SIGABA: Cryptanalysis of the Full Keyspace

Mark Stamp and Wing On Chan
Department of Computer Science
San Jose State University
San Jose, California

Abstract

In this paper we consider an attack on the SIGABA cipher under the assumption that the largest practical keyspace is used. The attack highlights various strengths and weaknesses of SIGABA and provides insight into the inherent level of security provided by the cipher.

Keywords: SIGABA, cryptanalysis, CSP–889, Converter M–134C, ECM Mark II

1 Introduction

Remove the Cipher Unit from the machine,
withdraw the Index Maze Spindle and remove the Index Wheels.
Destroy the Index Wheels by smashing them with a heavy hammer.
— SIGABA operating instructions [6]

The SIGABA cipher was developed by American cryptographers—including Friedman and Rowlett—prior to World War II.[1] As far as is known, no successful attack on SIGABA was ever conducted during its service lifetime. During WWII, the Germans are said to have quit collecting SIGABA intercepts since they deemed the problem hopeless [1].

In this paper we first give a reasonably complete description of the cipher. Then we consider the size of the SIGABA keyspace in some detail, followed by an outline of an attack on the machine, under the assumption that the maximum practical WWII keyspace is used. The attack aims to address the inherent level of security provided by the SIGABA design. This attack also highlights the crucial features of the SIGABA that made it so much more secure than other WWII era cipher machines such as the German Enigma or the Japanese Purple.

[1] Rowlett cited the design of SIGABA as his proudest accomplishment, not the breaking of Purple as might have been expected [1].

2 SIGABA Cipher Machine

There were several variants of the basic SIGABA design, and to further muddy the water, different branches of the military used different names for the same machine. The SIGABA machine in Figure 1 is thought to be equivalent to the CSP–889 (used by the Navy) and the Converter M–134C or SIGABA (different names, but the same device, used by the Army). In addition, the name ECM Mark II was used during the development of the machine that would become SIGABA. Here, we stick with the name SIGABA.

Figure 1: A SIGABA Machine [5]

The SIGABA cipher includes a typewriter keyboard for entering the plaintext (or ciphertext), and an output device for printing the corresponding ciphertext (or plaintext). Like the well-known German Enigma cipher, SIGABA is a rotor machine, but there are several important differences between the two. Cryptographically, the most significant differences are that whereas Enigma uses three rotors, SIGABA employs five rotors to permute the letters, and whereas Enigma rotors step like an odometer, the SIGABA cipher rotor motion is controlled by a set of ten additional rotors, for a total of 15 rotors. In effect, it is as if the motion of the SIGABA encryption rotors is controlled by another rotor cipher machine. This causes the SIGABA rotors to step irregularly, which is a major improvement over the Enigma and other regularly-stepping rotor machines. SIGABA also lacks the reflector and stecker (plugboard) that are found in the Enigma.

The fifteen SIGABA rotors consist of five *cipher rotors*, five *control rotors* and five *index rotors*, where the cipher rotors permute the input letters and the other two banks of rotors drive the cipher rotors. The cipher and control rotors are interchangeable, and these rotors are also designed so they can be inserted backwards. The cipher and control rotors each permute the 26 letters. The five index rotors each permute the ten digits $0, 1, 2, \ldots, 9$ and, of course, the index rotors are not interchangeable with the other rotors. Apparently, the index rotors could also operate in the reverse orientation [9], but we ignore this feature below, since it does not affect our analysis of the cipher. Figure 3 illustrates the cryptographic components of SIGABA in encryption mode.

After a letter is encrypted or decrypted, from one to four of the cipher rotors step. The number and selection of the stepping cipher rotors is controlled by the other two

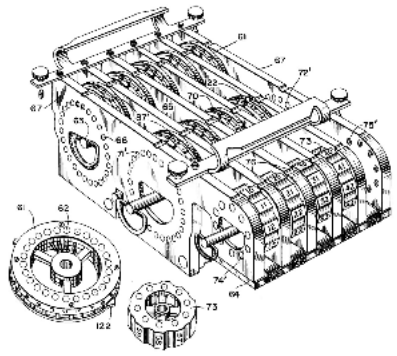

Figure 2: SIGABA Rotors [8]

banks of rotors, that is, the control and index rotors.

For each letter typed, the rightmost control rotor receives four simultaneously energized inputs, which we assume to be `F`, `G`, `H` and `I`. These four letters are permuted according to the five control rotors and the resulting four permutation-dependent output letters are combined before being input to the index rotor bank. Let I_j denote the input to element j of the leftmost index rotor and `A` through `Z` the outputs of the control rotor bank. Then

$$\begin{array}{lll} I_1 = \texttt{B} & I_4 = \texttt{F} \vee \texttt{G} \vee \texttt{H} & I_7 = \texttt{P} \vee \texttt{Q} \vee \texttt{R} \vee \texttt{S} \vee \texttt{T} \\ I_2 = \texttt{C} & I_5 = \texttt{I} \vee \texttt{J} \vee \texttt{K} & I_8 = \texttt{U} \vee \texttt{V} \vee \texttt{W} \vee \texttt{X} \vee \texttt{Y} \vee \texttt{Z} \\ I_3 = \texttt{D} \vee \texttt{E} & I_6 = \texttt{L} \vee \texttt{M} \vee \texttt{N} \vee \texttt{O} & I_9 = \texttt{A} \end{array} \tag{1}$$

where (1) is interpreted to mean that, for example, input 3 of the leftmost index rotor is active if output `D` or `E` (or both) results from the control rotors; otherwise input 3 is inactive. Note that I_0 is missing, which implies that input 0 is always inactive. Since four values are input to the control rotors, due to the "OR" of the outputs, anywhere from one to four of the inputs to the index rotors are active at each step.

The middle three control rotors step in an odometer-like fashion—almost. The fast, medium, and slow control rotors are indicated by F, M and S, respectively, in Figure 3, where the fast rotor steps with each letter, the medium rotor steps once for each 26 steps of the fast rotor, and the slow rotor steps once for each 26 steps of the medium rotor. The stepping of these three rotors differs from an odometer only in the order of the fast, medium and slow rotors. The initial setting of all five control rotors is adjustable, but the leftmost and rightmost control rotors do not step during encryption or decryption.

The output of the control rotor bank enters the index rotor bank. The index rotors do not step, but their order and initial positions are adjustable. For a particular message, the index rotors effectively implement a simple substitution of $0, 1, \ldots, 9$ (i.e., a fixed permutation of $0, 1, \ldots, 9$). From one to four (inclusive) of the inputs to the index rotor bank are active, and the number of active outputs is equal to the number of active inputs.

It is important to emphasize that this description—and, consequently, the attack

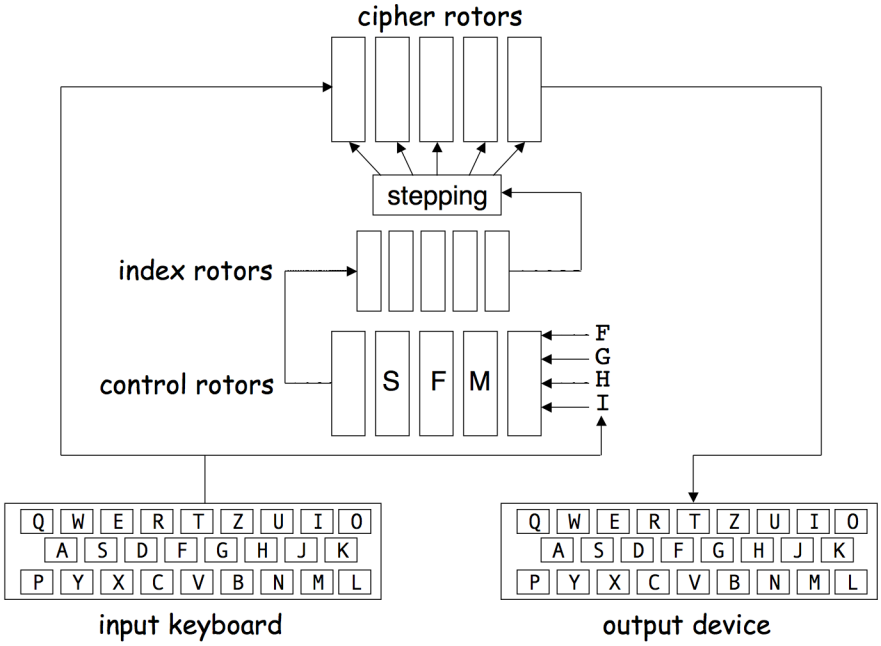

Figure 3: SIGABA Encryption

we discuss below—applies to the Converter M–134C and the CSP–888/889 versions of SIGABA, but not the CSP–2900. The CSP–2900 uses a completely different stepping maze which results in very different cipher rotor motions [10].

As mentioned above, the cipher and control rotors are interchangeable. In addition, each of these rotors can be inserted in either of two orientations—forward or reverse. In the reverse orientation, the letters on the cipher wheel will appear upside down to the operator.

When a rotor is in its forward orientation, the shifting is, for example, from `O` to `N` to `M` and so on. Figure 4 illustrates successive shifts of a single SIGABA cipher (or control) rotor in its forward orientation. Interestingly, the labeling of the SIGABA rotors is the same as that of the Enigma rotors, but the direction of stepping is the opposite.

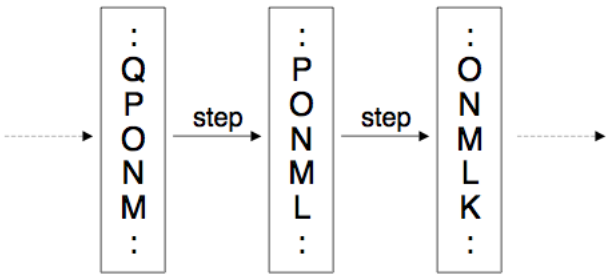

Figure 4: SIGABA Rotor in Forward Orientation

In the reverse orientation the cipher (or control) rotor shifting is from O to P to Q, with the letters appearing upside down, from the operator's perspective. The stepping of a rotor in reverse orientation is illustrated in Figure 5. As discussed in [4] and [11], implementing rotors in software requires some care, and reversed rotors create an additional complication. Also, from Figure 3 we can see that in encrypt mode, the signal passes through the control rotors from right-to-left and the cipher rotors from left-to-right, which creates yet another slight complication when implementing SIGABA in software.

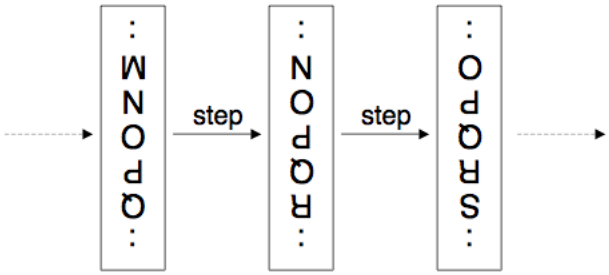

Figure 5: SIGABA Rotor in Reverse Orientation

Curiously, the SIGABA index rotors are labeled in the opposite direction of the cipher and control rotors, that is, the numbers increase in the downward direction as illustrated in Figure 6. The index rotors do not step step when encrypting or decrypting.

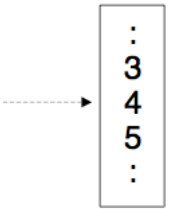

Figure 6: Index Rotor

An interesting quirk of SIGABA is that the letter Z is changed to X before encrypting, and a word space is changed to a Z before encrypting. If the result of decryption is a Z, a space is output. In this way, messages can be encrypted and decrypted with word spaces included, which makes parsing the decrypted message easier. The only drawback is that both plaintext X and Z will be decrypted as X. For example, for some setting of SIGABA, the plaintext message

ZERO␣ONE␣TWO␣THREE␣FOUR␣FIVE␣SIX

encrypts as

IEQDEMOKGJEYGOKWBXAIPKRHWARZODWG

and this ciphertext decrypts as

XERO␣ONE␣TWO␣THREE␣FOUR␣FIVE␣SIX

where "␣" is a word space.

We assume that the odometer effect of the middle three control rotors occurs when the slower rotor steps from O to the next letter, regardless of the orientation of the slower rotor. For example, if the fast rotor is in the forward orientation and currently at the letter O, then the fast and medium rotors will both step when the next letter is typed on the keyboard.[2]

The output values (or value) of the index rotors determines which of the cipher rotors step. Let

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

where O_i is the output from contact i of the index rotor bank. Then the leftmost cipher rotor steps if C_0 is active, the second (from left) cipher rotor steps if C_1 is active and so on. Since there are from one to four active outputs of the index rotors, anywhere from one to four of the cipher rotors will step with each letter typed.

To decrypt with SIGABA, all of the rotors are initialized and stepped precisely as in encryption mode, as described above. However, the inverse cipher rotor permutation must be used. This can be accomplished by feeding the ciphertext letters through the cipher rotors in the opposite direction, as illustrated in Figure 7.

3 SIGABA Keyspace

The SIGABA key is specified by the choice of rotors and their initial positions. If we assume that all possible rotors are available, then a different initial position simply corresponds to a different rotor. Consequently, for the calculation of the theoretical size of the SIGABA keyspace, we can assume that the rotors are all set to some standard initial position. Then the number of keys depends only on

1. The choice of the five cipher rotors.

2. The choice of the five control rotors.

3. The choice of the five index rotors.

[2] In [9] it is claimed that in the forward orientation, the turnover occurs when going from O to N, but in the reverse orientation, it occurs when going from A to B. However, our description of the "ratchet" effect is consistent with the most popular SIGABA software simulator [12].

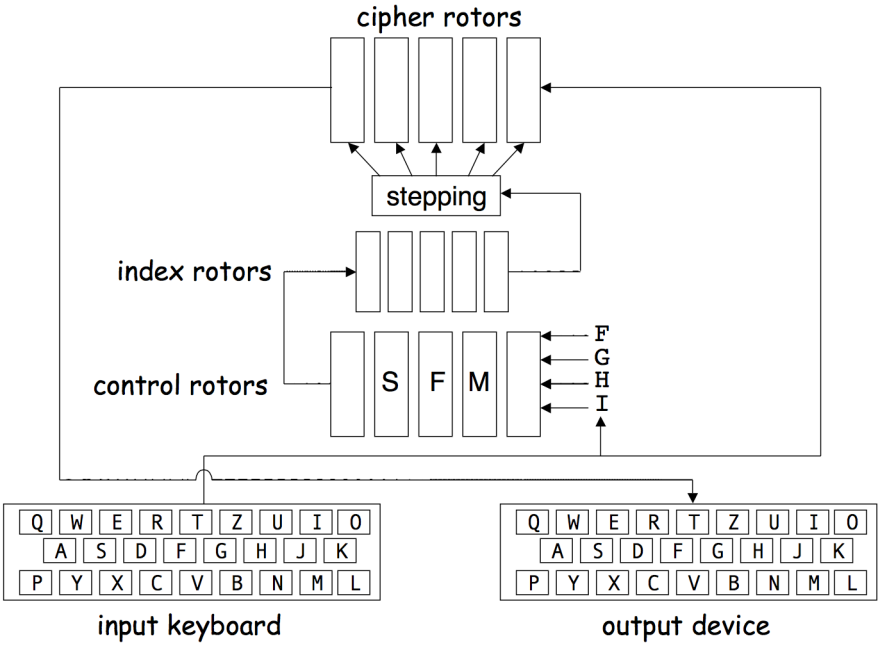

Figure 7: SIGABA Decryption

There are 26! choices for each of the cipher and control rotors. Similarly, there are 10! choices for each of the index rotors. However, since the index rotors do not step, there are only 10! distinct index permutations. This gives a total keyspace of about

$$(26!)^{10} \cdot 10! \approx 2^{884} \cdot 2^{22} \approx 2^{906}.$$

However, the size of the practical SIGABA keyspace is far less than this astronomical number would indicate. During WWII, only ten rotors were generally available for the ten cipher and control rotor slots.[3] Each of these rotors can be inserted forwards or backwards. The order of these ten rotors and their orientations (forward or reverse) must be included in the practical keyspace calculation. In addition, each of the five index rotors can be set to any of 10 positions, and each of the control rotors can be set to any of 26 positions.

In principle, each of the five cipher rotors could also be set to any of 26 positions. However, the usual SIGABA keying procedure set these rotors to a default value, then stepped the rotors in a nonstandard manner—simultaneously stepping the control rotors to their actual starting positions. In addition, the index rotors generally were inserted in one fixed order, in which case only their initial settings were variable. Taking these restrictions into account, it would appear that for SIGABA, as used in WWII, the keyspace was of size

$$10! \cdot 2^{10} \cdot 26^5 \cdot 10^5 \approx 2^{71.9}$$

[3]In fact, several different sets of rotors were used in the various SIGABA machines [3]. For the purposes of our analysis, we assume that one fixed set of ten cipher/control rotors and one fixed set of five index rotors is available. Furthermore, we assume that these rotors are known to the attacker.

as claimed in [4].

However, a careful reading of the SIGABA manual [7] reveals that the setting of the control rotors was sent in the clear as a *message indicator* or MI. Therefore, assuming the MI was intercepted and its meaning was known to the attacker, the actual keyspace for SIGABA—as it was generally used in WWII—was of size

$$10! \cdot 2^{10} \cdot 10^5 \approx 2^{48.4} \tag{2}$$

as (correctly) stated in the article [9]. But, on the SIGABA-encrypted POTUS-PRIME[4] link between Roosevelt and Churchill, the control and cipher rotor settings were set independently, and neither was sent in the clear, which implies a keyspace in excess of 95 bits [9] (in the next section we provide a precise calculation of the keyspace for this case).

A keyspace of size $2^{48.4}$ is small enough that today it is susceptible to an exhaustive key search.[5] But a keyspace of this magnitude would have been unassailable using 1940s technology, provided no shortcut attack was available.

4 SIGABA Attack

For this attack, we assume that all three banks of rotors are set independently. We also assume that there is only one set of index rotors, and that these five rotors can be placed in any order, and that a total of ten rotors are available for use as cipher and control rotors. The control and cipher rotors can be inserted in any order and there are two orientations for each of these rotors. Under these assumptions, the keyspace is apparently of size

$$10! \cdot 2^{10} \cdot 26^{10} \cdot 5! \cdot 10^5 \approx 2^{102.3}.$$

As noted above, there are only 10! distinct index permutations. Since $5! \cdot 10^5 > 10!$, the effective keyspace is reduced slightly to

$$10! \cdot 2^{10} \cdot 26^{10} \cdot 10! \approx 2^{100.6}.$$

However, due to the fact that pairs of index rotor outputs are ORed together to determine the cipher rotor stepping, only

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

distinct index permutations can be distinguished. This reduces the essential keyspace to no more than

$$10! \cdot 2^{10} \cdot 26^{10} \cdot 2^{16.8} \approx 2^{95.6}.$$

This keyspace of size $2^{95.6}$ does not represent the way that SIGABA was generally used in WWII, but it does represent the largest keyspace that could have been used

[4] President Of The United States – PRIME Minister.

[5] The Data Encryption Standard (DES) has a 56-bit key and it has been successfully attacked by an exhaustive key search.

at that time. That is, it represents the largest practical keyspace, given the hardware that was typically available with a SIGABA machine in WWII. Increasing the number of available rotors would increase the keyspace, but we limit ourselves to the number of rotors that will fit in the device at one time, since this is typically all that was available with the cipher. Finally, we assume that all of the rotors and the inner workings of the device are known to the cryptanalyst.

Our attack requires some amount of known plaintext. This attack occurs in two phases—a primary phase and a secondary phase. In the primary phase, we try all cipher rotor initializations, retaining those that are consistent with the known plaintext. Then in the secondary phase, we again use the known plaintext, this time to determine the control and index rotor initializations, and thereby recover the key.

Suppose that we have a SIGABA-encrypted ciphertext message, where the first several letters of the corresponding plaintext are known. In the primary phase we deal with the cipher rotors—their order, orientations and initial settings. Collectively, we refer to the cipher rotor initializations as the cipher rotor *settings*. We refer to an incorrect choice of cipher rotor settings as a *random* setting, while the correct setting is said to be *causal*.

For each cipher rotor setting that survives the primary phase, a secondary phase is required. The secondary phase consists of trying possible control and index rotor settings to determine which are consistent with the known plaintext. In this way, the random primary survivors are eliminated and, in the causal case, we determine the key.

Here, our goal is to outline the attack in sufficient detail to convince the reader of its correctness. Some improvements to the basic attack are also discussed. Many of these improvements appear as exercises in [11]. Additional empirical results will appear in [2].

4.1 Primary Phase

We are assuming that ten different cipher rotors are available. Also, each cipher rotor has two possible orientations and 26 possible initial positions. Therefore, the number of ways to select and initialize the five cipher rotors is

$$\binom{10}{5} \cdot 5! \cdot 2^5 \cdot 26^5 \approx 2^{43.4}.$$

For each of these choices, we determine whether the setting is consistent with the known plaintext as follows.

Recall that for each letter typed, from one to four of the cipher rotor rotates. This implies that once we specify the cipher rotors, their orientations and their initial settings, the number of possible new permutations at any given step is

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

Now suppose that we correctly guess the cipher rotor settings at some point in time. We can then generate each of the 30 possible subsequent permutations and determine which are consistent with the next known plaintext letter. That is, we can test each of these 30 subsequent permutations to see which encrypt the next known plaintext letter to the corresponding ciphertext letter. For each surviving permutation, we can repeat this process using the next known plaintext letter and so on.

Modeling the encryption permutations as uniformly random, the matches follow a binomial distribution with $p = 1/26$ and $n = 30$, yielding an expected number of matches of $30/26 \approx 1.154$ per step.[6] This can be viewed as a branching phenomenon, where the number of possible paths tends to increase with each known plaintext letter analyzed. That is, at each step, the number of possible paths increases, which seems to be the opposite of what we would like to see occur. Nevertheless, we can obtain useful information from this process, as outlined below, but first we consider a simple example.

Suppose we have selected five of the ten candidate rotors as cipher rotors, and we have placed them in a specified order and selected their orientations. This, together with the initial positions of the selected cipher rotors constitutes a putative setting. Consider, for example, the case where the selected cipher rotors are set to `AAAAA`, that is, each of the five cipher rotors is initialized to `A`. Then we know the putative encryption permutation and if it does not encrypt the first known plaintext to the first known ciphertext, this cipher rotor setting is not causal and we can discard it. This immediately reduces the number of candidates by a factor of 26, since there is only a 1/26 chance of a letter matching at random.

Suppose that the first letter does match. Then we must try all 30 possible steps of the five cipher rotors and save any of these that encrypt the second plaintext letter to the second ciphertext letter. Since we make 30 comparisons, The expected number of matches that occur at random is, as mentioned above, $30/26 \approx 1.154$. An example of this process is illustrated in Figure 8. In this example, the first known plaintext letter is consistent with the initial setting `AAAAA`, and the first three letters are consistent with each of the given paths.

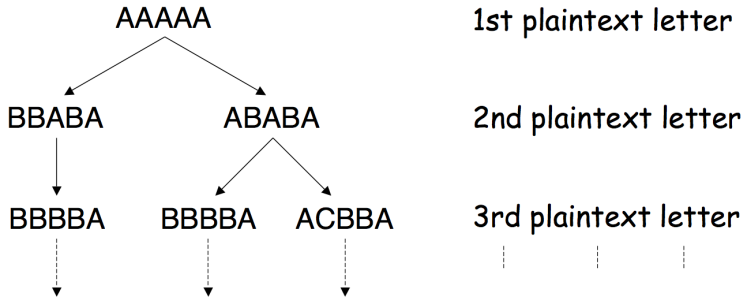

Figure 8: Example of Consistent Paths

[6]Note that if SIGABA employed four or fewer cipher rotors, the number of expected matches would be less than one, resulting in a much weaker cipher.

Note that at the third plaintext letter in Figure 8 we have two consistent paths ending at `BBBBA`. Since the next step depends only on the current cipher rotor settings, and since we are only interested in the initial setting (not the entire path), we can merge these paths as illustrated in Figure 9. This merging is useful since it effectively reduces the number of paths under consideration, while not degrading the success of this phase of the attack.

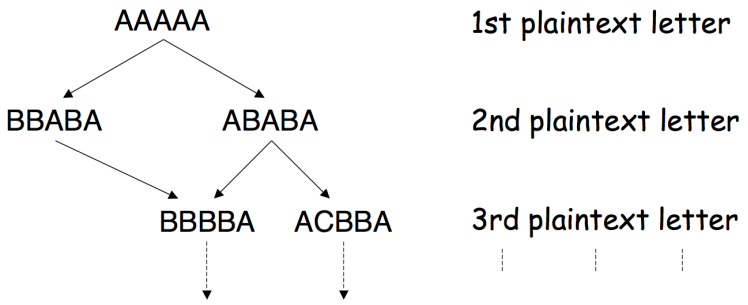

Figure 9: Merging Paths

In the random case, the analysis above holds, so that at each step we expect an increase by a factor of 1.154 (before merging). In contrast, the causal case provides a slightly higher increase on average, since we are assured one causal match, with the remaining elements matching as in the random case. This gives us a method to statistically distinguish random from causal and thereby reduce the number of random cases.

The results in Table 1 were obtained as follows. Given a crib consisting of "steps" consecutive known plaintext letters, we conducted the indicated number of "tests." Each test consisted of first generating random settings for the order and initialization of the rotors. Then, since the first letter is encrypted before stepping the rotors, the first plaintext letter was encrypted to determine whether it matches the first ciphertext letter. A random setting only survives the first step with a probability of 1/26. If a setting survives this first step, then all 30 possible cipher rotor steps are tried and any that are consistent with the second plaintext letter are saved. This is repeated for subsequent letters, until no path survives a given step (in which case the setting is known to be random), or the indicated number of steps has been taken. Table 2 contains the analogous results for causal rotor settings, that is, the correct setting is tested using the same procedure just described for the random settings.

For both of these cases, we have merged paths, as discussed above and illustrated in Figure 9. Table 1 indicates that using 30 known plaintext letters, we expect only a fraction of about 0.00427 of the random settings to survive ("non-zero settings"), and each of these survivors will have expanded to an average of about 16.5 merged paths, with a maximum for the cases tested of 84. In contrast, Table 2 shows that with 30 known plaintext letters, we expect the causal path to have generated about 29.6 consistent branches, with, for the 10,000 cases tested, a maximum of 151 and a minimum of just a single consistent path.

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 1: Random 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

Table 2: Causal Case

The results in Table 1 show that we can eliminate the majority of random settings using a small amount of known plaintext. Note that although the number of random settings decreases, the total number of merged paths ("non-zero settings" times "avg per non-zero") actually increases slightly as more known plaintext is used.

From the causal results in Table 2, we see that we can further reduce the number of random settings by saving only those that are, say, above the expected mean in the corresponding causal case. Of course, this refinement implies that we will sometimes discard the causal case, with the probability depending on the selected threshold. That is, we can reduce the number of primary phase survivors, at the expense of a lower probability of success. Unfortunately, Tables 1 and 2 indicate that the variance is high, so a significant number of random cases will remain for any reasonable probability of success.

For a small amount of known plaintext, the primary phase work factor is on the order of $2^{43.4}$, since most random paths do not survive the first known plaintext test. However, if we use more known plaintext and if we save all merged paths, then the amount of work may exceed $2^{43.4}$, since the number of surviving merged paths tends to grow. For example, suppose we use 100 known plaintexts. After the first known plaintext, we have $2^{43.4}/26 \approx 2^{38.7}$ surviving paths. From Table 1 we see that after 100 known plaintext letters, we expect the number of surviving merged paths to have increased to about

$$2^{43.4} \frac{203 \cdot 100.5}{10^5} \approx 2^{41.1}. \tag{3}$$

That is, when using the 99 known plaintexts after the first, the number of merged paths increases from $2^{38.7}$ to $2^{41.1}$, and at each step we must process all of the merged paths. We can approximate the number of paths at step k by $2^{38.7}x^k$. Then we have $2^{38.7}x^{99} = 2^{41.1}$, which implies $x \approx 1.017$, and the primary work is given by

$$2^{43.4} + 2^{38.7} \sum_{k=0}^{99} 1.017^k = 2^{43.4} + 2^{38.7} \frac{1.017^{100} - 1}{0.017} \approx 2^{46.7}.$$

From 3 we see that with 100 known plaintext letters, only about $2^{41.1}$ merged paths survive the primary phase and, from Table 1, about $0.00204 \cdot 2^{43.4} \approx 2^{34.5}$ random settings survive.

As discussed above, the number of random settings decreases with more known plaintext. However, in the secondary phase discussed in Section 4.3 below, each of the merged paths must be tested and the number of merged paths increases as more known plaintext letters are used. This appears to be the opposite of what is desired. But in spite of having more cases to test in the secondary phase, the information obtained from the merged paths yields a more efficient attack overall.

4.2 Secondary Phase

For each of the cipher rotor settings that survived the primary phase, a secondary test is required. This secondary test will determine whether a primary phase survivor is consistent with any setting of the control and index rotors. In the process, we eliminate random survivors from the primary phase and for the causal survivor we determine the rotor settings and thereby recover the key.

For the secondary test, suppose we choose the order and initial positions of the index rotors and the order, orientation and initial positions of the control rotors, given the putative cipher rotor settings from the primary phase. The number of settings for the index and control rotors appears to be

$$5! \cdot 10^5 \cdot 5! \cdot 2^5 \cdot 26^5 \approx 2^{58.9}.$$

However, as noted above, there are only about $2^{16.8}$ distinct index permutations, which reduces the overall work factor to

$$2^{16.8} \cdot 5! \cdot 2^5 \cdot 26^5 \approx 2^{52.2}.$$

That is, the work factor for the secondary part of the attack appears to be on the order of $2^{52.2}$ for each putative setting that survived the primary phase. Fortunately, we can improve on this naïve implementation of the secondary phase by considering the merged paths—as opposed to settings—that survive the primary phase.

4.3 Secondary Phase Refinements

To reduce the secondary work factor we again rely on known plaintext. Here, we only outline the plan of attack with details provided for selected issues that arise.

The interaction of the control rotors and the index rotors is illustrated in Figure 10, where we have collapsed the five control rotors into a single permutation (denoted as "control") and, similarly, the five index rotors are considered as a single permutation (denoted as "index"). The four inputs to the control permutation, `F`, `G`, `H` and `I`, are activated at each step. This results in four active outputs, which are combined as indicated before being fed into the index permutation. At least one—and at most four—inputs to the index permutation will be active. The outputs from the index permutation are combined in pairs, as indicated, and these determine which of the cipher rotors—denoted C_0, C_1, C_2, C_3, C_4 from left to right—step. At least one cipher rotor will step, and at most four will step.

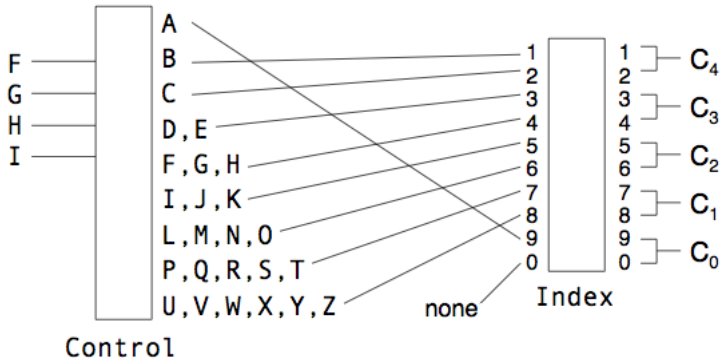

Figure 10: Control and Index Permutations

In Figure 10, the control permutation changes with each letter typed, but the index permutation is fixed for the entire message. Since the control permutations are changing, we model their output as uniformly random, that is, we assume that each of the $\binom{26}{4}$ combinations of output letters is equally likely at each step. Then, due to the way that the control rotor outputs are ORed together, the inputs to the index permutation are not uniform. For example, input 8 will be active much more often than inputs 1, 2 or 9, and input 0 is never active.

The outputs of the index rotors are ORed in pairs, and the results determine which cipher rotors step. Therefore, if we have sufficient information on the frequency of the stepping of individual cipher rotors, we can assign probabilities to the index permutations. Note that for any merged path that survives the primary phase, we

obtain putative stepping counts for each of the cipher rotors. For example, consider the merged paths in Figure 9. Since the initial rotor positions were `AAAAA`, for the merged paths ending with `BBBBA` we know that cipher rotors C_0 through C_3 each stepped once and rotor C_4 did not step. Furthermore, this holds for both of the paths that were merged into this particular path. That is, with respect to the stepping counts for individual rotors, there is no loss of information due to the merging of paths.

Next, we consider a specific example to illustrate the relationship between an index permutation and the cipher rotor stepping counts. For example, suppose that the index permutation is $(5, 4, 7, 9, 3, 8, 1, 0, 2, 6)$, that is, input 0 is mapped to output 5, input 1 is mapped to output 4, and so on. Then by considering the pairs of outputs that determine the cipher rotor stepping, we can see that some cipher rotors will step more often than others. The example in Table 3 illustrates this point. In this example, cipher rotor C_4 steps if either output 1 or 2 (or both) of the index permutation is active. For the index permutation in Table 3, inputs 6 and 8 are mapped to outputs 1 and 2, respectively. Inputs 6 and 8 of the index permutation correspond to outputs 6 and 8 of the current control permutation, and at least one of these outputs is active if one or more of the 10 letters `L`, `M`, `N`, `O`, `U`, `V`, `W`, `X`, `Y` or `Z`, which are connected to these outputs, is active. On the other hand, cipher rotor C_2 steps only when output `A` of the control permutation is active. As a result, C_4 will step much more often than C_2.

	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

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

Assuming the control rotors generate random permutations, the expected number of steps for cipher rotor i depends solely on the number of control rotor output letters that feed into C_i, as illustrated in Figure 10. All of the 45 possible input pairs and the corresponding number of control rotor output letters are tabulated in Table 4.

If we have sufficient known plaintext available, we obtain information related to the "count" column of Table 4 for each cipher rotor, simply based on a count of the number of times that cipher rotor i steps. Then from the "pairs" column, we obtain restrictions on the index permutation.

This begs the question of how much known plaintext is required for this phase of the attack. We can estimate the requirement as follows. Each cipher rotor is connected to k control rotor outputs (via the index permutation), where k is in the range of 1 to 11, inclusive. We can determine the expected "stepping ratios" for a cipher rotor when it is connected to exactly k control rotor outputs. These fractions will sum to much more than one, since more than one rotor generally steps. To

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)

Table 4: Index Permutation Input Pairs

compute these ratios, we assume all outputs of the control rotors are equally likely and we generate all $\binom{26}{4} = 14{,}950$ of these equally likely outputs, counting the number of times that at least one element of each of the pairs in Table 4 occurs. The resulting stepping ratios are given in Table 5, where "step ratio" is obtained by dividing "step count" by 14,950. Note that these results are independent of the index permutation.

	example		
letters	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

Table 5: Cipher Rotor Stepping Ratios

Now given putative cipher rotor stepping counts (as determined from known plaintext in phase one of the attack), the numbers in Table 5 can be used to determine the most likely pairs of control rotor output letters connected to each cipher rotor. Since these connections occur via the index permutation, combining this information with Table 4 significantly reduces the number of possible index permutations.

A valid index permutation must contain five pairs from Table 4, where $0, 1, \ldots, 9$ each appear once within the set of pairs, and the corresponding "letters" columns of the five pairs must sum to 26 (since all 26 letters are connected). It is not difficult to compute all such groupings of five pairs—we find there are 2148 such groupings, assuming that the letter counts are ordered from, say, smallest to largest.

Now given frequency counts for the stepping of individual cipher rotors, we can hope to distinguish the number of letters connected (via the index permutation) to each cipher rotor. For example, suppose the frequency counts show that each of cipher rotors C_0, C_1, C_2, C_3, C_4 had stepping ratios of $0.15, 0.29, 0.60, 0.74, 0.91$, respectively. According to Table 5, these results indicate that the number of letters connected to cipher rotors C_0, C_1, C_2, C_3, C_4 are, most likely, $1, 2, 5, 7, 11$, respectively. Six of the 2148 valid combinations of five pairs derived from Table 4 are consistent with this ordering. These consistent sets of pairs appear in Table 6.

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

Table 6: Sets of Pairs Consistent with Letter Counts $1, 2, 5, 7, 11$

It is straightforward to show that the 2148 consistent groupings of five pairs reduces to 89 distinct *categories* based on the corresponding letter counts. For example, one of these 89 categories corresponds to letter counts of $1, 2, 5, 7, 11$ and, as noted above, there are six pairs of five that are consistent with this category. On average, about 24 sets of five pairs are consistent with each category, and the range is from 3 to 72.

Given stepping counts computed from known plaintext, we can compute a score to determine the best match among the 89 categories as follows. Let x_i be the "step ratio" in row i and the last column of Table 5. Then $i \in \{1, 2, \ldots, 11\}$ and x_i is the expected fraction of the time that a cipher rotor steps when it is connected to i letters. For notational convenience, let $x_0 = 0$ and $x_{12} = 1$.

From the known plaintext we compute the stepping ratios s_0, s_1, s_2, s_3, s_4, where we may assume that

$$s_0 < s_1 < s_2 < s_3 < s_4.$$

For each s_j, we first determine the index i for which $x_i \le s_j < x_{i+1}$. Then let

$$t_j = \frac{s_j - x_i}{x_{i+1} - x_i} + i$$

with the proviso that if $t_j < 1$, then let $t_j = 1$, and if $t_j > 11$, let $t_j = 11$. 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 to cipher rotor j. Note that we are using a linear interpolation for points between consecutive x_i. This (or something comparable) is required since the x_i are not equally spaced.

Now given the t_j computed in the previous paragraph, let $(u_0, u_1, u_2, u_3, u_4)$ be one of the 89 categories discussed above. Recall that

$$u_0 < u_1 < u_2 < u_3 < u_4.$$

We compute the the score as (the square of) the Euclidean distance,

$$d = (t_0 - u_0)^2 + (t_1 - u_1)^2 + \cdots + (t_4 - u_4)^2.$$

The category that is at the minimum distance from $(t_0, t_1, t_2, t_3, t_4)$ is selected as the most likely category.

We can now estimate the known plaintext requirement for the secondary phase. Table 7 contains empirical results, assuming the indicated numbers of known plaintext letters are available, in each case using the scoring method discussed in the previous paragraph to select the closest category. These results show that, for example, with 100 known plaintexts, we obtain the correct category about 23% of the time, and exactly two of the elements are incorrect (and almost certainly one is off by "+1" the other is off by "−1"), with a probability of about 59%. Consequently, with 100 known plaintext letters, we have a probability of about 0.82 that the correct category is either the best-scoring category or one of the other 10 (or fewer) categories that are nearest to the best-scoring category. Since there are only 24 sets of pairs per category (on average), given 100 known paintext letters we need to test fewer than 2^8 sets of pairs (on average) and we will obtain the correct index permutation with a probability of about 0.82.

There is more information available than we have used to compute the numbers in Table 7. For example, if at any point, only one cipher rotor steps, we can immediately eliminate rows 1, 2 and 3 from Table 4, since there are always four active outputs from the control permutation. Although a single rotor stepping is a relatively rare event (occurring about 2.5% of the time according to Table 8, below), when it occurs, it immediately eliminates nearly 1/4 of the possible index permutations.[7] Using such refinements, we could expect to reduce the known plaintext requirement from that indicated in Table 7, while not diminishing the probability of success. However, the

[7]Using a similar analysis, when two cipher rotors step, we only gain a small amount of information about the index permutation, and when three or four rotors step, we gain no additional information. However, it may be beneficial to separate the cases where one, two, three or four cipher rotors step instead of lumping all of the stepping information together, since the distributions differ in each case. The drawback is that the counts for each case will be much smaller, given the same amount of known plaintext.

plaintext	pairs correct 0	1	2	3	4	5	iterations
50	0.0287	0.2213	0.1837	0.4756	0.0000	0.0910	10^6
100	0.0036	0.1076	0.0672	0.5884	0.0000	0.2332	10^6
150	0.0006	0.0517	0.0234	0.5522	0.0000	0.3721	10^6
200	0.0001	0.0253	0.0085	0.4722	0.0000	0.4939	10^6
250	0.0000	0.0128	0.0033	0.3900	0.0000	0.5939	10^6
300	0.0000	0.0064	0.0013	0.3153	0.0000	0.6769	10^6
400	0.0000	0.0018	0.0002	0.2023	0.0000	0.7957	10^6
500	0.0000	0.0005	0.0001	0.1300	0.0000	0.8694	10^6
1000	0.0000	0.0000	0.0000	0.0157	0.0000	0.9843	10^6

Table 7: Secondary Known Plaintext

number of rotors that step at a given iteration may be difficult to utilize due to the merging of paths in the primary phase.

In summary, using the cipher rotor stepping counts, we can reduce the number of index permutations we must analyze to a small fraction of the $10!/32 \approx 2^{16.8}$ that we would otherwise need to consider. With sufficient known plaintext, the average number of permutations that we need to check is about 2^8, and we can likely reduce this further by using more of the available information.

Assuming about 100 known plaintext letters are available, the average work factor for the secondary phase of the attack is about

$$2^8 \cdot 5! \cdot 2^5 \cdot 26^5 \approx 2^{43.4} \tag{4}$$

and, again, it should be possible to reduce the factor of 2^8. This work factor is a significant improvement over the naïve implementation of the secondary phase, and comparable to the work for the primary phase of the attack. However, the secondary work factor in (4) applies to each surviving merged path from the primary phase. Consequently, we can improve the overall attack by either reducing the number of merged paths from the primary phase or by making the secondary phase more efficient (or both).

Next, we suggest a method to take further advantage of the computed cipher rotor stepping counts obtained in the primary phase. For each distinct index permutation we can compute the probabilities p_i, for $i = 1, 2, 3, 4$, that precisely i cipher rotors step, where the probabilities are computed over all possible control rotor outputs. That is, for each of the $10!/32$ distinct index permutations, each of the 4-letter control rotor outputs—of which there are $\binom{26}{4}$—is assumed to be equally likely and each of these outputs is combined as indicated in (1). The average, maximum and minimum over all index permutations appears in Table 8.

One interesting feature of the results in Table 8 is that in each case, the range of possible values is small. More importantly, the ranges only overlap when 2 and 4

rotors step	average	maximum	minimum
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 8: Cipher Rotor Stepping

rotors step, and not at all in any other case. Consequently, without making any assumption about the index permutation, the cipher rotor steppings obtained in the primary phase can be used to assign a score to each primary phase survivor. For a given primary survivor, this score is computed based on the number of cipher rotors that step for each known plaintext letter, using the average probabilities in Table 8. Then in the secondary phase, we can test the highest scoring primary survivors first, then test the next highest scoring survivors, and so on, until the key is recovered. This stepping information could instead be employed to "trim" unlikely paths in the primary phase, thereby reducing the number of primary survivors. In any case, merging paths create a slight complication, since different numbers of rotors can step to arrive at a particular merge point. One solution is to simply take the maximum probability of paths that merge.

We have shown that the typical secondary work factor for each primary merged path is no more than about 2^{43}, assuming sufficient known plaintext is available. This amount of work is clearly feasible today, although the attack is not trivial to implement. The primary phase of this attack has a similar work factor and it is also feasible. However, for the attack described in this paper, the primary phase yields a large number of survivors, which makes the overall cost of the attack extremely high.

5 The Bottom Line

It is instructive to compare the work factor of our SIGABA attack with other simpler attacks. First, consider a straightforward exhaustive key search. For consistency with the attack discussed in this paper, we view the exhaustive key search as having a primary phase and a secondary phase, where there are $2^{43.4}$ primary cases (cipher rotor settings), and each of these requires work of $2^{52.2}$ in the secondary phase (control and index rotor settings). This gives a maximum work factor of $2^{95.6}$ and an expected work of $2^{94.6}$, and the attack would always succeed, assuming we could actually do the work.

Next, consider an attack which uses a single known plaintext letter. Analogous to our attack outlined in this paper, we have a primary phase where we test each of the $2^{43.4}$ cipher rotor settings, and only $1/26$ of these will match the one known plaintext letter. Then for each of these $2^{43.4}/26 \approx 2^{38.7}$ primary survivors, we have

secondary work of $2^{52.2}$. This gives a total work of $2^{90.9}$ and, therefore, an expected work of $2^{89.9}$. Again, the probability of success is one.

Now suppose that we have 100 known plaintext letters. Then we can use the primary phase in Section 4.1 to reduce the number of cipher rotor settings from $2^{43.4}$ to about $0.00203 \cdot 2^{43.4} \approx 2^{34.5}$. For each of these surviving settings, we can apply the straightforward secondary phase discussed in Section 4.2. This secondary phase has a work factor of $2^{52.2}$ per primary survivor, which gives a maximum work factor of $2^{86.7}$ and an expected work of $2^{85.7}$. In this case, the probability of success is one.

Finally, consider an attack with 100 known plaintext letters where we use the primary phase discussed in Section 4.1 and the "refined" secondary phase discussed in Section 4.3. Then in the secondary phase we must test all surviving merged paths—as opposed to settings—and from (3), we see that about $2^{41.1}$ such paths are expected. With 100 known plaintext letters, the secondary phase work is, according to (4), no more than $2^{43.4}$ per primary survivor. Therefore, the total work for this attack is no more than $2^{84.5}$, and the expected work is at most $2^{83.5}$, while the probability of success is about 0.82 (see Table 7 and the related discussion). While this work factor is only a modest improvement over the more straightforward secondary test, and the attack is now only probabilistic, as noted above, there are various further refinements that are likely to reduce the work significantly. In particular, trimming low probability paths in the primary phase should be a good strategy.

The comparison between these attacks is summarized in Table 9. Note that "secondary work" is the work per primary survivor.

attack	primary survivors	secondary work	total work	probability of success
exhaustive key search	$2^{43.4}$	$2^{52.2}$	$2^{95.6}$	1.00
1 known plaintext	$2^{38.7}$	$2^{52.2}$	$2^{90.9}$	1.00
100 known plaintexts	$2^{34.5}$	$2^{52.2}$	$2^{86.7}$	1.00
100 known plaintexts	$2^{41.1}$	$2^{43.4}$	$2^{84.5}$	0.82

Table 9: Attack Comparison

Table 9 shows that our attack, while far from practical, is more efficient than the more obvious attacks on the full SIGABA keyspace. However, the improvement is modest, a substantial crib is required and the attack is only probabilistic (but with a high probability of success). Perhaps this should be viewed as additional evidence of the strength of the SIGABA design, particularly in comparison to other World War II era ciphers.

6 Conclusion

SIGABA, as typically used in WWII, has a keyspace of size $2^{48.4}$, which implies that an exhaustive key search has a work factor of $2^{47.4}$. However, the SIGABA-encrypted POTUS-PRIME link between Roosevelt and Churchill used the full available keyspace of more than 95 bits. It is interesting that keyspaces of these sizes were chosen. From the designers' perspective, there would be no incentive to have a keyspace that is larger than a known shortcut attack, since a larger keyspace entails more secret settings and consequently more chance for errors and miscommunication.

In WWII, a work factor of $2^{47.4}$ would certainly have been untouchable, particularly for tactical communications. Nevertheless, for the strategically important communication between Allied leaders, it would have been reasonable to use a larger key size, provided that the larger key actually yielded additional security. Based on this logic, it would seem likely that the designers of SIGABA believed that the cipher provided something close to a full 95 bits of security. In this paper, we have outlined an attack that requires somewhat less than 95 bits of work, and it is certainly possible to improve on the attack presented here.

It is worth noting that the designers of SIGABA almost certainly viewed the keyspace issue in a more intuitive manner than we do in this paper. Evidently, the stepping maze was designed to avoid the problems inherent with regularly-stepping cipher rotors. Given this design criteria, the designers of SIGABA might simply have chosen the rotors so that there were far too many combinations for an exhaustive search, and so as to create the desired irregular motion. In any case, it would be interesting to know more about the attacks that were considered by Rowlett and Friedman, and their reasons for designing SIGABA as they did.

Acknowledgment

The authors thank Frode Weierud for numerous helpful comments and suggestions which greatly improved this paper.

References

[1] S. Budiansky, *Battle of Wits*, The Free Press, 2000

[2] W. O. Chan, Cryptanalysis of SIGABA, Master's Thesis, Department of Computer Science, San Jose State University, May 2007

[3] S. J. Kelly, *Big Machines*, Aegean Park Press, 2001

[4] M. Lee, Cryptanalysis of the SIGABA, Master's Thesis, University of California, Santa Barbara, June 2003, at `ucsb.curby.net/broadcast/thesis/thesis.pdf`

[5] National cryptologic museum, the big machines exhibit, at `www.nsa.gov/museum/museu00002.cfm`

[6] Operating instructions for ECM Mark 2 (CSP 888/889) and CCM Mark 1 (CSP 1600), at `www.hnsa.org/doc/crypto/ecm/index.htm`

[7] R. Pekelney, ECM MARK 2 and CCM MARK 1, at `www.hnsa.org/doc/crypto/ecm/`

[8] L. F. Safford and D. W. Seiler, Control circuits for electric coding machines, United States patent number 6,175,625, January 2001

[9] 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

[10] G. Sullivan, The ECM Mark II: some observations on the rotor stepping, *Cryptologia*, Vol. 26, No. 2, April 2002

[11] M. Stamp and R. M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Wiley Interscience, 2007

[12] USS Pampanito, at `www.maritime.org/ecmapp.htm`

About the Authors

Mark Stamp has many years of experience in information security. He can neither confirm nor deny that he spent seven years as a cryptanalyst with the National Security Agency, but he can confirm that he recently spent two years designing and developing a security product at a small Silicon Valley startup company. Dr. Stamp currently holds an academic position in the Department of Computer Science at San Jose State University where he teaches courses on information security. He has written two textbooks, *Information Security: Principles and Practice* (Wiley 2006) and *Applied Cryptanalysis: Breaking Ciphers in the Real World* (Wiley 2007).

Wing On Chan graduated from San Jose State University with a Bachelors Of Science degree in Computer Science and is now completing his Masters Of Science degree, also at San Jose State University and also in Computer Science. He enjoys reading up on the latest trends in computer technology, especially the marketing claims of "unbreakable" security technologies.