as cipher rotors and the five rotors that will be used as control rotors must be selected from the ten available rotors. Next, a permutation of the five cipher rotors and a permutation of the five control rotors must be selected. For each of the ten rotors, an initial starting position must be picked. Once that is done, the ten rotors are inserted into the cipher rotors are inserted, the five index rotors must be selected, as well as their starting position. Index rotors can only be inserted in one orientation. Once the permutation and starting positions are chosen, the index rotors can be inserted into the

index rotor bank in the rotor cage. Once that is done, the machine is initialized with the key and is now ready to encrypt or decrypt a message.

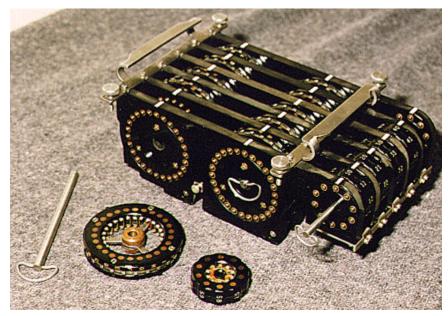
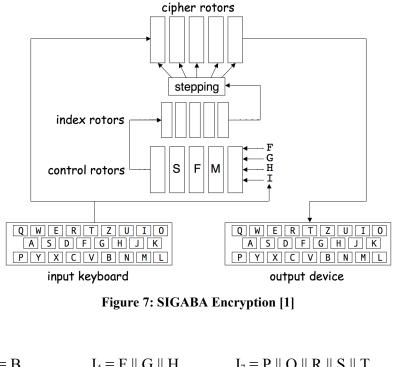


Figure 6: ECM Mark II Rotor Cage [9]

2.2 Encryption

During the encryption process, the plaintext is entered using the keyboard. When a key is pressed on the keyboard, a signal is generated that is sent to two of the three banks of rotors in the rotor cage. The first signal is sent to the left side of the cipher rotor bank. It is then permuted through the five cipher rotors to produce the ciphertext. The second signal is sent to the right side of the control rotor bank. However, the signal is handled differently from the signal sent to the cipher rotor bank.

For the control rotors, the signal is used to energize the input into the control rotor bank. Four inputs to the control rotor bank are energized when a key is pressed on the keyboard. The four inputs are always 'F', 'G', 'H', and 'I' regardless of which letter was pressed on the keyboard. These four signals are then permuted through the control rotors in a right to left fashion. Once the four signals emerge from the left side of the control rotor bank, the signals go through an ORing to determine which inputs for the index rotor bank are energized. The following table shows the index rotor bank inputs that are energized for the different control rotor outputs.



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

Table 1: Active Index Rotor Inputs

In the table, I_j is the jth input of the index rotor bank. For example, " $I_7 = P || Q || R || S || T$ " means that the seventh input to the index rotor bank is active if any of the four outputs from the control rotor bank are P, Q, R, S, or T. I_0 is never energized. Table 1 applies to the CSP-889 version of the machine only. The later CSP-2900 version operates in a different manner. In that version, the mapping of the output letters of the control rotors to active inputs of the index bank was different. In addition, instead of just 'F' 'G', 'H', and 'I' being active inputs to the control rotors, 'D' and 'E' were also activate [4].

After each letter that is keyed, one to three of the control rotors will step. Counting from the left, the fast control rotor is the third rotor in the control rotor bank, the medium control rotor is the fourth rotor in the control rotor bank, and the slow control rotor is the second rotor in the control rotor bank. The fast rotor steps once for each letter keyed into the keyboard. The medium control rotor steps once every time the fast rotor transitions from O to another letter. For the forward orientation, this would be a transition from O to N. For the reverse orientation, it would be from O to P. In [4], it is claimed that for a reversed rotor, the transition occurs at A to B rather than O to P. Our description of the transition occurring at O to P is consistent with the two simulators at [7] and [8]. The slow control rotor steps once every time the medium from O to N in the forward orientation or O to P in the reverse orientation. The first and fifth control rotors

remain fixed during operation and are not changed by the encryption process like the fast, medium, and slow control rotors.

Due to the ORing of the control rotor bank's output, one to four of the index rotor bank's inputs will be energized. The active signals are permuted by the index rotor bank in a left to right fashion. The outputs of the index rotor bank are then ORed again, though in a different manner, to determine which cipher rotor should step. The following table shows which cipher rotor will step based on the outputs of the index rotor bank.

 $\begin{array}{l} C_0 = O_0 \mid\mid O_9 \\ C_1 = O_7 \mid\mid O_8 \\ C_2 = O_5 \mid\mid O_6 \\ C_3 = O_3 \mid\mid O_4 \\ C_4 = O_1 \mid\mid O_2 \end{array}$

Table 2: Cipher Rotor Stepping Table

The table is in the format of $C_j = O_x \parallel O_y$. This means that cipher rotor j will step if the outputs of the index rotor bank contain either x or y.

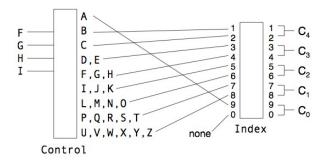


Figure 8: Control & Index Rotor ORing [1]

An interesting quirk of SIGABA's encryption algorithm is that the letter Z, and a word space are treated slightly differently than other letters. Other letters are sent to the cipher rotor bank without any modifications. However, the letter Z, and a word space are modified before being sent to the cipher rotor bank. If the letter Z is input on the keyboard, it is changed to an X before being sent to the cipher rotor bank. If a word space is input on the keyboard, it is changed to the letter Z before being sent to the cipher rotor bank.

2.3 Decryption

Decryption works in the same manner as encryption except with two changes. The machine is initialized in the same manner with the same key. However, when a key is pressed on the keyboard, a signal is sent to the right side of the cipher rotor bank instead of the left side. The second change is how the decryption of the letter Z and a word space work. When the output of the cipher rotor bank is the letter Z, it is changed to a space before being sent to the output device. Something to note here is that the decrypted plaintext will never have the letter Z in it. Any Z's in the original plaintext will be decrypted as an X. Table 3 shows what happens to the letter Z and spaces during encryption and decryption.

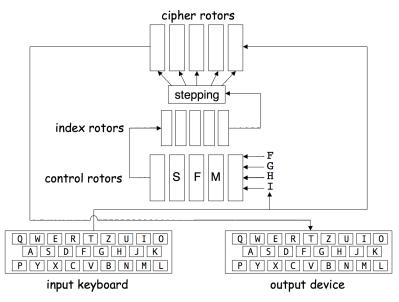


Figure 9: SIGABA Decryption [1]

Plaintext	ZERO ONE TWO THREE FOUR FIVE SIX
Ciphertext	IEQDEMOKGJEYGOKWBXAIPKRHWARZODWG
Decrypted Ciphertext	X ERO ONE TWO THREE FOUR FIVE SIX

Table 3: Encryption/Decryption Quirk

2.4 Physical Security & Operation Guidelines

A cryptographic system is only as strong as the people who operate it. The United States considered physical security important during the war, as evident by the operational guidelines and equipment available for SIGABA that was not required for the actual use of the machine. Although reliance on physical security was used during the war, today, heavy reliance on physical security of a cryptographic system could be a serious mistake. Kerckhoffs Principle states that a cryptographic system's strength should only depend on keeping the key a secret while not keeping the system's algorithm a secret. This means that an attacker is assumed to have full knowledge of how the algorithms and the system work. In other words, the system is not a black box system that obtains its strength from "security by obfuscation". During the war, the United States formally trained operators and monitored their compliance with operation procedures. When procedures were not followed, memorandums were sent to inform operators about the errors and the consequences of those errors. The following are excerpts from the memorandum in [9].

"The principles of communication security cannot be over stressed, for such security is vital to the success of operations. Errors which seem minor in themselves may, when accumulated, offer to the enemy an entering wedge for the eventual compromise of a system. The object of this memorandum is to enlist your cooperation in protecting our cipher systems and hence our national security."

"THE PRICE OF SECURITY IS ETERNAL VIGILANCE."

"CARELESS COMMUNICATIONS COST LIVES"

As for the physical security of these cipher machines, safes were often used to house them. A Type 8 Safe Locker (Figure 10) used to house the SIGABA machines weighed 172 pounds when empty [5], with the actual SIGABA machine weighing around 94 pounds [9]. A "semi-portable" field safe, the Army Field Safe CH 76 (Figure 11), was used for installation of the cipher machine at advanced bases. The total weigh of this safe, the cipher machine, and it's wooden box, was around 650 pounds, with provisions for the housing of two M1 Thermite bombs [5].

In [5], the operations manual has several sections (111 - 117) that deal specifically with destruction of the machine and any other confidential information related to the machine, such as code lists and rotors. These sections include instructions on how to remove and dispose of the wires within a rotor, how to smash the rotor wheels, where to dispose of the pieces, and even how to use the explosives in the demolition kit to destroy the machine if the need arose. One wonders how this occurred on a naval vessel that was under attack and in danger of sinking.

In addition to the safes, the machines were usually under armed guard. The Americans had strict rules about where SIGABA could be deployed. The area it would be used in had to be secure. SIGABA wasn't to be used in the field unless it was at a base where it was under constant security. The machine wasn't given to Allied nations during the war since the United States was afraid that if their strong cipher made it to the hands of the enemy somehow, that the enemy ciphers would become "invincible". The POTUS-PRIME link that is described later may be a partial exception to this.



Figure 10: Type 8 Safe Locker [5]



Figure 11: Army Field Safe CH 76 [5]

2.5 Theoretical Keyspace

First, we will discuss the theoretical keyspace of the SIGABA machine. We consider a key to include the follow.

- 1. The choice of the five cipher rotors.
- 2. The choice of the five control rotors.
- 3. The choice of the five index rotors.

Each cipher and control rotor permute the 26 letters of the alphabet. This means that each cipher and control rotor have 26! different possible permutations. Similarly, the index rotors permute the digits to another digit. This means the index rotors each had 10! different permutations. Combining these different permutations gives a theoretical keyspace of $(26!)^5 * (26!)^5 * (10!)^5 \approx 2^{993}$ different keys for the machine. We do not need to consider the starting positions of the 15 rotors in this calculation since we are considering all possible rotor wirings. A different starting position would be equivalent to another rotor wiring. For

this reason, we can treat all the possible rotor wirings as being set to some standard starting position. Since the index rotors do not step at all during the operation of the machine, the $(10!)^5$ permutations for the index rotors reduce down to 10! distinct permutations. This reduces the theoretical keyspace down to $(26!)^{10} * 10! \approx 2^{906}$.

The theoretical keyspace seems to indicate that the key is the equivalent of a modern cipher key that is 906 bits long, which is over three and a half times longer than the largest encryption key today of 256 bits. If this were true, it would certainly explain why there are no recorded instances of SIGABA ever being broken during the war by enemy forces. However, is this keyspace accurate? Unfortunately, the answer is no. SIGABA did not have a real keyspace of 906 bits. Several factors limited the actual keyspace of the machine during its operation lifetime.

2.6 Practical Keyspace

The assumptions we made when determining the theoretical keyspace of SIGABA are unrealistic. It would be impossible to make rotors for each possible wiring. It is also impossible for that much equipment to be used in the field. In reality, there were only 10 rotors available for the cipher and control rotors and 5 rotors for the index rotors. There were several sets of rotors that available for use, but for our purposes, we will consider only one set of 15 rotors. This means that there are 10! ways to permute the 10 26-letter rotors, and 2^{10} ways to orient them. For each cipher and control rotor, there are 26 possible starting positions. For each of the index rotors, there are 10 possible starting positions. This gives a practical keyspace of $10! * 2^{10} * 26^{10} * 10^5 \approx 2^{95}$ bits. Was this the actual keyspace available during the operational life of the SIGABA machine? Unfortunately, it wasn't. Two factors reduced the practical keyspace even further.

First, the cipher rotors can be set to any starting position. However, they were usually set to a standard position and stepped in a nonstandard manner, while at the same time, stepping the control rotors. This effectively reduced the keyspace by a factor of 26^5 since the starting position of the cipher rotors is constant. This means that the keyspace is now reduced to 10! $* 2^{10} * 26^5 * 10^5 \approx 2^{72}$ bits, as claimed in [10].

Another factor that further reduced the practical keyspace is that a message indicator was transmitted with the ciphertext for a message. Looking at the operation manual for SIGABA shows that the control rotors' starting positions are sent in the clear with the encrypted ciphertext message [5]. If an attacker intercepted a message and knew the meaning of the message indicator, that reduced the practical keyspace by a factor of 26^5 . With these two factors, the actual keyspace available for SIGABA during its operational lifetime would have been $10! * 2^{10} * 10^5 \approx 2^{48.4}$ bits. Today, a key of this size is vulnerable to an exhaustive key search. The Data Encryption Standard (DES) uses a 56 bit key and has been successfully attacked using an exhaustive key search. However, during World War II,

it would have been impossible to attempt an exhaustive key search unless there was a shortcut attack that could reduce the keyspace to a more manageable size for World War II era technology.

There is a variant of SIGABA used between United States President Franklin D. Roosevelt and British Prime Minister Winston Churchill during the war that was more secure called POTUS-PRIME¹ [4]. Instead of sending the control rotor settings in the clear as part of a message indicator, a codebook using three letter codewords was used instead. A codeword is also used to indicate the cipher rotor settings, in addition to the control rotor settings. These two codewords were sent with the message indicator instead. This increased the keyspace since the cipher rotors could be set independently and the control rotor settings weren't sent in the clear with the message indicator. This gave the POTUS-PRIME variant a keyspace of $10! * 2^{10} * 26^5 * 26^5 * 10^5 \approx 2^{95.4}$ bits.

In Section 2.4, we mentioned that the United States did not allow access to SIGABA, even to Allied nations. The POTUS-PRIME link seems to be a contradiction to this. However, what most likely happened was that the machine in Britain was guarded and operated by American forces. The operators would send and receive the messages and then relay the messages to Churchill and his staff without giving the British direct access to the machine.

¹ <u>**P**</u>resident <u>**O**</u>f <u>**T**</u>he <u>**U**</u>nited <u>**S**</u>tates – <u>**Prime**</u> Minister

3. Attacks On SIGABA

3.1 Previous Work

There have been two previous attempts to attack SIGABA. The first attempt is described in [4]. In this attack, John J. G. Savard and Richard S. Pekelney describe an attack that requires no known plaintext. Their attack relies only on intercepted ciphertext messages and does not rely on knowing the plaintext beforehand. For their attack, the plaintext is recovered using Kerchoffs superimposition, which is described in [10]. The attack attempts to reconstruct nearly complete cipher alphabets produced by the cipher rotors. By looking at the different alphabets that are reconstructed, they can find cases of rotor steppings. Once the wirings of the cipher rotors are reconstructed, they have the stepping motions of the cipher rotors, which they mention can be used to attack the control rotors. This attack requires a large amount of intercepted messages. The authors estimate that ten to fifteen messages sent during the same day using the same key would be needed. This is highly improbable. In their description, there was no mention of the expected work factor for this attack.

The other attack on SIGABA is from Michael Lee [10]. In Lee's attack, he first examines attacking simpler versions of SIGABA that only have one, two, and three cipher rotors. The attack on a single rotor version of SIGABA recovered the rotor wiring. In the attack on the two and three rotor versions of SIGABA, he assumes that the wiring of the rotors is known. The attack will recover the plaintext, the order of the rotors, and their initial positions. No attacks are described for a four or five rotor SIGABA machine, though the estimated time needed for attacking a four or five rotor machine using the attack on a three rotor machine are extrapolated.

3.2 SIGABA Attack

For the attack on SIGABA, we assume that all three rotor banks can be set independently, no settings are sent in a message indicator, that there are 10 rotors available for use as cipher and control rotors and 5 rotors available for the index rotors and the internal wiring of the 15 rotors is known to the attacker. Cipher and control rotors may be inserted in either the normal or reversed orientations, while the index rotors can only be inserted in the normal orientation. This gives us a keyspace of $10! * 2^{10} * 26^5 * 26^5 * 5! * 10^5 \approx 2^{102.3}$ bits. Recall that the outputs of the index rotor bank are ORed together in order to determine which of the cipher rotors will step. Instead of having $5! * 10^5$ different index rotor settings, we only have $\frac{10!}{2^5} = 113,400 \approx 2^{16.8}$ distinct index rotor settings. This reduces the keyspace down to $10! * 2^{10} * 26^5 * 26^5 * 113,400 \approx 2^{95.8}$.

The attack on SIGABA will consist of two different phases. Phase 1 tries all possible cipher rotor initial positions and determines which settings are consistent with the known plaintext/ciphertext pair. For each setting that is consistent, we will also know which rotors are used as cipher rotors and what orientation they are inserted into the machine in. Here, we will refer to the rotors used, their orientations, and their initial positions collectively as a "setting". There will be two types of settings: random and causal. Random settings are settings that survive Phase 1 but are incorrect settings for the plaintext/ciphertext pair. Since Phase 1 only considers the cipher rotors, there may be an incorrect surviving setting that is valid. However, this setting may become invalid once the control and index rotors are also examined. The causal setting is the actual setting used to encrypt the known plaintext to the known ciphertext. Phase 1 will recover all possible cipher rotor settings.

In Phase 2, we take the survivors from Phase 1 and attempt to recover the control rotor settings. In this attack, the index rotor settings are not recovered directly as a permutation of the five index rotors and their positions. The index rotor setting will be recovered as an equivalent permutation of the 10 digits. In effect, the index rotors will be recovered as a collapsed version of the five rotors.

3.3 Phase 1

During Phase 1 of the attack, we will need to select five of the ten available rotors to use as cipher rotors. There are $\binom{10}{5} = 252$ possible ways to pick five rotors. For the five selected rotors, there are 5! ways to arrange them. For each of these rotors, we can insert them in either the normal or reversed orientation. For each rotor, there are 26 possible starting positions. This gives $\binom{10}{5} * 5! * 2^5 * 26^5 \approx 2^{43.4}$ initial settings that we need to try.

For phase 1, we analyze the cipher rotor bank in isolation. For each initial setting, we pass a plaintext letter through the cipher rotors to determine the corresponding ciphertext letter. If the output of the cipher rotor bank matches the known ciphertext that we have, we attempt to recursively test the remaining letters. After the first letter is encrypted, one to four of the cipher rotors can step. This means that we need to try the 30 possible steppings of the cipher rotors and determine which, if any, of the 30 possible steppings will encrypt the next plaintext letter correctly. At each step after the initial plaintext letter, this 30 stepping test must be done.

At first glance, the testing of all initial settings seems like a fixed amount of work and is equivalent to an exhaustive key search. However, if we model the encryption permutations as being uniformly random, we get a binomial distribution where the probability of a match

 $p = \frac{1}{26}$ and n = 30. This means that for any given letter (beyond the first), we expect the

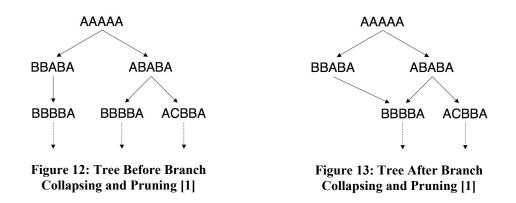
number of survivors to grow at a rate of $\frac{30}{26} \approx 1.15$ per letter. This is a property of the

machine having five rotors. In a machine that uses the same cipher but with less than five rotors, the growth rate is less than one, indicating a decrease in survivors, as the message length gets longer. Such a machine would have a much weaker cipher due the decreasing number of survivors. Attacks on machines that use the same cipher but use less than five cipher rotors are described in [10]. The paper in [10] also extrapolates the amount of time needed to attack the full five-rotor version of Sigaba. However, that paper fails to take into account the branching phenomenon. While a decrease in survivors is what we would like to see, we can still get useful information from the survivors.

The five rotor design made SIGABA more secure since it means that at any given step beyond the first letter, there are $\frac{30}{26} \approx 1.15$ surviving paths. A way to decrease the number of surviving paths in Phase 1 is to store a record of the survivors in a tree-like format and merge any branches that have the same common parent node. In Figure 12, the tree has two children for the starting position AAAAA. There are two possible steppings for the cipher rotors from the position AAAAA. For each of those two steppings, they have a valid stepping to the third letter in the message. If two paths both reach the same intermediate position at the same level of the tree, which in Figure 10 would the intermediate position of BBBBA at step 2, one of the paths can be trimmed. This is seen in Figure 11. The actual path that is trimmed does not matter since from this step on, the two paths will be identical. In Figure 11, the path from position BBABA at step 1 to BBBBA is trimmed and is merged into the path from ABABA to BBBBA. Based on this trimming and merging of paths, we can eliminate a significant number of paths if we keep track of paths we have already visited before without decreasing our chances of finding the correct setting. Since we are only concerned with the initial starting position (in this example AAAAA), we can prune the branch from BBABA and "redirect" it to the child of ABABA, as shown in Figure 13. This is partially described in [1]. In the random case, before any merging, we expect the

number of paths to increase by a factor of $\frac{30}{26} \approx 1.154$. In the causal case, we expect the

number of paths to be greater since we are guaranteed one causal match, with the remaining elements matching in the random case. This allows us to statistically distinguish between the causal case and the random cases. This distinction will also reduce the number of random cases that we must test later.



The results in Table 4 are derived using the following method. Given a known plaintext/ciphertext pair, we ran a certain number of tests on it. A random setting for the order and initial positions of the rotors was generated. Next, the first plaintext letter is encrypted before any rotors are stepped. Here, a random setting will survive the first step with a 1/26 probability. If the setting survives the first letter, the rotors can be stepped in 30 different ways. Any stepping that is consistent for the second letter is reached, then the path is valid and will be considered for Phase 2. The other option is that if at any letter, none of the 30 possible steppings for the cipher rotors yields a consistent path, that path is eliminated since it is a random path. Table 5 was generated using the same method. However, instead of a random setting, the causal setting was used.

From Table 4, we can see that as the number of letters (steps) increases, the number of surviving settings decreases. This shows that the number of random settings can be reduced using a small amount of known plaintext and ciphertext. However, we need to note that although the number of surviving settings decreases, the number of surviving paths (non-zero settings * average per non-zero) increases. For example, in Table 4, for a 30-letter message, we expect that 0.427% of the random settings (Non-Zero Settings) will survive, with each survivor expanding to an average of about 16.5 paths and a maximum of 84 paths. For the causal case, Table 5 shows that for a 30-letter message, we expect 29.6 paths with the 10,000 cases tests. We also expect a minimum of one and a maximum of 151 consistent paths.

Steps	Tests	Non-Zero Average Per		Maximum
(Letters)		Settings	Non-Zero	
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	4:	Random	Case	[1]	
abic	ч.	Random	Case	լեյ	

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

 Table 5: Causal Case [1]

From the information in Table 5, we could reduce the number of random settings by saving only those settings that meet some threshold. One example would be if the setting exceeds the expected mean in the causal case. This refinement to Phase 1 would decrease the number of random settings, but at the same time, it makes the attack probabilistic since we may end up discarding the causal case.

If we are given a small amount of known plaintext and its corresponding ciphertext, we expect the work factor to be on the order of $2^{43.4}$ since most of the random cases will not survive the first known plaintext. If we use more plaintext letters and save the merged paths, then we may exceed $2^{43.4}$ since the number of surviving paths increases as more letters are used.

Suppose we had a 100 letter known plaintext and ciphertext pair. After the first known plaintext letter, we have $\frac{2^{43.4}}{26} \approx 2^{38.7}$ surviving paths. From Table 4, in the row for 100 steps, the number of surviving merged paths increases to about

$$\frac{203*100.5}{10^5}*2^{43.4}\approx 2^{41.4}$$

This means that when 99 letters are used after the first letter, the number of merged paths increases from $2^{38.7}$ to $2^{41.1}$ and all merged paths must be processed at each step. We can approximate the number of paths at an arbitrary step k by using $2^{38.7}x^k$. For this example, we would have $2^{38.7}x^{99} = 2^{41.1}$. Solving this equation gives us $x \approx 1.017$. Using

$$\sum_{i=m}^{n} x^{i} = \frac{x^{n+1} - x^{m}}{x - 1}$$

the primary work can be calculated by

$$2^{43.3} + \sum_{0}^{99} 2^{38.7} * 1.017^{k} = 2^{43.4} + 2^{38.7} \left(\frac{1.017^{100} - 1}{0.017} \right) \approx 2^{46.7}$$

For 100 plaintext letters, we see that only about $2^{41.1}$ merged paths will survive Phase 1. From Table 4, about $\frac{203 * 100.5}{10^5 * 100} * 2^{43.4} \approx 2^{34.5}$ random settings survive.

At the end of Phase 1, we will have a list of all initial cipher rotor settings that can possibly be the part of the correct key. However, there will also be settings included in this list that are not valid since it may not be possible to step the cipher rotors in the same manner as the steppings of Phase 1.

3.4 Phase 2

In Phase 2, we attempt to recover the control rotor settings. We are assuming that no message indicator was used here and that the control rotor settings were set independently. Since we already used five rotors for the cipher rotors in Phase 1, we only need to determine the order of the remaining five rotors and their orientations. After determining how the rotors are inserted, we need to determine the starting position of the control rotors. This gives us $5! * 2^5 * 26^5 \approx 2^{35.41}$ settings to try. As states previously mentioned, the actual number of index rotor permutations is only $113,400 \approx 2^{16.8}$. Combining the control and index rotors, we have about $5! * 2^5 * 26^5 * 113,400 \approx 2^{52.2}$ different settings to try. We test

the survivors of Phase 1 by setting the cipher rotors to the setting used by the survivor, then testing each of the $2^{52.2}$ settings for the control and index rotors. Any survivors of Phase 2 are valid keys for the known plaintext/ciphertext pair. Here, it appears that for each surviving setting from Phase 1, we have a work factor on the order of $2^{52.2}$. Fortunately, we can improve on this rather naïve implementation of Phase 2.

4. Attack Refinements

The attack described in Sections 3.3 and 3.4 is more or less an exhaustive key search. While an exhaustive key search for the SIGABA keyspace as it was used during the war ($\approx 2^{48.4}$ bits) is possible given the speed and power of today's computers, an exhaustive key search for the practical keyspace is still not feasible, even with today's computers.² Here we will examine possible ways to reduce the keyspace in both Phase 1 and Phase 2 of the attack.

For Phase 2, we will look at an improvement that is also partially described in [1]. We will examine the frequency of the active outputs of the control rotor bank to the index rotor bank. Recall from Table 1 that the inputs for the index rotor bank are energized by a variable number of outputs from the control rotor bank. Input 8 of the index rotor bank is energized by six outputs of the control rotor bank, but inputs 9, 1, and 2 are each only energized by one output of the control rotor bank. With the frequency of the stepping of the cipher rotors from Phase 1's survivors, we can estimate the probabilities for the index rotors' permutation.

In Figure 8, we show how the control and index rotors interact. In that figure, we have collapsed both banks of rotors into one equivalent rotor. The control rotors receive four energized inputs for F, G, H, and I. The four inputs are passed through the control rotor and combined according to the rules in Table 1 before being sent to the index rotors. The one to four active index rotor inputs are combined according to the rules in Table 2. Since the control rotors permutation changes with each letter, we model the collapsed version of the

control rotors as being uniformly random. This means that we assume all of the $\begin{pmatrix} 26\\4 \end{pmatrix}$ =

14,950 combinations of outputs from the control rotors are equally likely. However, since the outputs of the control rotors are ORed together according to Table 1, the inputs to the index rotors are not uniform. Input 8 on the index rotors will be active more than inputs 1, 2, or 9 since input 8 is activated by 6 letter, whereas inputs 1, 2, and 9 are activated by only one letter.

The index rotor outputs are ORed together according to the rules in Table 2 to determine which cipher rotors will step. If we can determine the frequency with which the cipher

² The 56 bit key for the Data Encryption Standard (DES) has been successfully attacked using an exhaustive key search. [2]

rotors step, we can assign probabilities to the index permutation. For each of the surviving paths from Phase 1, we have a list of the cipher rotor steppings. By using this list, we can determine how many times each cipher rotor steps. For the merged paths from Phase 1, we do not lose any information. In Figure 11, we have the initial position of AAAAA. There are two paths from AAAAA to the second letter, BBABA and ABABA. At the third letter, both of those paths merged to BBBBA. Since both paths reached BBBBA, we know that on both paths, cipher rotors C_0 , C_1 , C_2 , and C_3 stepped once while cipher rotor C_4 did not step.

Let us consider an index permutation of (5, 4, 7, 9, 3, 8, 1, 0, 2, 6). This permutation indicates that an input of 0 maps to an output of 5, an input of 1 maps to an output of 4, and so forth. If we consider the pairs of outputs that will determine the cipher rotor steppings, we see that some rotors will step more often than others. In Table 6, inputs 6 and 8 map to outputs of 1 and 2 respectively. The inputs of 6 and 8 correspond to the outputs of 6 and 8 from the control rotors. At least one of these outputs from the control rotors will be active if at least one of the following letters are the active output of the four signals though the control rotors according to Figure 8: L, M, N, O, U, V, W, X, Y, Z. The same method can be used to determine the letters that control the other four cipher rotors. From the counts, we can see that for this index permutation, cipher rotor C_0 will step more than all the other cipher rotors.

		Cipher Rotor				
\mathbf{C}_0 \mathbf{C}_1 \mathbf{C}_2 \mathbf{C}_3 \mathbf{C}_3						
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	

Table 6: Index Permutation (5, 4, 7, 9, 3, 8, 1, 0, 2, 6) [1]

If we assume that the control rotors generate random permutations, the expected number of steps for a given cipher rotor i depends only on the number control rotor output letters that feed into cipher rotor C_i . The list of all 45 input pairs and their corresponding number of letters is show in Table 7. If a sufficient amount of plaintext is known, we can obtain information related to the Count column of Table 7 for each cipher rotor based on a count of the number of times that cipher rotor i has stepped. From this count, we can make restrictions on the index permutation using the Pairs column.

We can estimate the amount of plaintext that is needed in the following manner. Each cipher rotor is connected to k control rotor outputs, where $1 \le k \le 11$. We can determine the expected stepping ratios for a rotor connected to exactly k control rotor outputs. These will sum up to a value greater than one since more than one rotor generally steps. To compute

the ratios, we assume all control rotor outputs to be equally likely and generate all $\begin{pmatrix} 26\\4 \end{pmatrix} =$

14,950 outputs, counting the number of times that at least one element of each pair in Table 7 occurs. Table 8 shows the results, where Step Ratio is obtained by dividing the Step Count column by 14,950. Note that these results are independent of the actual index rotor permutation.

Letters	Count	Pairs		
1	3	(0,1)(0,2)	2) (0,9)	
2	4	(0,3)(1,2)	2) (1,9) (2,9)	
2 3 4	5	(0,4)(0,5)	5) (1,3) (2,3) (3	3,9)
4	7	(0,6) (1,5	5) (2,5) (5,9) (1,4) (2,4) (4,9)
5	6	(0,7) (1,6	6) (2,6) (6,9) (3,4) (3,5)
6	6	(0,8)(1,7)	7) (2,7) (7,9) (3	3,6) (4,5)
7	6	(1,8) (2,8	8) (8,9) (3,7) (4	4,6) (5,6)
8	3	(3,8) (4,7	7) (5,7)	
9	3	(4,8) (5,8	8) (6,7)	
10	1	(6,8)		
11	1	(7,8)		
Ta	able 7: Iı	ndex Perm	utation Input	Pairs [1]
Letter	s Exar	nple Pairs	Step Count	Step Ratio
1		(0,1)	2,300	0.1538462
2		(0,3)	4,324	0.2892308
3		(0,4)	6,095	0.4076923
4		(0,6)	7,635	0.5107023
5		(0,7)	8,965	0.5996656
6		(0,8)	10,105	0.6759197
7		(1,8)	11,074	0.7407358
8		(3,8)		
9		(4,8)	12,570	0.8408027
10		(6,8)	13,130	0.8782609
11		(7,8)	13,585	0.9086957

 Table 8: Cipher Rotor Stepping Ratios [1]

Now given the cipher rotor stepping counts from Phase 1, we can use Table 8 to determine the most likely pairs of control rotor output letters for each cipher rotor. Since these are connected to the index permutation, this information combined with the information from Table 7 will reduce the number of possible index permutations.

A valid index permutation must contain five pairs from Table 7 such that two conditions are met. The first condition is that each digit is used once and only once. The second condition is that the number of letters for the five pairs must sum up to exactly 26. If we order all the valid sets of five pairs that have letters that sum to 26 and use each digit only once, we find that there are 2148 groupings.

Now that we have the stepping counts from Phase 1, we can calculate the stepping ratios for each cipher rotor. The stepping ratio is just the step count divided by the number of letters. Once we have the ratios computed, we can attempt to distinguish the number of letters connected to each cipher rotor in the index permutation.

Consider an example where the cipher rotors had the following stepping ratios.

Cipher Rotor	Stepping Ratio
C_0	0.15
C_1	0.29
C_2	0.60
C_3	0.74
C_4	0.91

Table 9: Example Stepping Ratios [1]

Using the results from Table 8, the most likely number of letters connected to cipher rotors C_0 , C_1 , C_2 , C_3 , C_4 are 1, 2, 5, 7, and 11 respectively. Of the 2148 valid combination of pairs from Table 7, there are six sets of pairs that are consistent with the number of letters derived from Table 8. These six sets of pairs are shown in Table 10.

Set	Pairs
1	(0,1) (2,9) (3,4) (5,6) (7,8)
2	(0,1)(2,9)(3,5)(4,6)(7,8)
3	(0,2)(1,9)(3,4)(5,6)(7,8)
4	(0,2)(1,9)(3,5)(4,6)(7,8)
5	(0,9)(1,2)(3,4)(5,6)(7,8)
6	$\begin{array}{c} (0,1) (2,9) (3,4) (5,6) (7,8) \\ (0,1) (2,9) (3,5) (4,6) (7,8) \\ (0,2) (1,9) (3,4) (5,6) (7,8) \\ (0,2) (1,9) (3,5) (4,6) (7,8) \\ (0,9) (1,2) (3,4) (5,6) (7,8) \\ (0,9) (1,2) (3,5) (4,6) (7,8) \end{array}$

Table 10: Sets Of Pairs Consistent with Letter Counts 1, 2, 5, 7, and 11 [1]

According to [1], the 2148 consistent groupings of five pairs can be reduced to 89 distinct categories based on letter counts. Using the previous example, one of the 89 categories would be one that corresponded to letter counts of 1, 2, 5, 7, and 11 letters. On average, there are 24 sets of five pairs for each category, with the actual range being between 3 and 72.

With the stepping counts computed, we can now compute a score to determine which of the 89 categories is the best match. Let x_i be the step ratio in row i for Table 8. Then $1 \le i \le 11$ and x_i is the expected fraction of the time that a cipher rotor steps when it is connected to i letters. For convenience, let $x_0 = 0$ and $x_{12} = 1$.

For the known plaintext, we compute the five stepping ratios s_0 , s_1 , s_2 , s_3 , and s_4 where we assume $s_0 < s_1 < s_2 < s_3 < s_4$. For each s_j stepping ratio, we determine the index i for which $x_i \le s_j < x_{i+1}$. Then we let $t_j = \frac{s_j - x_i}{x_{i+1} - x_i}$ with the provisions that if $t_j < 1$, we set $t_j = 1$ and if $t_i > 11$, we set $t_i = 11$. Here we note that $i \le t_j \le i + 1$ and $t_0 < t_1 < t_2 < t_3 < t_4$. Each t_j is a decimal representation (including the fractional part) of the most likely number of letters connected with cipher rotor j. We are using a linear interpolation for the points between consecutive x_i values since the x_i values are not equally spaced.

Next, we let $(u_0, u_1, u_2, u_3, u_4)$ be one of the 89 categories that was discussed earlier where $u_0 < u_1 < u_2 < u_3 < u_4$. We compute the score as the square of the Euclidean distance. For each category, we compute $d = (t_0 - u_0)^2 + (t_1 - u_1)^2 + (t_2 - u_2)^2 + (t_3 - u_3)^2 + (t_4 - u_4)^2$. The category that has the smallest distance d from $(t_0, t_1, t_2, t_3, t_4)$ is selected as the most likely category.

Table 11 shows some empirical results about how many letters of known plaintext we would need for the secondary phase of the attack using the scoring method discussed. In the table, we can see that for 100 known plaintext letters, we have a 0.2332 probability of having all five pairs correct. If we consider the case where two of the pairs are off by +1 letter and -1 letter, we get the percentage in the last column of the table. For 100 letters, we would have a 0.8216 probability that the pairs are either all correct, or off by +1/-1. Since there are 24 sets of pairs on average for a category, we would need to test less than 2^8 sets

of pairs on average $(24 * \begin{pmatrix} 5 \\ 2 \end{pmatrix} = 24 * 10 = 240 < 2^8)$ to get the correct index permutation

with a 0.8216 probability of success.

We have more information available to us from Table 11. If at any point, only one of the five cipher rotors step, then we can eliminate rows 1, 2, and 3 from Table 7 since the control rotor permutation always have four active outputs. From Table 12, we see that a single rotor stepping is a relatively rare occurrence. It only occurs about 2.5% of the time. When it does occur, it can eliminate up to ¹/₄ of the possible index permutations. Using such refinements, we expect to be able to reduce the plaintext requirements from Table 11 without decreasing out probability of success. However, since we merged paths in the primary phase, it may be difficult to utilize this information.

Plaintext	0	1	2	3	4	5	Iterations	Probability
Letters								(+/- 1 Letter)
50	0.0287	0.2213	0.1837	0.4756	0.0000	0.091	1000000	0.5666
100	0.0036	0.1076	0.0672	0.5884	0.0000	0.2332	1000000	0.8216
150	0.0006	0.0517	0.0234	0.5522	0.0000	0.3721	1000000	0.9243
200	0.0001	0.0253	0.0085	0.4722	0.0000	0.4939	1000000	0.9661
250	0.0000	0.0128	0.0033	0.3900	0.0000	0.5939	1000000	0.9839
300	0.0000	0.0064	0.0013	0.3153	0.0000	0.6769	1000000	0.9922
400	0.0000	0.0018	0.0002	0.2023	0.0000	0.7957	1000000	0.9980
500	0.0000	0.0005	0.0001	0.1300	0.0000	0.8694	1000000	0.9994
1000	0.0000	0.0000	0.0000	0.0157	0.0000	0.9843	1000000	1.0000

Table 11: Secondary Known Plaintext [1]	able 11: Secondary Known Pl	aintext [1]
---	-----------------------------	-------------

To summarize, by using the cipher rotor stepping counts, we can reduce the index permutations to a fraction of the $\frac{10!}{32} \approx 2^{16.8}$ that would need to be considered. With enough known plaintext, we can reduce to around 2^8 permutations, and possibly even less.

Assuming we had a message with 100 known plaintext letters, the average work factor for Phase 2 would be around $2^8 * 5! * 2^5 * 26^5 \approx 2^{43.3}$. It should be possible to further reduce the factor of 2^8 . This is a significant improvement over the naïve implementation of Phase 2's work factor of $2^{52.2}$. These refinements to Phase 2 also make the work factor comparable to the work of Phase 1. However, the work factor applies to each survivor of Phase 1. To improve the overall performance of the attack, we need to reduce either the number of survivors from Phase 1 and/or make Phase 2 more efficient.

The following is a method to use the information from the cipher rotor stepping counts obtained in Phase 1. For each distinct index permutation, we can compute the probabilities p_i for i = 1, 2, 3, and 4 that exactly i cipher rotors step, where the probabilities are computed over all possible control rotor outputs. Recall that we are modeling each of the four letter control rotor outputs as being equally likely. The average, maximum and minimum over all the index permutations appear in Table 12.

Rotors	Average	Maximum	Minimum
That Step			
1	0.0109	0.0247	0.0027
2	0.2543	0.3579	0.1694
3	0.5669	0.5954	0.5177
4	0.1679	0.2368	0.0996

Table 12: Cipher Rotor Steppings

An interesting thing to notice about the results from Table 12 is that the range of the values is small and only overlap when 2 and 4 rotors step. The ranges for 1 and 3 rotors stepping do not overlap. Consequently, this means that we assign a score to each survivor of Phase 1 without making any assumptions about the index permutation. For each survivor, we compute a score based on the number of cipher rotors that stepped for each known plaintext letter. Then, in the secondary phase, we can test the highest scoring survivor, then the second highest, and so on. This method would trim away unlikely paths in Phase 1, reducing the number of survivors that are sent to Phase 2. The merging of paths in Phase 1 creates a slight complication since different number of rotors can be stepped to reach the merge position. A solution to this would be to take the maximum probability of the paths that merge.

Assuming we have sufficient known plaintext, we have shown that the work factor for the secondary phase for each survivor of Phase 1 is around 2⁴³. This amount of work is feasible for today's technology, although the actual attack is not trivial to implement. The primary phase has a similar work factor and is feasible for today's technology. However, in the attack described here, the number of survivors from Phase 1 is large, which makes cost of the attack quite high.

5. Attack Comparisons

It would be beneficial to compare the work factors related to attacking SIGABA. To remain consistent, we will split each attack into a primary and secondary phase. We first discussed a straightforward exhaustive key search. The primary phase for an exhaustive key search has $2^{43.4}$ different cipher rotor settings to test. For a straightforward secondary phase, as described in Section 3.4, there were $2^{52.5}$ different control and index rotor settings to test for each of settings from the primary phase. This gives a maximum work factor of $2^{95.6}$ and an expected work factor of $2^{94.6}$ to find the correct setting. This attack has a success rate of one since all settings are tested.

Now let us consider an attack that uses one known plaintext letter. The primary phase for this attack would again consist of trying all the cipher rotor settings, yielding a work factor of $2^{43.4}$. However, since we have one known plaintext letter, we expect only 1/26 of the

settings to survive. This gives us an estimate of $\frac{2^{43.4}}{26} \approx 2^{38.7}$ survivors for the primary phase.

The secondary phase requires trying all control and index rotor settings, which like the exhaustive key search described in the last paragraph, has a work factor of $2^{52.2}$. This gives a total work factor of $2^{90.9}$. Again, we expect to find the correct setting after a work factor of $2^{89.9}$ with a probability of success of one.

Next, we consider an attack that uses 100 plaintext letters. According to the primary phase described in Section 3.3, we expect $\frac{203 * 100.5}{10^5 * 100} * 2^{43.4} \approx 2^{34.5}$ survivors from Phase 1. Using a straightforward Phase 2, as described in Section 3.4, which has a work factor of $2^{52.2}$, this attack would have a maximum work factor of $2^{86.7}$ and an expected work factor of $2^{85.7}$ before we find the correct setting. Again, the probability of success in this attack is one.

Finally, we consider an attack using the primary phase described in Section 3.3 and the refined secondary phase from Section 4 with 100 known plaintext letters. In the secondary phase, we must test the surviving merged paths instead of the settings. From Section 3.3, we see that the number of merged paths grows to about $2^{41.1}$ paths. From Section 4, we know that the work for the secondary phase with 100 known plaintext letters is about $2^{43.4}$. This gives a total work factor of around $2^{84.5}$, with an expected work factor of $2^{83.5}$. The probability for success in this attack is only 0.82 (see Table 11).

While the last attack described in this section is a modest improvement over a straightforward secondary phase and is now only probabilistic with regards to success, there are refinements that can be made to reduce the work factor. Trimming of paths with a low probability in Phase 1 is one such refinement.

The different attacks mentioned above are summarized in Table 13. This table shows that while our attack on SIGABA is far from being practical, it is more efficient than the obvious attacks on the full keyspace of SIGABA. However, for our attack to succeed, we must have a favorable amount of known plaintext. Our attack is also probabilistic, though with enough known plaintext, we have a fairly high probability of success. This can been seen as additional evidence as to why SIGABA was never broken during World War II.

Attack	Primary	Secondary	Total	Probability
	Survivors	Work	Work	Of Success
Exhaustive Key Search	243.4	2 ^{52.2}	$2^{95.6}$	1.00
1 Known Plaintext	2 ^{38.7}	$2^{52.2}$	$2^{90.6}$	1.00
100 Known Plaintexts	2 ^{34.5}	2 ^{52.2}	$2^{86.7}$	1.00
100 Known Plaintexts	241.1	$2^{43.4}$	$2^{84.5}$	0.82

Table 13: Attack Comparisons [1]

6. Simulator

For this paper, a simulation of the SIGABA cipher machine was needed in order to implement and test the different parts of the attack. A simulator has been coded that we believe closely matches the behavior of the online simulator written by Richard Pekelney at [8], which appears to be the standard SIGABA simulator. There is Windows-based simulator with a better graphical user interface at [7]. The Windows-based SIGABA simulator is also based on the simulator written by Richard Pekelney. Our simulator does not contain as many features as the simulator online but the encryption and decryption algorithm matches the behavior of the online simulator in CSP-889 mode.

All the attacks described in Section 3 & 4 were tested using the simulator we have written since we needed to have a simulator where we have control over the different sections of code that represent the encryption and decryption algorithm. Another reason why we needed our own simulator is that the attack's execution efficiency is an important factor. The execution efficiency of a Java program is low, so we had to write our simulator in C, which has higher execution efficiency. For our simulator, we duplicated the rotor wiring from the Java simulator. However, it should be noted that of those wirings, only the index rotor wirings are actual rotor wirings. Richard Pekelney made up the wiring for the control and cipher rotors since the rotors he had access to were straight-pass-through rotors only.

In our simulator, the rotors are an array of offsets from their respective letters. Suppose we had a cipher rotor that had the following offsets.

24 1 5 8 12 13 14 25 19 20 24 12 1 12 22 15 1 24 3 16 25 5 0 8 16 13

This means that 'A' is offset by 24 letters, 'B' is offset by 1 letter, 'C' by 5 letters, and so forth, where 'A' is considered position 0, 'B' position 1, and so forth. This means that this particular rotor has the following permutation.

YCHLQSUGBDIXNZKERPVJTAWFOM

The offsets for the index rotors are used in the same manner. Suppose we had an index rotor with the following offsets: 7 4 7 8 0 3 6 9 5 1. This means that 0 maps to 7, 1 maps to 5, 2 maps to 9, and so forth. The actual rotor permutation would be 7591482630. A list of all the rotor permutations used is included in Appendix A.

The source code for the simulator and the source code for the attack are included on the enclosed CD-ROM disc.

7. Conclusion

As generally used during World War II, SIGABA had a keyspace of size 2^{48.4}, which means the expected work for an exhaustive key search was 2^{47.4}. However, the POTUS-PRIME link between President Roosevelt and Prime Minister Churchill used the full keyspace of over 95 bits. For a designer of a cryptographic system, there is no reason to have a keyspace larger than a known shortcut attack on the system since a larger keyspace entails more settings and more chance for errors. During the war, the work factor for an exhaustive key search would have been impossible to do. Since SIGABA was used for strategically important tactical information, like the POTUS-PRIME link between President Roosevelt and Prime Minister Churchill, a larger keyspace may have been desired if it provided a greater amount of security. In this paper, we describe an attack on the full keyspace of SIGABA that requires less than 95 bits of work. The attack described here can certainly be improved so that it runs more efficiently. By making the attack more efficient, the number of survivors from Phase 1 and the number of cases that need to be tested in Phase 2 for each survivor of Phase 1 is reduced. This reduction will lead to a more practical attack.

SIGABA's large keyspace certainly played a role in ensuring that enemy forces never broke it. The designers of SIGABA were obviously aware of how important the seemingly random stepping of the machine was to the security of the cipher after having studied and broken other rotor-based cipher machines. Although they may not have looked at the strength of the cipher in terms of bits, they surely knew that their design would make an attack infeasible during World War II. Commanders of the Army and Navy were also aware of how important it was to physically guard the machines from capture and the detrimental effects of operational error. These two factors combined made SIGABA very hard to attack since it appeared to step randomly and physical security on the machine was very high. While the physical security of the machine was important since it made an attack harder, it is not something that should be depended on. Assuming an attacker could get hold of the machine and the rotor wirings, the cipher would still be secure. Even after more than fifty years, the analysis of attacks on SIGABA seem to indicate that having the machine and the rotor wirings would not be that helpful in breaking the cipher. This would indicate that during the war, if enemy forces had obtained the machine and rotors, they would still not be able to compromise the cipher.

8. References

- [1] M. Stamp and W. O. Chan, *Analysis of SIGABA*, Department of Computer Science, San Jose State University, April 2007
- [2] M. Stamp and R. M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Wiley Interscience, 2007
- [3] National Cryptologic Museum, The 'Big' Machines Exhibit, at http://www.nsa.gov/museum/museu00002.cfm
- [4] J. J. G. Savard and R. S. Pekelney, The ECM Mark II: Design, History, and Cryptology, *Cryptologia*, Vol. 23, No. 3, July 1999, pp. 211-228
- [5] R. Pekelney, ECM MARK 2 and CCM MARK 1, at http://www.hnsa.org/doc/crypto/ecm/
- [6] SIGABA Wikipedia, the free encyclopedia, at http://en.wikipedia.org/wiki/SIGABA
- [7] Frode Weierud's CryptoCellar | ECM MK II Cipher Machine, at http://frode.home.cern.ch/frode/crypto/simula/SIGABA/index.html
- [8] USS Pampanito CSP-889/2900 Emulation, at http://www.maritime.org/ecmapp.htm
- [9] R. Pekelney, USS Pampanito ECM Mark II, at <u>http://www.maritime.org/ecm2.htm</u>
- [10] M. Lee, Cryptanalysis of the SIGABA, University of California Santa Barbara, at http://ucsb.curby.net/broadcast/thesis/thesis.pdf

Appendix A: Rotor Permutations

Cipher Rotor 0	YCHLQSUGBDIXNZKERPVJTAWFOM
Cipher Rotor 1	INPXBWETGUYSAOCHVLDMQKZJFR
Cipher Rotor 2	WNDRIOZPTAXHFJYQBMSVEKUCGL
Cipher Rotor 3	TZGHOBKRVUXLQDMPNFWCJYEIAS
Cipher Rotor 4	YWTAHRQJVLCEXUNGBIPZMSDFOK
Control Rotor 0	QSLRBTEKOGAICFWYVMHJNXZUDP
Control Rotor 1	CHJDQIGNBSAKVTUOXFWLEPRMZY
Control Rotor 2	CDFAJXTIMNBEQHSUGRYLWZKVPO
Control Rotor 3	XHFESZDNRBCGKQIJLTVMUOYAPW
Control Rotor 4	EZJQXMOGYTCSFRIUPVNADLHWBK
Index Rotor 1	7591482630
Index Rotor 2	3810592764
Index Rotor 3	4086153297
Index Rotor 4	3980526174
Index Rotor 5	6497135280

Appendix B: Simulator Commands

Set of commands for the simulation include:

!quit	Quit program
1 !q	
!reset	Reload configuration and reset rotors
!r	
!encrypt	Switch to encryption mode [default mode]
!e	
!decrypt	Switch to decryption mode
!d	
!reverse	Reverse a rotor
!rev	
!encryptfromfile !	Encrypt using the plaintext from a file
eff	
!printConf	Print the rotor permutations
!pc	
!printPos	Print the rotor positions
!pp	
!printoffsets	Print the rotor offsets
!po	
!set	Set a rotor
!s	
!setpositions	Set the positions of the rotors. String of 15 characters. 0-9 must be
!sp	letters from A-Z and 10-15 must be digits 0-9
!setrotors	Set the rotors to use and their order. String of 15 characters. 0-9 must
!sr	be a permutation of 0-9 and 10-14 must be a permutation of 1-5

Appendix C: Glossary

Cipher Rotor	Rotor that permutes letters to letters. Interchangeable with the control rotors. Reversible.
Control	
Control	Rotor that permutes letters to letters. Interchangeable with the cipher rotors.
Rotor	Reversible.
CSP	Code and Signal Publication
ECM	Electronic Cipher Machine
Index Rotor	Rotor that permutes digits to digits
Key	Collection of settings used to initialize the machine. This includes:
	- The five rotors to be used as cipher rotors, their ordering, their initial
	positions, and their orientations.
	- The five rotors to be used as control rotors, their ordering, their
	initial positions, and their orientations.
	- The ordering of the five index rotors and their initial positions.
Path	An initial setting for the cipher rotors along with the stepping pattern of the
	cipher rotors that lead to the correct ciphertext
POTUS-	President of The United States – Prime Minister
PRIME	
Rotor	A mechanical wheel that permutes a set on inputs to a set of outputs.
Rotor Bank	A set of five rotors that are used for the same function.
Rotor Cage	Holds the three rotor banks: cipher rotor bank, control rotor bank, and index
	rotor bank.
Setting	Initial positions, and orientations of rotors