

# Breaking Enigma

*Samantha Briasco-Stewart, Kathryn Hendrickson, and Jeremy Wright*

<b>1 Introduction</b>	<b>2</b>
<b>2 The Enigma Machine</b>	<b>2</b>
2.1 Encryption and Decryption Process	3
2.2 Enigma Weaknesses	4
2.2.1 Encrypting the Key Twice	4
2.2.2 Cillies	5
2.2.3 The Enigma Machine Itself	5
<b>3 Zygalski Sheets</b>	<b>6</b>
3.1 Using Zygalski Sheets	6
3.2 Programmatic Replication	7
3.3 Weaknesses/Problems	7
<b>4 The Bombe</b>	<b>8</b>
4.1 The Bombe In Code	10
4.1.1 Making Menus	10
4.1.2 Running Menus through the Bombe	10
4.1.3 Checking Stops	11
4.1.4 Creating Messages	11
4.1.5 Automating the Process	11
<b>5 Conclusion</b>	<b>13</b>
<b>References</b>	<b>14</b>

# 1 Introduction

To keep radio communications secure during World War II, forces on both sides of the war relied on encryption. The main encryption scheme used by the German military for most of World War II employed the use of an Enigma machine. As such, Britain employed a large number of codebreakers and analysts to work towards breaking the Enigma-created codes, using many different methods. In this paper, we lay out information we learned while researching these methods, as well as describe our attempts at programatically recreating two methods: Zygaliski sheets and the Bombe.

## 2 The Enigma Machine

The Enigma machine was invented at the end of World War I, by a German engineer named Arthur Scherbius. It was commercially available in the 1920s before being adopted by the German military, among others, around the beginning of World War II. The commercial version of the Enigma machine was composed of three rotors, a reflector wheel, a keyboard, and an array of lights laid out like the keyboard. Each of the three rotors had 26 contacts on each side, and a unique internal wiring which connected contacts to each other. These contacts represented the 26 letters in the alphabet and the connections mapped one letter to another. The reflector wheel was mounted on the end of the machine to the left of the three rotors and connected pairs of contacts on the leftmost rotor to each other. Pressing a key on the keyboard would cause current to flow through the circuit which went from the key, through the contacts on the rotors and reflector wheel, back through the rotors in the opposite direction, and then to a lightbulb. This circuit created a complex substitution cipher that would change whenever a key was pressed. Pressing a key would cause one or more rotors to rotate one step, changing where the rotor contacts were connected. Each letter of the ciphertext was thus enciphered with a different substitution cipher, mimicking the effect of a one-time pad. It is important to note that this encryption process is symmetric (to decrypt a ciphertext, the ciphertext just had to be encrypted again) and a letter could be never encrypted to itself (since the reflector never connected a letter to itself, the letter must travel back through a different path than the one through which it came to the reflector).

The German military adapted the commercial Enigma machine before World War II, changing which rotors were used and adding 2 more. In addition, the rotors in the military Enigma machine were able to be swapped: any 3 of the 5 rotors could be used, in any order creating 60 combinations of rotor orders. Each of the 5 rotors had an inner alphabet ring attached to the center wiring that could be rotated independently from the rotor housing and outer alphabet ring. This added another factor of 26 possible combinations per rotor and the position of the inner alphabet ring was called the “ring setting.” In addition to the changes to the rotors, a plugboard was added that was inserted between the rotors and the keyboard and light assembly. This plugboard, “steckerbrett” in German and known as a “steckerboard” by British

codebreakers, had 26 sockets, one for each letter in the alphabet. A wire would be plugged into two letters' sockets which would swap the letters when going into the rotors from the keyboard and out of it to the lightboard. The German practice after January 1939 was to connect 10 pairs of letters on this plugboard, or "steckering" the two letters together, and leave 6 letters unconnected, or the letters were "self-steckered."

The rotors moved in order to introduce a different substitution cipher for every keypress. Each rotor has a notch at one letter called a "turnover point." When the turnover point was reached, the notch would engage a device that would, on the next key press, rotate the rotor to the left by one letter. The action of causing the other rotor or "step" to move was called a "turnover." Thus, the rightmost rotor stepped with every key press; the middle rotor stepped once with every full rotation of the right rotor, so every 26 key presses, and the leftmost rotor stepped once with every full rotation of the middle wheel, every 676 key presses. In addition, the middle rotor would step once whenever the leftmost rotor stepped which only happened when the middle rotor steps so this would cause a "double step."

There were a few other versions of Enigma machines developed throughout the war, including a four-rotor variant, a variant with a configurable reflector wheel, and a variant where the reflector wheel also rotated. We will focus on the version described above for the rest of this paper.

## **2.1 Encryption and Decryption Process**

The machine itself was only one part of the encryption process. Because of the many ways to configure the machine before encrypting, the German military followed a precise method that was used by every operator. This ensured that communication was secure and that it was decryptable by other operators who had the same procedural knowledge.

There are several configurable settings on the Enigma machine and they were split into two categories -- daily settings and per-message settings. Daily settings included which rotors to use and the rotor order, the "ring setting" for each rotor, and the plugs on the plugboard. Daily settings were physically disseminated throughout the military monthly. The per-message settings were set by the operator who was expected to choose a key to encrypt the message with (we will call this the "message key"), a key to encrypt the message key with (we will call this the "plaintext key"), and one of several "discriminants" which were random letter combinations intended to indicate which of the several sets of daily settings were being used (these were different for each branch of the German military). Both the message key and the plaintext key were used to set the starting positions of the three rotors.

To encrypt a message, an operator would first set up the Enigma machine with that day's daily settings for the specific branch they were operating under. Then, they would choose a plaintext key, and set the Enigma machine's rotors to that key. Next, they would choose a message key, and encrypt the 3 letters of that message key using their Enigma. Before May 1<sup>st</sup>,

1940, the message key was encrypted twice, resulting in 6 characters of ciphertext. After, the message key was only encrypted once, resulting in 3 characters of ciphertext. Once the message key was encrypted, the operator would reset the Enigma rotors to the message key, and encrypt the rest of the message. The resulting ciphertext, prepended with the 3 (or 6) encrypted characters of the message key, would be split into blocks of 5 characters and transmitted over radio, combined with a plaintext recipient, the callsign of the transmitter, the chosen discriminant, and the plaintext key.

On the receiving end, an Enigma operator would look at the discriminant to determine if the daily settings to which his Enigma machine was set up were the same as the ones the transmitter used. If the settings matched, the message was able to be decrypted and the operator would proceed by first setting the Enigma's rotors to the transmitted plaintext key. Using this, the first 3 (or 6) letters of the ciphertext were decrypted to reveal the message key. The operator would then reset the rotors to the position indicated by the decrypted message key and use the machine to decrypt the body of the message.

## **2.2 Enigma Weaknesses**

Bletchley Park was the center of code breaking-activity during most of World War II. The most important ciphers that were broken there during the war were Enigma and the Lorenz cipher. In order to decipher Enigma messages, the daily settings needed to be known. The cryptanalysts focused on several different weaknesses to aid in the search for these daily Enigma settings. In this section, we'll give a short description of many of the weaknesses they attempted to exploit, and in later sections we'll focus on Zygalski sheets and the Bombe, the two exploitation methods we focused on for our implementation.

### **2.2.1 Encrypting the Key Twice**

Before May 1<sup>st</sup>, 1940, established practice among German operators was to encrypt the chosen message key twice, creating a 6 character-long ciphertext that was prepended to the enciphered message. While cryptanalysts did not know what key was enciphered, they could draw conclusions from the patterns found in those 6 character-long ciphertexts. For example, if the same letter (in positions 1 & 4, 2 & 5, or 3 & 6) was enciphered into the same ciphertext letter in both places (e.g. KIEKIE -> AFGPFJ, note the I -> F relation), that would rule out approximately 60% of possible ring settings for that day. Given several of these “females”, as they were called by Bletchley Park cryptanalysts, one could rule out nearly all ring settings, leaving the actual settings for a particular day. More information about this weakness can be found in Section 3 on Zygalski sheets, which were created to exploit this.

### 2.2.2 Cillies

A surprising number of weaknesses in the Enigma-created ciphers were due to operator misuse. Over time, the cryptanalysts at Bletchley Park (and in Poland) noticed many trends that enabled them to more easily guess daily settings. One trend was that some expressions were used very often; for example, “EINS” (German for “One”) occurred often enough that the people at Bletchley Park created an “EINS dictionary” that consisted of all the possible encryptions of “EINS.” Messages were also occasionally re-transmitted using different ciphers from other parts of the military. If the second cipher that was used was already broken, the plaintext of the message was known and could be used to help find the Enigma settings. In some branches of the military there was also a rule that no rotor could be in the same position two days in a row so if the daily settings for the previous day had been found, the search space for the current day’s settings would be reduced by a fair amount.

Other weaknesses were due to the practices that operators would often employ. Operators tended to use keys that were easily guessable and related the plaintext key to the message key. For example, the keyboard on the Enigma machine was QWERTZ so operators tended to use the diagonals as keys. This meant if the plaintext key was “QAY” (the first diagonal), then a good guess for the message key was “WSX” (the second). In addition, for messages that had multiple parts, operators would occasionally not enter a new key for each subsequent part of the message, leaving the rotors as they were at the end of the previous part and using that as the key.

Another instance of operator mispractice was with the key used after setting up the Enigma machine with the daily settings. When setting the ring setting for a particular rotor, it was easiest to hold the indicator up and rotate the alphabet ring to match the ring setting for the day. Then, when the rotors were loaded into the machine, the initial placement of the rotors was usually within a few letters from the ring settings. Instead of choosing a new key, operators would sometimes just use what was showing on the rotors for the key (or a key only a few letters away). Using frequency analysis, the ring settings could be determined from the occurrences of these keys.

Overall, cillies were surprisingly useful to cryptanalysts in reducing the search space for the daily settings. We won’t continue to discuss cillies further because cillies focus on operator misuse which is hard to simulate and we focused on other methods to break the Enigma cipher.

### 2.2.3 The Enigma Machine Itself

The Enigma machine performed a complex substitution cipher with two main characteristics that cryptanalysts used to exploit the Enigma cipher. The first characteristic was that the cipher was symmetric (e.g. if “A” maps to “C”, then “C” must map to “A” at the same position in the Enigma machine). The second was that a letter can never map to itself (e.g. if “A” is in the ciphertext, it could not be an enciphered “A”). Both of these properties helped to develop the bombe, which we’ll discuss later in Section 4.

### 3 Zygalski Sheets

One of the weaknesses previously described was that operators (before May 1<sup>st</sup>, 1940) would encrypt the message key twice and include it with the message so the receiving operator could decrypt the message, which we'll refer to as the "indicator" for that given message. For instance, if the message key was "ABC", the operator would encipher "ABCABC", and the resulting ciphertext might be something like "QTPRFI." However, depending on the ring settings, the result might be something like "PSTPWA" where a specific letter in the key was enciphered to the same letter twice in the indicator (in this case the "A" was enciphered to a "P" both times).

This pair was called a "female" where there was a certain letter in the same position in both encrypted versions of the message key. The example above is called a 1,4 female, as the first and fourth characters are identical. Females like this are only possible in roughly 40% of initial ring setting configurations so if a female was found in a batch of messages, about 60% of the possible configurations for the ring settings can be ruled out.

When multiple messages are sent using the same settings, this information can be combined to rule out increasingly more possible configurations, until only one configuration remains. However, in order to do this, one would need a way of keeping enough information in a way that it can be used to rule out impossible setups. [8]

#### 3.1 Using Zygalski Sheets

The method for accomplishing this is credited to Polish mathematician Henryk Zygalski. The premise was to create a set of sheets for each rotor order possible (6 at the time since there were only 3 rotor options) and in each set, there would be one for each starting letter. A sheet would have a 26-by-26 grid (for the second and third letters) and a hole was punched at positions where a female could occur.

A batch of messages would be used to determine which sheets could be lined up where the set of sheets corresponded to a guess of the leftmost initial ring setting and the rotor order. Once enough sheets (generally 12) were lined up on a lit table, either light would shine through a single hole or no light would shine through. In the former situation, the codebreakers found a possible configuration of the ring settings, and could test it. In the latter, the guess of either rotor order or first ring setting was wrong, and they had to try again with a different initial guess.

This was a very time-consuming project: simply making the sheets took a few months when there were only 3 rotors. The introduction of 2 additional rotors would have increased the work by a factor of 10 (since there would then be 60 possible rotor orders). Even if all the sheets could be manufactured, there were 156 possible guesses for the rotor order and left ring setting.

This meant the process of lining up the sheets might be repeated about 70 times on average before seeing the correct result.

### **3.2 Programmatic Replication**

In order to replicate the use of Zygalski sheets in code, we opted not to create the sheets in advance, but instead generate them only when they were needed, while trying to recover the ring settings from a set of messages. Due to the advantages of modern computers, and the fact that we do not have to physically punch holes, this process is practically instantaneous. This means that ring settings can be recovered very quickly using computers.

In their physical form, Zygalski sheets were meant to be shifted based on the letters in the indicator. In code, we simulate this by using different ring settings to generate the sheet for each message. To do this, for a given run, we first choose a rotor order and left ring setting. Then, using the first indicator letter, the rotor order, and the ring setting, we generate a Zygalski sheet, here represented as a 2-dimensional array keyed by the second and third letters, by running through all the choices of second and third letters and checking if those choices of configuration could generate a female. If so, we mark that spot with true. Otherwise, it is marked false.

For subsequent messages, we generate another sheet with the same rotor order and left ring setting, but use different settings for the other two rings, specifically to “offset” it from the first sheet. For example, if our first message had P and Q as its second and third indicator letters, the second had N and B, and the guess being tested had A as the first ring letter, we would use ring settings A,  $(Z + N - P)$ ,  $(Z + B - Q)$  to generate the sheet that corresponds to the next indicator. When we lay that sheet on top of the first, we logical and every single entry in the 2d-arrays representing the stack of sheets so far and the newly added sheet in order to get a third array representing the resulting stack of zygalski sheets. Anywhere marked true in this stack has holes that line up all the way down, allowing light through, and represents a possible configuration that would produce the intercepted females. [9]

If at any point, there are no longer any truthy grid entries, we know that we have made an error, and change to the next rotor order or ring setting and try again. If we evaluate the batch of messages and end up getting only one truthy entry, we have a possible configuration for the ring settings.

### **3.3 Weaknesses/Problems**

This process cuts away all of the slowness of the manual labor associated with Zygalski sheets. The process becomes a matter of seconds, since 156 different iterations of a loop is trivial for a computer, and the calculations for creating Zygalski sheets are not complicated.

However, the shifts in the use of Enigma to only encipher the message key once essentially made this technique useless by removing females. Here, computing power is irrelevant, as the information that exposed the configuration settings is no longer available.

## 4 The Bombe

Once the Zygalski sheets were no longer useful to Bletchley Park, the cryptanalysts had to come up with some other way of determining the daily settings. Alan Turing and Gordon Welchman separately came up with the idea for a machine that could very quickly go through all possible settings for a particular rotor order and test a set of logical hypotheses. This set of hypotheses was built upon a “crib”. A “crib” is a word or phrase that was likely to be in a particular encrypted message. This crib was matched with a possible location in the ciphertext and used to create a graph where there was an edge for every position a plaintext letter (from the crib) mapped to a ciphertext letter (from the encrypted message). These graphs could have cycles (where one letter maps to another that eventually would map back to the first letter) which were useful in using the message to find the Enigma settings. It is this graph, called a “menu”, that would be put into Turing and Welchman's machine (called the “bombe”, after the Polish “bomby,” another Enigma-breaking machine) and set to run to find possible Enigma settings.

The bombe itself was a giant electro-mechanical device, standing approximately 6 feet tall, 7 feet long, and 2 feet deep. It consisted of 3 banks of 12 Enigma machine clones; each Enigma clone had 3 rotors, arranged vertically, with the topmost rotor corresponding to the rightmost rotor in an Enigma machine. The back of the bombe was a mass of cables, with sockets at the entrance and exit of each Enigma clone, as well as sockets to input and output ports on either side of the bombe. A menu would be connected into the bombe by first setting up several Enigma clones to have offsets specified by the edges in the constructed graph, and then connecting the inputs and outputs of those Enigma clones to other inputs and outputs, as specified by the menu. Each Enigma clone (and thus its input and output cables) consisted of 26 parallel connections.

A single cable represented a letter in the menu, and thus each of the 26 wires in that cable represented possible plugboard connections for that letter. The bombe was, in short, designed to find a trio of ring settings and a set of plugboard assignments that were logically compatible with each other. The user would input a guess into the bombe by flipping on one of the switches on the input side, connecting the input letter to another letter indicating some plugboard pair (e.g. “A” maps to “H”). This switch would apply a voltage to the “H” wire in the “A” cable, and that voltage would propagate through the system. For example, if “A” maps to “H” on the plugboard, and “A” in plaintext corresponds to “C” in ciphertext, and “H” enciphers to “P” through an Enigma machine at that offset, then “C” must map to “P” on the plugboard. This type of logical deduction would continue to happen until the system reached a steady state, in which one of three results were possible:



1. One relay on the output is energized: This would signify that the guess made at the beginning was correct, and the steady state consists of one voltage-carrying wire per cable on the back of the bombe. This is a successful result.
2. All but one relay on the output is energized: This is the complement of result 1, where the guess was incorrect. The wrong hypothesis led to a series of wrong assertions (represented by energized wires) which left only the correct wires un-energized. This is also a successful result because the un-energized relay represents the solution.
3. All relays can be energized: This signifies that there is effectively no stable cycle for this setting of the Enigma, and it can be ruled out.

The bombe, once set up and started, finds a steady state for the first Enigma setting, and checks to see if all the relays are energized. If they are, it rotates the top rotors on each Enigma clone in its menu, and tries again. The bombe is configured to stop if not all relays are energized, and these “stops” are then notated down and analysed further, and the bombe is re-started.

A bombe is likely to stop more than once throughout its run, and only one of these stops is the correct one. The others are known as “false stops”, and can be identified through a couple methods:

1. The first is to identify all of the plugboard pairs in the menu. These were noted down from which relays were energized when the bombe stopped. In a false stop, these plugboard pairings were likely to conflict (eg. both “C” and “D” are paired with “H”).
2. From there, if the plugboard pairs did not conflict, the settings and plugs would be setup on a normal Enigma machine replica and used to decrypt the entire message. If it looked like plausible German (allowing for the rest of the unknown plugboard pairs), the stop was likely correct.

Menus were often chosen to make the bombe stop as rarely as possible. In general, it was favorable for a menu to have as many letters as possible, and as many cycles as possible, as these would reduce the number of false stops. It was important, however, to have a menu whose span (the range in Enigma offsets throughout the menu) was not too large, as that would reduce the risk of having a turnover (where the middle rotor rotates) in the middle. Having a turnover happen somewhere in a menu would invalidate any stops produced by the bombe, as it did not handle turnovers.

Soon after the idea for the bombe was conceived, Welchman made the addition of the “diagonal board”. This was an additional set of connections on the back of the bombe, designed to further reduce the number of false stops. Its guiding principle was that the plugs in the plugboard worked reciprocally---if “E” was connected to “M”, then “M” must be connected to “E”. As such, the diagonal board connected the “E” wire in the “M” cable to the “M” wire in the “E” cable (and had similar connections for every other pair of letters). The diagonal board slightly reduced the importance of cycles in the menus, as it introduced (smaller) cycles into the workings of the bombe.

## **4.1 The Bombe In Code**

### **4.1.1 Making Menus**

Menus were made by matching “cribs” to enciphered messages that would likely contain that crib phrase or word. Instead of considering what would be “likely” in code, we made a menu for every possible location that the crib could appear in the message. A location would not be possible if any of the letters in the crib appeared in the same position in the message since letters could not be enciphered to themselves. For example, if the crib was 12 letters long, we checked every set of 12 letters in the message to see if any of the letters matched, and, if they didn’t, produced a menu for that location. The menus themselves were just graphs where every letter in the crib and message were nodes and the edges were the positions in which they “matched” in the crib placement. For example if the crib had an “E” in the third position and the message had a “L” in the third position, then an edge would be added to the graph that connected “E” and “L” at position 3. In order to make menus that work with the bombe that does not contain a diagonal board, the entire menu graph must be connected so implementation of this was also added.

As discussed in the conclusion, our implementation of the bombe was not as successful as the staff at Bletchley Park were at finding Enigma settings. One possible improvement that we did not get to implement was only using “good” menus. This would entail using Alan Turing’s analysis on how many stops there were estimated to be for a given menu and only choosing menus that resulted in few stops. The aspects of the graph needed to evaluate this would be the number of total letters in the menu and the number of cycles that appear in the graph.

### **4.1.2 Running Menus through the Bombe**

In code, running the bombe on a single rotor order and setting is represented as a graph. The nodes of the graph are called the “state” which represent the voltage of every wire in every cable that would be present in the bombe. An “on” wire is represented by a “true” value in the state. The edges of the graph are state transitions which represent the Enigma clones in the bombe. They are triggered individually when one side of an edge (one node)’s state is changed. When we implemented the diagonal board, this was also treated as a state transition, that was triggered whenever an edge was relaxed. Then, in order to find the steady state as the bombe did, we relaxed the graph, edge by edge until this relaxation resulted in no change in the state. If all of the wires were “on,” the bombe would discard this state and setting and move on to the next one. Otherwise, this state represented a “stop” and the result was forwarded on to the next stage as the bombe immediately moved on to the next setting. The results from the bombe included possible plugboard pairs as well as an Enigma setting or a total rotor “offset.” This offset would be the combination of both the rotor settings and the message key for the specific message run on the bombe.

### 4.1.3 Checking Stops

All of the results from the bombe needed to be checked to see if they were “false stops.” The first check was to make sure that the plugboard pairs that were returned were “legal.” This meant that no two letters were paired with the same letter (e.g. “S” and “H” both paired with “X”) and there was a maximum of 10 plugboard pairs and 6 “self-steckered” letters. For our implementation of the bombe, we made the simplifying assumption that every message would have exactly 10 plugboard pairs. If the results from the bombe passed both of these checks, the plugboard pairs were “legal” and further analysis was needed to determine if this was a false stop or the correct settings.

### 4.1.4 Creating Messages

In our ambitious attempt to mimic processing an entire day’s intercepted messages, we created thousands of messages to use as input to our automated bombe process (see section 4.1.5). As the plaintext of these messages we used several classic books from Project Gutenberg<sup>1</sup>. We read in the file of the book and enciphered each line with the same day’s Enigma settings and individual plaintext and message keys. The plaintext keys were included with the enciphered messages which were all written to another file.

### 4.1.5 Automating the Process

To automate the bombe process, we first loaded the input file and the guessed crib phrase. Then, we created goroutines (threads) for each “worker” in the process. The first set of workers created menus from the crib and encrypted messages as explained in Section 4.1.1. These workers sent the menus along with the messages they were created from to a bombe runner. This bombe runner would schedule incoming menus on whatever bombe machines (also goroutines) were available. The bombes would run as described in Section 4.1.2 and the results would be sent to the checker workers. The checkers would forward on only legal bombe results as described in Section 4.1.3 back to the main thread which would collect these results and perform post-processing.

In the system that we implemented, the number of menus and results that we received were too numerous to collect and post-process in a reasonable amount of time. Unfortunately, we were not able to include any other successful checking mechanisms in our code to reduce the number of possible results to find the Enigma settings for the day. We tried to research more about how to determine if a stop was correct and if so, how to find the ring settings from the result from the bombe (which only returned the aforementioned “offset”). The few ideas that we came up with, but did not have time to implement are the following:

---

<sup>1</sup> <https://www.gutenberg.org>

1. To determine if a stop is correct, we currently attempt to decrypt the message and display it to the user, so they can tell us if it looks like semi-legible plaintext. Ideally, the computer could tell if the text was semi-legible without user input to speed up the process. We had the idea to check for semi-legible text programmatically, by importing a dictionary of whatever language the plaintext was in, and taking Hamming distance measurements between the decrypted message and common words. After considering this Hamming distance to be small enough to reasonably be a match to the ciphertext, we could determine the remaining plugboard pairs by using the word the ciphertext is closest to as a new menu and repeating the process until all plugboard pairs were found.
2. When the bombe stops, it outputs the Enigma settings that lead to the stop. We called this the “quote key” (spoken form of ‘key’), as it was neither the ring settings for the day nor the actual message key for the message used in the menu, but rather the composition of the two: “message key” - “ring setting” = “quote key”. We thought of a possible way to recover the actual ring settings from this quote key, rotor by rotor:
  - a. The rightmost rotor is the easiest to recover the ring setting for. Any message longer than 26 characters is guaranteed to have a middle rotor turnover in it (see Section 2 for more information about turnovers). As such, we can look for where that turnover occurs, and see what the quote key is at that point. Because we know which the rotor order from the bombe result, we know its default turnover position (with the default ring setting), and can recover the ring setting by looking at that difference (actual turnover position - expected turnover position).
  - b. The middle rotor ring setting is recoverable in the same way, only it is a lot more rare to find a message with a leftmost rotor turnover in it. That said, Bletchley park processed thousands of messages a day, and as a leftmost rotor turnover happens every 676 characters, there were most likely at least a few messages containing leftmost rotor turnovers.
  - c. The leftmost rotor’s ring setting is the hardest to recover, as it does not influence any turnovers (there is no rotor to the left of the leftmost rotor). As such, the method we came up with, involved finding the other two ring settings and simply guessing the third. We planned to test our 26 guesses on the key encryption for a message we had cracked using the bombe. Given the plaintext key, the encrypted message key, and the lower two letters of the plaintext message key (recovered by adding the discovered right and middle rotor ring settings to the respective letters of the quote key), we can simply decrypt the encrypted message key with each possible ring setting for the leftmost rotor, stopping when we find one that decrypts the lower two letters correctly.

## 5 Conclusion

Once implemented, we attempted to run our entire decryption system on some test-encrypted messages. It took a surprisingly long time! Part of this was because we did not implement a lot of the prioritizing that the cryptanalysts at Bletchley Park did, both in terms of choosing menus that were less likely to produce false stops as well as in terms of choosing stops that were more likely to be correct. In addition to this our bombe implementation ran (on a 2013 MacBook Air) about as fast (throughput and latency-wise) as one dedicated bombe.

On one hand, it is pretty impressive that a tiny laptop can rival a 1-ton machine in speed. On the other hand, this is not as impressive as we expected, given that the 1-ton machine was created 70 years ago and technology has improved a significant amount since then. Modern computers can run billions of operations in a second, but despite this amazing increase in power, human ingenuity and specialized hardware can still compete.

We think that overall, the conclusion to be made here is that dedicated hardware has an innate advantage over all-purpose hardware (the bombe's instantaneous steady-state working faster than our relaxation method). In order to outpace the efforts of codebreakers at Bletchley Park, we would have to exercise similar amounts of ingenuity and effort, optimizing our approaches algorithmically, and possibly creating specialized hardware. To do so would be using the same techniques and effort they did, which definitely speaks to the conclusion that simply having faster hardware does not invalidate the efforts of the researchers of the past. While Enigma would not be an effective cryptography standard today, it remains a nontrivial challenge to break, even in the face of modern technology.

## References

- [1] D. Davies and U. of London. Royal Holloway, The Bombe - a Remarkable Logic Machine. Special lecture series, Royal Holloway, University of London, 1999.
- [2] Wikipedia contributors, “Enigma machine — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 10-May-2018].
- [3] Wikipedia contributors, “Cryptanalysis of the enigma machine — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 10-May-2018].
- [4] Wikipedia contributors, “Enigma rotor details — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 10-May-2018].
- [5] Wikipedia contributors, “Bombe — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 10-May-2018].
- [6] Tony Sale, “Alan turing, the enigma and the bombe.,” 1991. [Online; accessed 14-May-2018].
- [7] Frank Carter, “The turing bombe.” [Online; accessed 08-May-2018].
- [8] Bill Casselman, “The polish attack on enigma ii: Zygalski sheets,” 2013. [Online; accessed 13-May-2018].
- [9] Tony Sale, “The breaking of enigma by the polish mathematicians,” 1991. [Online; accessed 13-May-2018].
- [10] G. Welchman, The Hut Six Story: Breaking the Enigma Codes. M & M Baldwin, 1997.
- [11] D. W. Davies, “Effectiveness of the diagonal board,” *Cryptologia*, vol. 23, no. 3, pp. 229–239, 1999.
- [12] J. Wright, “The turing bombe victory and the first naval enigma decrypts,” *Cryptologia*, vol. 41, no. 4, pp. 295–328, 2017.