ENIGMA: SIMULATION AND ANALYSIS OF SECURITY

by

Elisabeth Joanne Moore

Honors Thesis

Appalachian State University

Submitted to the Department of Mathematical Sciences
and The Honors College
in partial fulfillment of the requirements for the degree of

Bachelor of Science

May, 2017

Approved by:

[illegible]

Rick Klima, Ph.D., Thesis Director

[illegible]

Donna Lillian, Ph.D., Second Reader

[illegible]

Vicky Klima, Ph.D., Honors Director, Department of Mathematical Sciences

Ted Zerucha, Ph.D., Interim Director, The Honors College

Abstract

During World War II, the Enigma machine was used as a message encryption tool by the German military. While it was believed that the Enigma machine produced an unbreakable code, this turned out to not be the case. This thesis begins with a summary of six versions of the Enigma, five that were actually used in the field, and a theoretical version that would have maximized security. An analysis of security is done, in which combinatorics is used to count the number of possible unique settings for each version of the machine. Estimates are also provided for the time required to break each version of the machine using a brute-force attack with various processing speeds that were achievable at various times in history. Finally, descriptions are included of three Enigma simulators constructed for this thesis. These simulators include a three-dimensional paper simulator, a digital simulator written using the Python programming language, and an electromechanical simulator built to resemble an actual Enigma machine.

Contents

1 Background

The original Enigma machine was invented by Arthur Scherbius (Figure 1), a German electrical engineer, late in World War I [4]. Scherbius's design was for an electromechanical, rotor-driven encryption machine, the first of its kind to be patented. After many design iterations, Scherbius started marketing his most successful design, the Enigma D, to businesses throughout Germany [7]. He also marketed the machine to the German military, which declined it at first. It was eventually picked up by the German military, just over a decade before the start of World War II, however, when the Germans discovered their codes from World War I had been broken [4].

Figure 1: *Arthur Scherbius; image reproduced with permission of Crypto Museum* [7]

Poland, realizing the likelihood of another war, began work on decrypting the Enigma. Not having access to the military version of the machine that was in development, the three members of the Polish Cipher Bureau, Marian Rejewski (Figure 2), Henryk Zygalski, and Jerzy Rózyki, focused their efforts on decrypting the commercial one. Using information about the workings of the Enigma and dozens of intercepted messages, the Cipher Bureau developed a machine called the *Bomba kryptologiczna* (Figure 3) that could decrypt the commercial Enigma [5, 7].

Figure 2: *Marian Rejewski; image reproduced from WikiCommons* [14] *under CC-BY-SA*

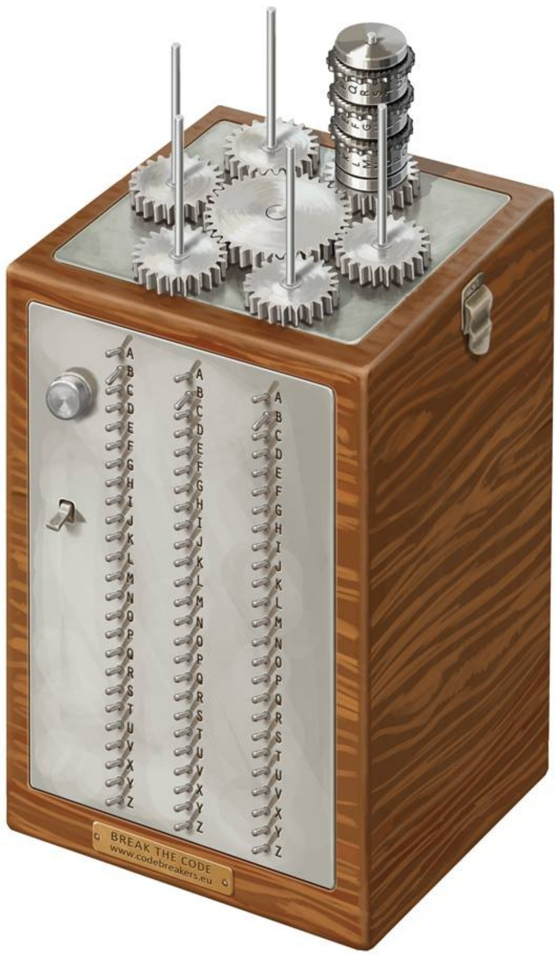


Figure 3: *The Bomba; image reproduced with permission of Crypto Museum* [7]

Poland continued its work on the Enigma machine until Germany's invasion at the beginning of World War II. In order to preserve the work that had been done up to that point, the Poles smuggled all of their collected work on the machine to Great Britain via France [7]. The British Cipher Bureau, located in Bletchley Park, England (Figure 4) and led by Alan Turing (Figure 5), used the information provided by the Polish Cipher Bureau to build the *Bombe* (Figure 6), a machine capable of decrypting the military version of the Enigma [5].

Figure 4: *Bletchley Park, the location of British Enigma decryption work; image reproduced with permission of Bletchley Park Trust* [2]

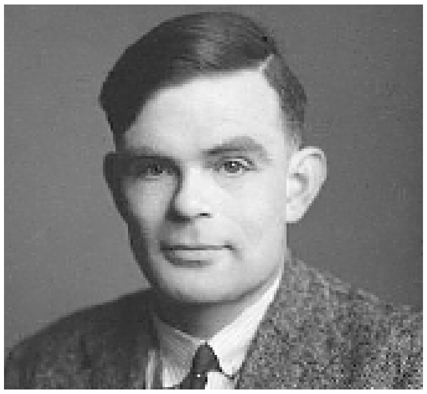


Figure 5: *Alan Turing; image reproduced with permission of Crypto Museum* [7]

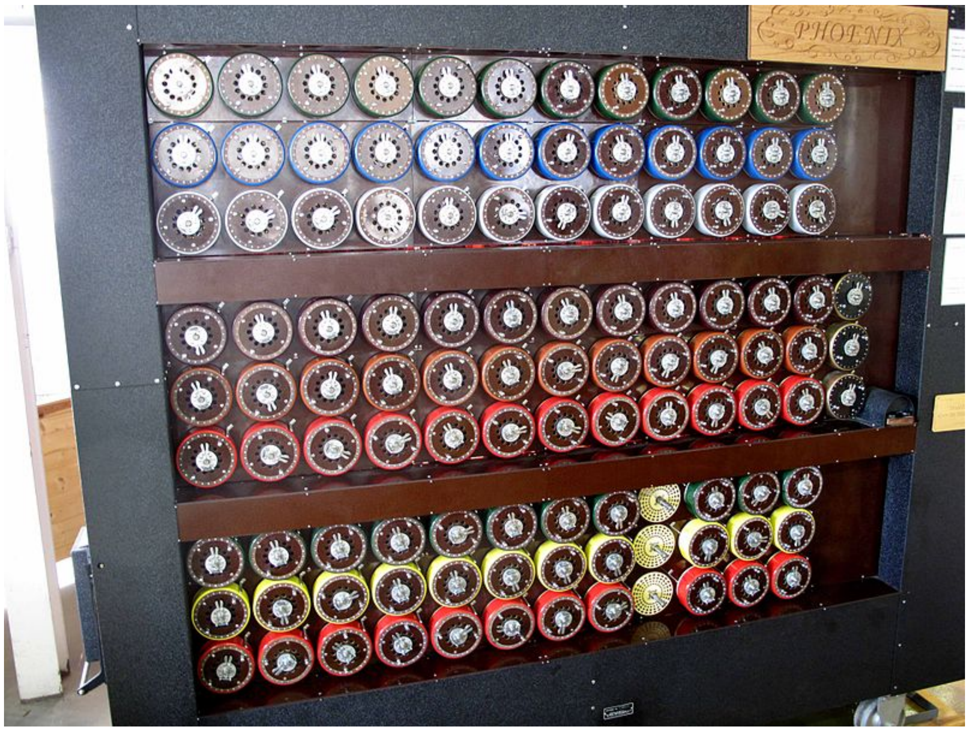


Figure 6: *The Bombe, as designed by Alan Turing; image reproduced from WikiCommons* [10] *under the Creative Commons CC0 1.0 Universal Public Domain Dedication*

2 Machine Description

The Enigma machine (Figure 7) as it was used used throughout World War II consisted of several components: a keyboard for entering an input message, a plugboard, rotors, and a reflector, all through which the input message was encrypted or decrypted, and a lampboard for displaying the output message.

When a letter was pressed on the keyboard, an electric current was transmitted through the machine that eventually became the output letter. The current first passed through the plugboard, where a series of sockets corresponded to the letters on the keyboard (Figure 8). The sockets could be left open or connected to each other in pairs via a collection of cables. If two sockets were connected, either letter connected would be converted to the other when the current passed through the plugboard. If a letter passed through an open socket, it would leave the plugboard unchanged [4].

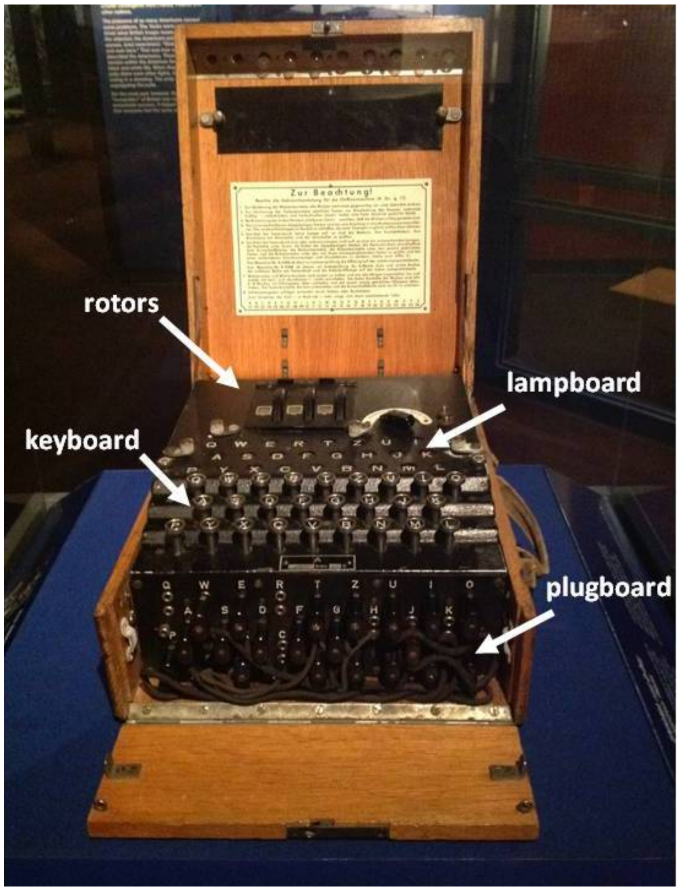


Figure 7: *Enigma machine with labeled components; image reproduced with permission of the National WWII Museum* [6]

After the plugboard, the current would pass through a series of wired wheels, called rotors (Figure 9). Each rotor had 26 electrical contacts on each face, one to represent each letter. The contacts on one side were wired to the contacts on the other, but not straight across, scrambling the letters as they passed through a rotor. In this way, a letter would be swapped for another every time it passed through a rotor. Rotors could be rotated into any of 26 positions before being inserted into the machine and a ring of letters around each rotor could also be rotated into any of 26 positions without the wiring also rotating. The physical wiring of the rotors was fixed, though, and only a limited number of different rotor wirings were produced. A series of multiple rotors was used in the machine, lined up from left to right, resulting in letters being swapped for others multiple times during one round of encryption or decryption [4].

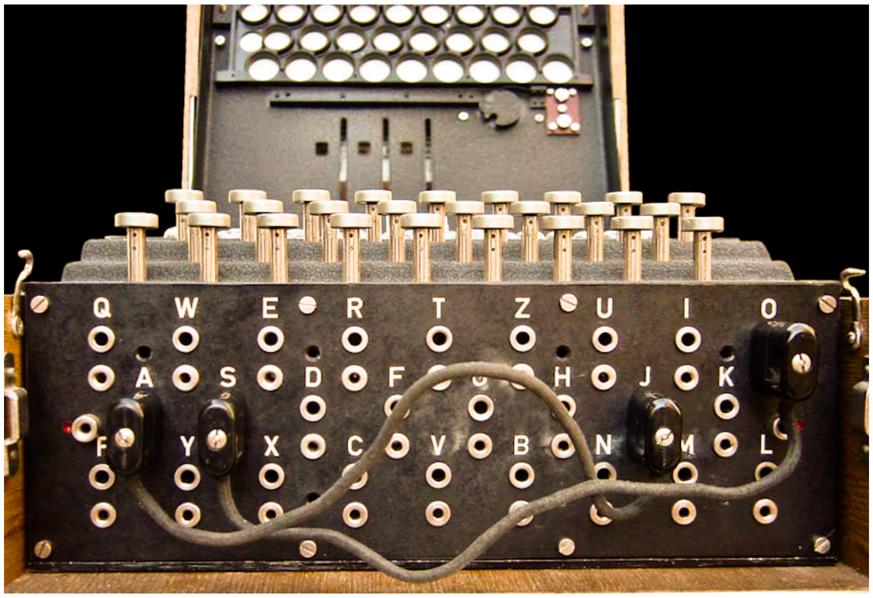


Figure 8: *Close-up of a plugboard; image reproduced from WikiCommons* [10] *under the GNU Free Documentation License*

Every time a key was pressed on the keyboard, the far right rotor would rotate forward by one letter. Every time that rotor rotated 26 times, or through the whole alphabet, the rotor to its left would rotate forward by one letter. Similarly, every time this rotor rotated through the whole alphabet, the rotor to its left would rotate forward by a letter. This changed the operation of the machine from being a monoalphabetic cipher to being a polyalphabetic cipher, meaning that the same input letter would not necessarily produce the same output letter every time, and that different input letters could produce the same output letter. This change resulted in the Enigma being much more difficult to break than substitution ciphers that preceded it [4].

After the current had passed through all of the rotors, it would enter a reflector. This was essentially half of a rotor, a wired wheel with contacts on only a single side. Every letter on the reflector was connected to another, resulting in 13 connections. Similarly to the plugboard, the connected letters would be swapped for each other by the wiring. Like the rotors, the wiring was fixed and reflectors with only a limited number of different wirings were ever produced [4].

The current would pass through the rotors in reverse order after the reflector, and then again through the plugboard, before finally being displayed through the lampboard as the final output.

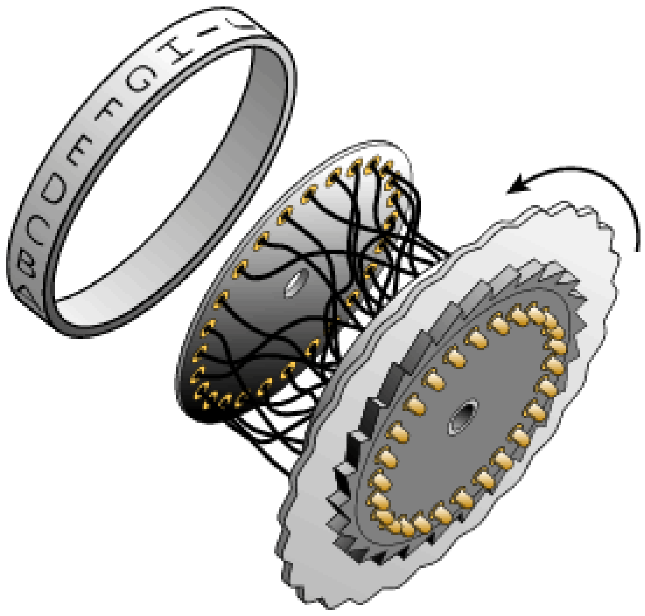


Figure 9: *Exploded view of an Enigma rotor; image reproduced with permission of Crypto Museum* [7]

This meant that, in total, each letter would pass through the plugboard and rotors twice and the reflector a single time. Figure 10 shows an example of the path current representing a single letter would follow through the machine during encryption or decryption [4].

It it important to note that the symmetry of the path followed by the current through the machine meant that passing an encrypted message back through the machine would decrypt the message. Using the same initial configuration and inputting the encrypted message would result in the message being decrypted as it passed through the machine, allowing the decrypted message to be read off from the lampboard, one letter at a time, just as the encrypted message was read from the lampboard starting with the same initial configuration of the machine. It is also important to note that even with the limited number of different rotors and reflectors that were actually produced, this along with the large number of possible wirings for the plugboard produced a vast number of possible initial configurations for an Enigma.

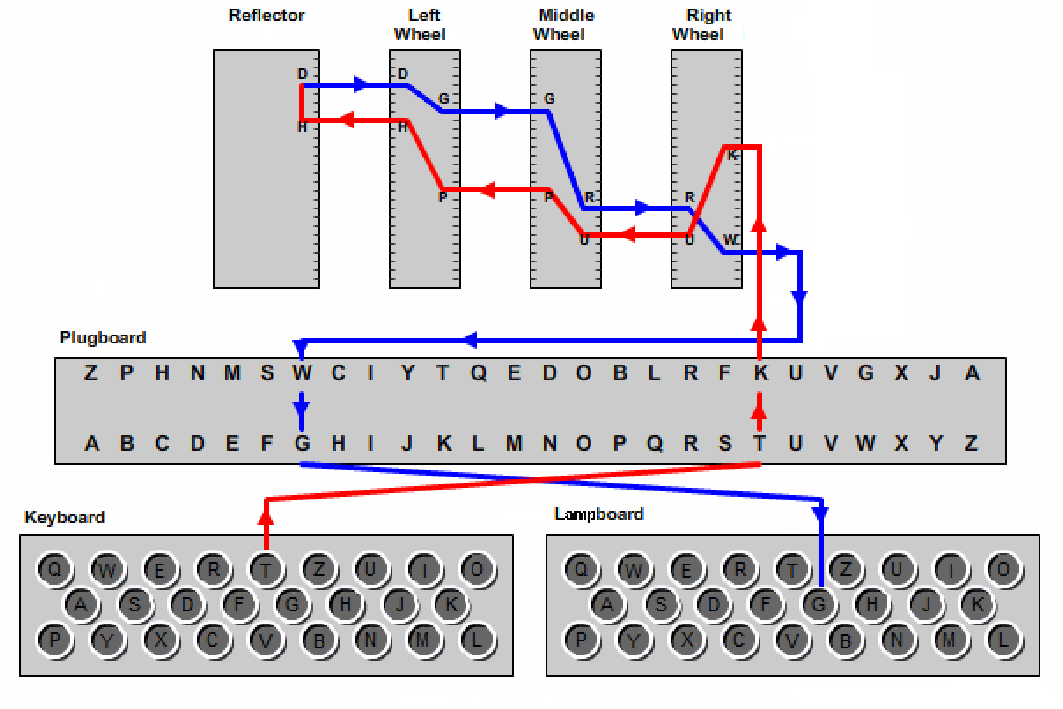

Reflector
Left Wheel
Middle Wheel
Right Wheel
Plugboard
Z P H N M S W C I Y T Q E D O B L R F K U V G X J A
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Keyboard
Lampboard

Figure 10: *Example of path followed by current during encryption or decryption; figure created by the author*

3 Analysis of Security

During World War II, the Germans were convinced of the Enigma's unbreakability. This was due to the massive number of initial configurations produced by the various settings on the machine, each of which would produce a distinct output. Even so, as the war progressed, the Germans made improvements to the machine that increased the number of initial configurations, and, therefore, its security.

The Enigma D (Figure 11) was released in 1926. It featured three interchangeable rotors as well as a single reflector, all of which could be rotated before being inserted into the machine [7]. This resulted in a total of 4.8×10^{10}, or 48,190,861,056 to be exact, possible initial configurations. To see how this value can be determined, consider the following:

- Rotor arrangements: $P(3,3) = 6$
- Rotor and reflector rotations: $(26^3) \cdot 26 = 17{,}576 \cdot 26 = 456{,}976$

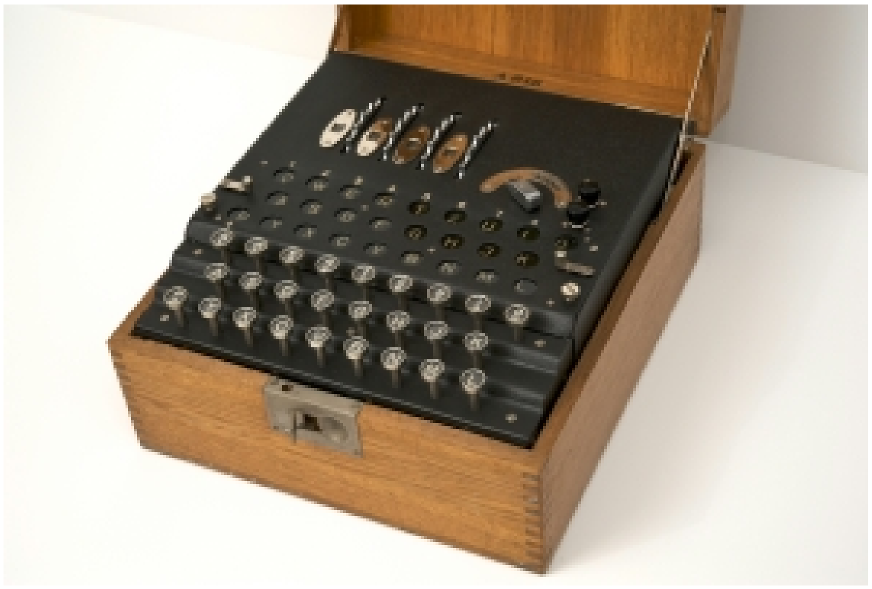


Figure 11: *Enigma D; image reproduced with permission of Crypto Museum* [7]

- Ring settings: 26^3 = 17,576
- Total: $6 \cdot 456{,}976 \cdot 17{,}576 = 48{,}190{,}861{,}056$.

In 1932, the *Wehrmacht* Enigma (Figure 12), also known as the Services Enigma or Enigma I, was introduced by the German military. It featured notable improvements over the commercial design, mainly the addition of the plugboard and two reflectors to select from. However, unlike Scherbius's design, these reflectors were fixed and could not be rotated before being inserted. The rotors, however, could be rotated. In typical military usage, 10 cables were used on the plugboard to connect 20 sockets [7]. This gave a total of 5.6×10^{23}, or 558,785,174,101,703,220,672,000 to be exact, possible initial configurations. To see how this value can be determined, consider the following:

- Plugboard socket selections: $C(26, 20)$ = 230,230
- Plugboard plug configurations: $19 \cdot 17 \cdot 15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 654{,}729{,}075$
- Rotor arrangements: $P(3, 3) = 6$

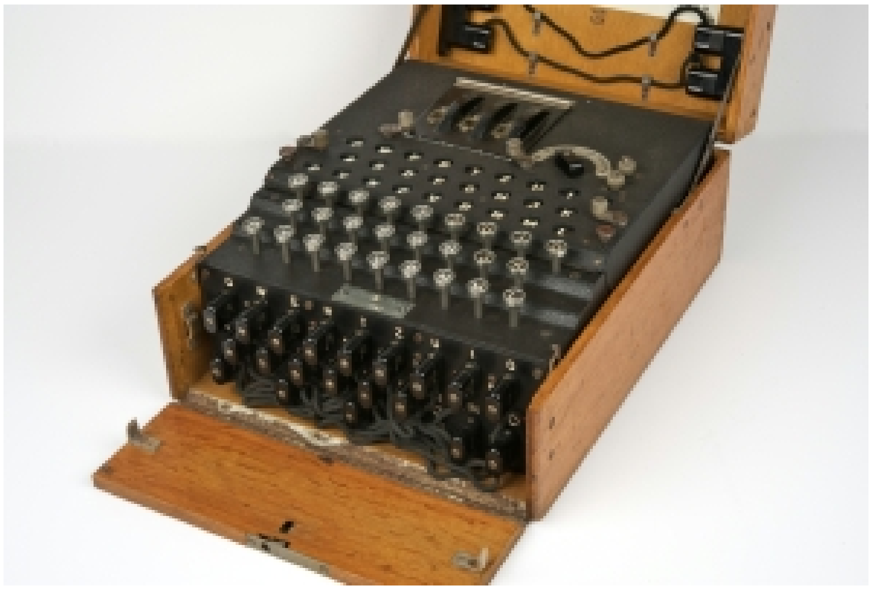


Figure 12: *Enigma I; image reproduced with permission of Crypto Museum* [7]

- Reflector selection: $C(2, 1) = 2$
- Rotor rotations: $26^3 = 17{,}576$
- Ring settings: $26^3 = 17{,}576$
- Total: $230{,}230 \cdot 654{,}729{,}075 \cdot 6 \cdot 2 \cdot 17{,}576 \cdot 17{,}576 = 558{,}785{,}174{,}101{,}703{,}220{,}672{,}000$.

In 1938, the security of the *Wehrmacht* Enigma was further increased with the addition of two additional rotors to choose from, giving a total of five rotors from which three were selected at a time [7]. This gave a tenfold increase in security, since the number of possible rotor arrangements increased from $P(3, 3) = 6$ to $P(5, 3) = 60$, bringing the total number of possible initial configurations up to 5.6×10^{24}, or 5,587,851,741,017,032,206,720,000 to be exact.

The *Kriegsmarine* Enigma or M1 Enigma, specific to the German Navy, was developed in 1932. It was similar to the Enigma I, featuring only slight changes in the design of the carrying case [7]. Hence, the number of initial configurations possible with the the M1 was the same as that with the Enigma I.

The Enigma M4 (Figure 13), also specific to the German Navy, replaced the M1 in 1942,

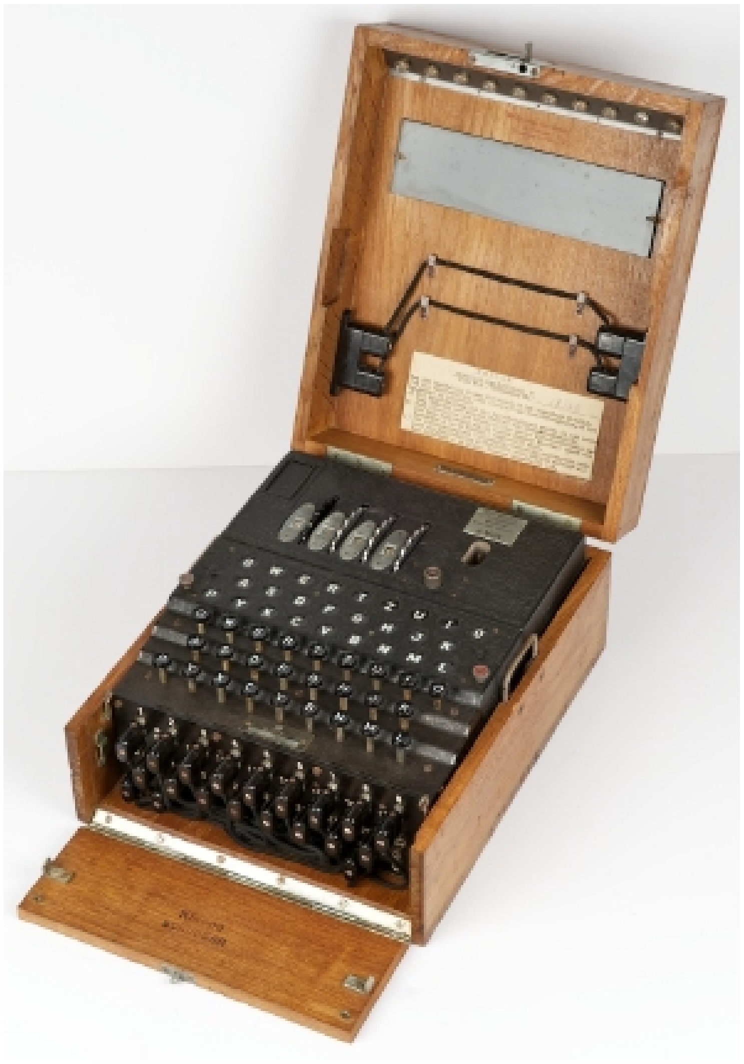


Figure 13: *Enigma M4; image reproduced with permission of Crypto Museum* [7]

in response to German suspicions that the Allied Powers had found a way to read messages encrypted with the Enigma. The M4 added three more rotors to the five that could be selected from for the *Wehrmacht*, making for a total of eight rotors to choose from. There was also an additional slot for a thinner rotor, of which two were provided. All four of these rotors had rings which could be rotated, and the rotors themselves could also be rotated before being inserted into the machine [7]. Additionally, it had two different reflectors from the *Wehrmacht* Enigma, although since the *Wehrmacht* pair were not used, this did not impact the number of initial configurations possible. The M4 saw a huge improvement over the *Wehrmacht* Enigma, allowing for 4.2×10^{28}, or 42,306,743,101,588,154,243,518,464,000 to be exact, possible initial configurations. To see how this value can be determined, consider the following:

- Plugboard socket selections: $C(26, 20) = 230{,}230$
- Plugboard plug configurations: $19 \cdot 17 \cdot 15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 654{,}729{,}075$
- Rotor arrangements: $P(8, 3) = 336$
- Thin rotor arrangement: $P(2, 1) = 2$
- Reflector selection: $C(2, 1) = 2$
- Rotor rotations: $26^4 = 456{,}976$
- Ring settings: $26^4 = 456{,}976$
- Total: $230{,}230 \cdot 654{,}729{,}075 \cdot 336 \cdot 2 \cdot 2 \cdot 456{,}976 \cdot 456{,}976$
  $= 42{,}306{,}743{,}101{,}588{,}154{,}243{,}518{,}464{,}000.$

The Germans gave strict instructions for the use of the Enigma machine in the field, for example requiring the use of exactly 10 plugboard cables, which of course limited the number of possible initial configurations. If these restrictions had been removed and a theoretical machine with the maximum security possible had been employed, the number of possible initial configurations for the three-rotor version of the Enigma would have increased to 2.2×10^{117}, or to be exact, the following number:

2,219,905,255,326,754,558,321,796,279,240,740,945,784,849,756,500,641,611,
360,181,393,092,780,349,124,164,128,934,846,831,282,900,646,672,465,920,
000,000,000.

Although the plugboard could accommodate anywhere from 0 to 13 cables, the Germans limited the number used in the field to exactly 10 to simplify the key information that had to be indicated for each day's initial configuration. Allowing for any number of cables from 0 to 13 would obviously have increased the number of possible initial configurations of just the plugboard. With exactly 10 cables, the number of possible configurations of just the plugboard results from multiplying the number $C(26, 20) = 230{,}230$ of ways to choose 20 sockets to connect with 10 cables times the number $19 \cdot 17 \cdot 15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 654{,}729{,}075$ of ways to connect the 20 sockets after they have been chosen. This value is $230{,}230 \cdot 654{,}729{,}075 = 1.5 \times 10^{14}$, or 150,738,274,937,250 to be exact. To find the number of possible configurations of just the plugboard without the number of cables used dictated, we must count the number of ways to configure the plugboard for each number of cables from 0 through 13 separately, and then add these results. The outcome of adding these results is 5.3×10^{14}, or 532,985,208,200,576 to be exact. To see how this value can be determined, consider the following:

- For 0 cables:

  Number of ways to choose the sockets: $C(26, 0) = 1$

  Number of ways to connect the sockets: 1

  Total: $1 \cdot 1 = 1$

- For 1 cable:

  Number of ways to choose the sockets: $C(26, 2) = 325$

  Number of ways to connect the sockets: 1

  Total: $325 \cdot 1 = 325$

- For 2 cables:

  Number of ways to choose the sockets: $C(26, 4) = 14{,}950$

  Number of ways to connect the sockets: $3 \cdot 1 = 3$

  Total: $14{,}950 \cdot 3 = 44{,}850$

- For 3 cables:

Number of ways to choose the sockets: $C(26, 6) = 230{,}230$

Number of ways to connect the sockets: $5 \cdot 3 \cdot 1 = 15$

Total: $230{,}230 \cdot 15 = 3{,}453{,}450$

- For 4 cables:

  Number of ways to choose the sockets: $C(26, 8) = 1{,}562{,}275$

  Number of ways to connect the sockets: $7 \cdot 5 \cdot 3 \cdot 1 = 105$

  Total: $1{,}562{,}275 \cdot 105 = 164{,}038{,}875$

- For 5 cables:

  Number of ways to choose the sockets: $C(26, 10) = 5{,}311{,}735$

  Number of ways to connect the sockets: $9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 945$

  Total: $5{,}311{,}735 \cdot 945 = 5{,}019{,}589{,}575$

- For 6 cables:

  Number of ways to choose the sockets: $C(26, 12) = 9{,}657{,}700$

  Number of ways to connect the sockets: $11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 10{,}395$

  Total: $9{,}657{,}700 \cdot 10{,}395 = 100{,}391{,}791{,}500$

- For 7 cables:

  Number of ways to choose the sockets: $C(26, 14) = 9{,}657{,}700$

  Number of ways to connect the sockets: $13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 135{,}135$

  Total: $9{,}657{,}700 \cdot 135{,}135 = 1{,}305{,}093{,}289{,}500$

- For 8 cables:

  Number of ways to choose the sockets: $C(26, 16) = 5{,}311{,}735$

  Number of ways to connect the sockets: $15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 2{,}027{,}025$

Total: 5,311,735 · 2,027,025 = 10,767,019,638,375

- For 9 cables:

  Number of ways to choose the sockets: $C(26, 18) = 1{,}562{,}275$

  Number of ways to connect the sockets: $17 \cdot 15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 34{,}459{,}425$

  Total: 1,562,275 · 34,459,425 = 53,835,098,191,875

- For 10 cables:

  Number of ways to choose the sockets: $C(26, 20) = 230{,}230$

  Number of ways to connect the sockets: $19 \cdot 17 \cdot 15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 654{,}729{,}075$

  Total: 230,230 · 654,729,075 = 150,738,274,937,250

- For 11 cables:

  Number of ways to choose the sockets: $C(26, 22) = 14{,}950$

  Number of ways to connect the sockets: $21 \cdot 19 \cdot 17 \cdot 15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 13{,}749{,}310{,}575$

  Total: 14,950 · 13,749,310,575 = 205,552,193,096,250

- For 12 cables:

  Number of ways to choose the sockets: $C(26, 24) = 325$

  Number of ways to connect the sockets: $23 \cdot 21 \cdot 19 \cdot 17 \cdot 15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 316{,}234{,}143{,}225$

  Total: 325 · 316,234,143,225 = 102,776,096,548,125

- For 13 cables:

  Number of ways to choose the sockets: $C(26, 26) = 1$

  Number of ways to connect the sockets: $25 \cdot 23 \cdot 21 \cdot 19 \cdot 17 \cdot 15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1 = 7{,}905{,}853{,}580{,}625$

  Total: 1 · 7,905,853,580,625 = 7,905,853,580,625.

Thus the total number of possible configurations of the entire plugboard with no restrictions on the numbers of cables used is the following number:

1 + 325 + 44,850 + 3,453,450 + 164,038,875 + 5,019,589,575 + 100,391,791,500

+ 1,305,093,289,500 + 10,767,019,638,375 + 53,835,098,191,875

+ 150,738,274,937,250 + 205,552,193,096,250 + 102,776,096,548,125

+ 7,905,853,580,625

= 532,985,208,200,576.

For the *Wehrmacht* Enigma late in the war, 5 rotors were provided, from which 3 were selected each time the machine was used. Each rotor had to be hand-built, an extremely time-consuming and difficult process. In order to speed up production of Enigma machines, the number of rotors available was limited so that the machines could be produced more rapidly. Many more rotors were theoretically possible, though, 26! = 403,291,461,126,605,635,584,000,000 to be exact, since A could be wired to any of the 26 letters in the alphabet, B could be wired to any of the remaining 25 letters, and so forth.

The same principle applied to reflectors, with some slight modifications. Specifically, reflector contacts were always wired in pairs so that a letter could not be wired to itself. A could be wired to any of the remaining 25 letters; B, or C if A had been wired to B, could be wired to any of the 23 letters that now remained, and so on. Thus, the number of reflectors that were theoretically possible was $25 \cdot 23 \cdot 21 \cdot 19 \cdot 17 \cdot 15 \cdot 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 3 \cdot 1$ = 7,905,853,580,625, although only two were provided. Furthermore, although German operating instructions did not allow for rotation of the reflector before it was placed in the machine, nothing in the mechanism prevented this.

Combining all of these variable components in an Enigma results in the previously-noted theoretical number of possible initial configurations for the three-rotor version of the machine, 2.2×10^{117}, or to be exact, the following number:

2,219,905,255,326,754,558,321,796,279,240,740,945,784,849,756,500,641,611, 360,181,393,092,780,349,124,164,128,934,846,831,282,900,646,672,465,920,

000,000,000.

To see how this value can be determined, consider the following:

- Plugboard: 532,985,208,200,576
- Rotor arrangements: $P(403{,}291{,}461{,}126{,}605{,}635{,}584{,}000{,}000, 3)$
  = 65,592,937,459,144,468,297,405,473,480,371,753,615,896,841,298,988,710,328,
  553,805,190,043,271,168,000,000
- Reflector selection: 7,905,853,580,625
- Rotor and reflector rotations: $26^4 = 456{,}976$
- Ring settings: $26^3 = 17{,}576$
- Total: 532,985,208,200,576 · 65,592,937,459,144,468,297,405,473,480,371,753,615,
  896,841,298,988,710,328,553,805,190,043,271,168,000,000 · 7,905,853,580,625 ·
  456,976 · 17,576

  = 2,219,905,255,326,754,558,321,796,279,240,740,945,784,849,756,500,641,611,
  360,181,393,092,780,349,124,164,128,934,846,831,282,900,646,672,465,920,
  000,000,000.

4 Decryption Speeds

Despite the fact that the Enigma was cracked by the Allies by the end of World War II, it could only be decrypted with the use of the *Bombe*. Only a few *Bombe* units were built, meaning that despite the breaking of the Enigma machine, the ability to make use of this was limited to those who had access to a *Bombe*. However, using the *Bombe* still took a significant amount of information. Specifically, one had to have information that would help point to the initial configuration. Without this information, the *Bombe* would still have to potentially try every possible initial configuration before a message could be decrypted [10].

Bearing this is mind, the Enigma machine had the potential to be a valid form of encryption long after World War II, due to the length of time it would take to break it through a brute-force attack. Unless one's message was specifically intercepted by someone with access to a *Bombe* and whom had the required information, breaking an Enigma-encrypted message would take prohibitively long.

In a brute-force attack, one tries every single possible combination in order to find the correct decryption. To use a simple example, in a brute-force attack on a 3-digit combination lock, one would try every combination starting with 1-1-1, going to 1-1-2, 1-1-3, and so on, until the correct combination was found. In the worst case scenario, every single one of the 999 combinations would have to be tried.

In the worst-case scenario for a brute-force attack on an Enigma machine, one would have to try every initial configuration possible. However, one would not have to try to decrypt the entire message to determine if the correct initial configuration had been assumed. To this point, to ensure that the correct settings were being used each time, German users encrypting with Enigma in the field were instructed to begin with the a user-chosen three-letter sequence, say XYZ, being encrypted twice. Then on the other end, if the decrypted message began with a repeated three-letter sequence, XYZXYZ in this case, the recipient of the message would know that the correct initial configuration had very likely been used. If not, the recipient would know that an incorrect initial configuration had been used.

This meant that to check if an assumed initial configuration was correct, one could begin by comparing only the first and fourth letters in the attempted decryption. If these letters did not match, it would be known that the assumed initial configuration was incorrect. If they did match, then the second and fifth letters could be compared. If these letters did not match, it would again be known that the assumed initial configuration was incorrect. If they did match, then the third and sixth letters could be compared. Only if all three pairs matched was it necessary to look any further in an attempted decryption. On average, $\frac{25}{26}$ of the time one should be able to stop after the fourth letter; upon having to proceed, $\frac{25}{26}$ of the time one should be able to stop after the fifth letter; and again if having to proceed, $\frac{25}{26}$ of the time one should be able to stop after the sixth letter. This meant that, in total, one should only have to decrypt more than six letters $\frac{1}{26^3} = \frac{1}{17{,}576}$ of the time. After those initial six letters, decrypting ten additional letters and checking to see if that portion of the decrypt was legible should suffice to indicate if the message should be decrypted in full. While false positives, or incorrect configurations that might by chance decrypt the first ten letters of the message into a readable phrase, were possible, they were unlikely enough that it would be worth fully decrypting any message that was readable through the first ten letters.

To check a single initial configuration by hand would have taken roughly one and a half minutes, or 90 seconds, on average. Most of the time required for one to check an initial configuration by hand would be spent on setup; the time spent actually decrypting letters after that is insignificant. One of the first computers, ENIAC, was able to carry out 5000 computations per second when it was completed in 1946 [11]. The CDC 6600, the first supercomputer, could carry out up to 3 million operations per second [9]. This is compared to the processing power available to the typical university student in 2017, who uses a calculator that can perform 1.16 million operations per second [13, 15], and a computer that, on average, can perform 83 billion operations per second [1, 13]. The average modern-day supercomputers can perform 17 quadrillion operations per second [12]; the world's current fastest supercomputer, the Sunway TaihuLight, can perform 93 quadrillion operations per second [12].

For the commercial Enigma, decrypting a single letter involved 8 calculations: stepping the rotor, going through each of the three rotors, going through the reflector, and back through each of the three rotors. The *Wehrmacht* and *Kriegsmarine* Enigmas required 10 calculations; the number of rotors and reflectors available to choose from did not affect the number of calculations required, but going through the plugboard at the beginning and end added two calculations. The M4 Enigma, because it introduced an extra rotor, required two extra calculations to decrypt each letter, bringing its total up to 12 calculations per letter. The theoretical Enigma, since it assumes a 3-rotor Enigma would take only 10 calculations per letter.

Finding the maximum amount of time it would take for a particular system to cryptanalyze a message encrypted with a given version of the Enigma can be done as follows. First, the time required when decrypting by hand is simple; since the time given is per initial configuration, one only needs to multiply the number of initial configurations by the length of time taken to test each one. For computer decryption speeds, one first divides the number of initial configurations that a version produces by 17,576 to find the number of initial configurations that would have to be checked beyond the sixth letter. That number is then multiplied by 16 and again by the number of calculations needed per letter for that version of the Enigma to find the total number of calculations that would need to be performed to check every initial configuration. When working with the human decryption speed, this step is skipped, since that speed is per initial configuration rather than per calculation. One then divides by the number of calculations the system can perform per second to find the number of seconds it would take to run all of the necessary calculations. Once this number is found, it can be converted into other units of time as desired. Examples of these calculations for human and one type of computer can be seen below. The results for the given systems and versions of the Enigma machine are summarized in the tables that follow (Figures 14, 15).

To determine the maximum time it would take for a human to find the correct initial configuration for a commercial Enigma, one multiplies the number of initial configurations possible with a commercial Enigma by the time it takes for a human to test a single configuration.

- Initial configurations: 48,190,861,056
- Time taken per initial configuration: 90 seconds
- Total time: $48{,}190{,}861{,}056 \cdot 90 = 4{,}337{,}177{,}495{,}040$, or 4.3×10^{12}, seconds

This time can be converted into years by dividing by 3600 (the number of seconds in an hour), then by 24 (the number of hours in a day), followed by 365 (the number of days in a year).

To determine the maximum time it would take the Sunway TaihuLight to find the correct initial configuration for the theoretical Enigma, one divides the number of initial configurations possible by 17,576 to account for how often the repeated sequence was correct, and multiplies by the number of letters that need to be checked (16) and by the number of calculations required per letter for that version of the the Enigma (10). This number is then divided by the Sunway TaihuLight's speed.

- Initial configurations: 2.2×10^{117}
- Repeated sequence: 17,576
- Number of letters: 16
- Calculations per letter: 10
- Computer speed: 93 quadrillion, or 9.3×10^{16}, calculations per second
- Total time: $\big((2.2 \times 10^{117} \div 17{,}576) \cdot 16 \cdot 10\big) \div 9.3 \times 10^{16} = 2.2 \times 10^{98}$ seconds

Like with the calculation above, this time can be converted to years by dividing appropriately, giving 7.0×10^{90} years.

	Commercial	Original *Wehrmacht*	*Kriegsmarine*	M4	Theoretical
Human	4.3×10^{12}	5.0×10^{25}	5.0×10^{26}	3.8×10^{30}	2.0×10^{119}
ENIAC	6.8×10^{4}	1.0×10^{18}	1.0×10^{19}	9.1×10^{22}	4.0×10^{111}
CDC 6600	119	1.7×10^{15}	1.7×10^{16}	1.5×10^{20}	6.8×10^{108}
University Calculator	302	4.4×10^{15}	4.4×10^{16}	2.9×10^{20}	1.8×10^{109}
University Computer	4.2×10^{-3}	6.3×10^{10}	6.3×10^{11}	5.6×10^{15}	2.4×10^{104}
Average Supercomputer	2.0×10^{-8}	3.0×10^{5}	3.0×10^{6}	2.7×10^{10}	1.2×10^{99}
Sunway TaihuLight	3.8×10^{-9}	5.5×10^{4}	5.5×10^{5}	4.9×10^{9}	2.2×10^{98}

Figure 14: *Time, in seconds, to cryptanalyze the Enigma by each system; figure created by author*

Overall, if someone utilized a theoretical Enigma and kept the initial configurations secure, it would clearly be impossible to carry out a brute-force attack on a useful timescale. Having to

	Commercial	Original *Wehrmacht*	*Kriegsmarine*	M4	Theoretical
Human	1.4×10^5 years	1.6×10^{18} years	1.6×10^{19} years	1.2×10^{23} years	6.3×10^{111} years
ENIAC	18.9 hours	3.2×10^{10} years	3.2×10^{11} years	2.9×10^{15} years	1.3×10^{104} years
CDC 6600	119 seconds	5.4×10^7 years	5.4×10^8 years	4.8×10^{12} years	2.2×10^{101} years
University Calculator	302 seconds	1.4×10^8 years	1.4×10^9 years	9.2×10^{12} years	5.7×10^{101} years
University Computer	4.2 milliseconds	2.0×10^3 years	2.0×10^4 years	1.8×10^8 years	7.6×10^{96} years
Average Supercomputer	20 nanoseconds	3.5 days	34.7 days	856 years	3.8×10^{91} years
Sunway TaihuLight	38 nanoseconds	15.3 hours	6.4 days	155 years	7.0×10^{90} years

Figure 15: *Time, in alternate units, to cryptanalyze the Enigma by each system; figure created by author*

try every initial configuration would necessitate an amount of time to break that is on par with those required for modern encryption methods.

5 Simulators

The length of time it would take to carry out a brute-force attack on a full theoretical Enigma machine puts it on par with modern encryption methods. Despite this, the separate steps behind the cipher are mathematically very simple. This allows for there to be a multitude of ways to simulate the workings of an Enigma machine. The simplest of these ways is to build a three-dimensional, completely mechanical model of the diagram shown in Figure 10. Franklin Heath Ltd., in the United Kingdom, created a model of the Enigma using rings cut out of paper (Figures 16, 17). When rotated around a spindle, in this case a Pringles® can, the encoding of each letter can be tracked all the way through the machine.

This simulator's greatest benefit comes from the lack of technical or mathematical skill required to create and use it; assembling a paper Enigma simulator takes less than an hour and requires nothing more than scissors, tape, and a cardboard tube. While this method gives an extremely clear picture of how an Enigma machine works and allows one to see exactly what is happening as the rotors progress, it is an extremely slow method and it is very prone to error.

There are numerous electronic simulators built with languages such as C++, Java, and Python. These simulators are easy to operate; in even the most complicated of instances, they usually only involve selecting the plugboard and rotor settings. However, without a deep

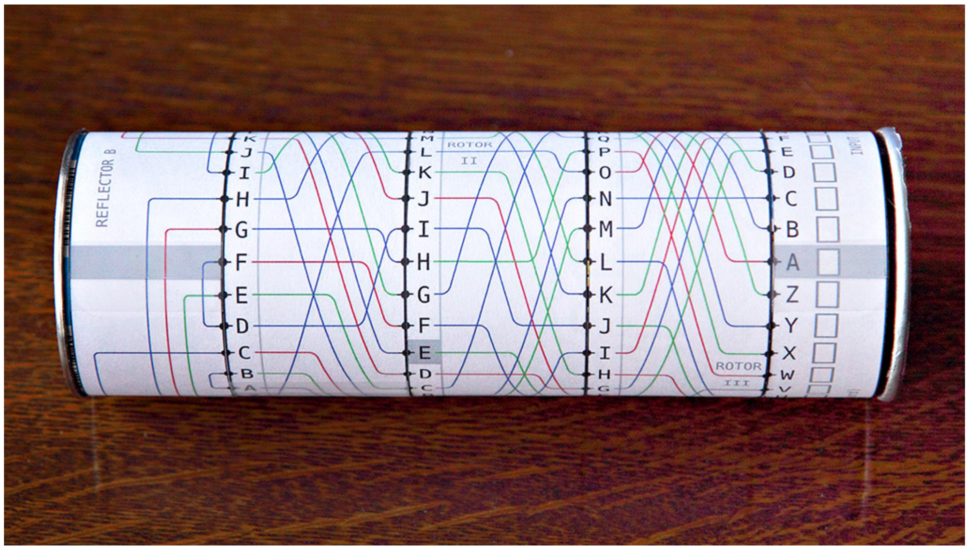


Figure 16: *Fully-assembled paper Enigma simulator; image reproduced with permission of Franklin Heath Ltd.* [3]

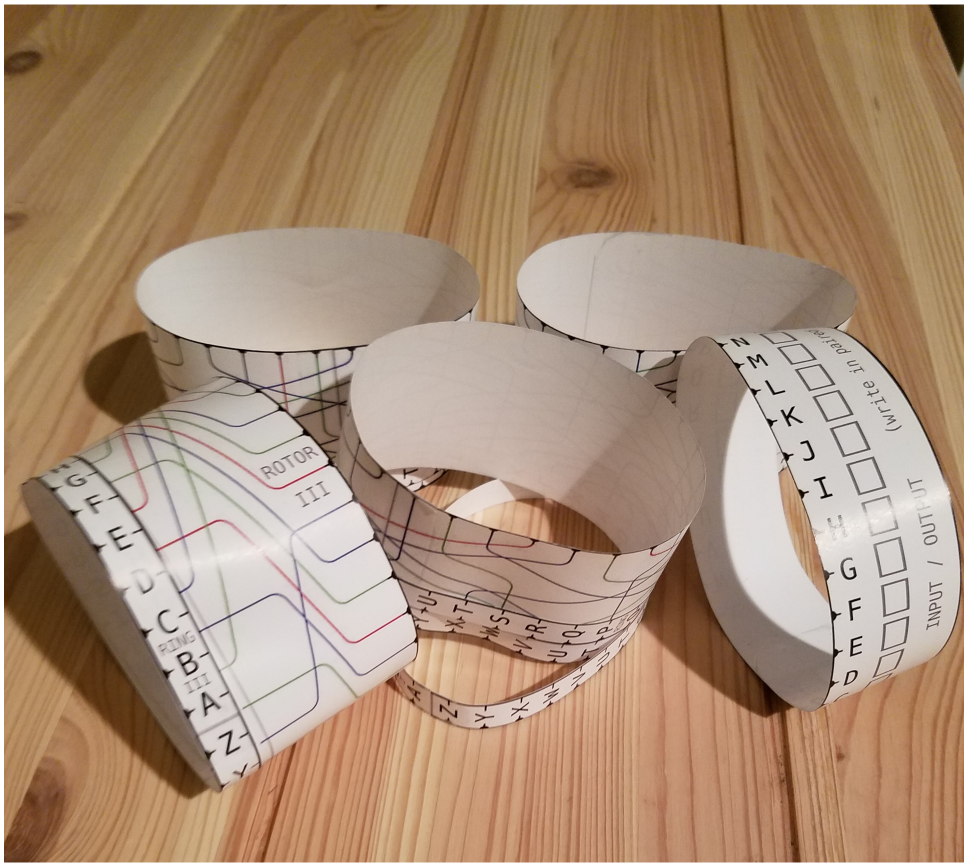


Figure 17: *Rings used in the paper Enigma simulator; image taken by the author*

understanding of the language being used, one has to take it on faith that the simulator was written correctly and, correspondingly, that the message was encoded properly. The code also leaves one without an understanding of what the Enigma machine actually was. Running the code allows one to encrypt and decrypt a message, but ultimately leaves one without an understanding of what the machine even looked like, let alone the workings involved.

For this project, an Enigma simulator was written in Python, a portion of the code for which can be seen in Figure 18. The complete code for this Python simulator can be seen in Appendix A. The code can simulate a commercial or *Wehrmacht* Enigma in its entirety. In the case of Python and C++, the most common languages used for Enigma simulators after Java, there is the added problem that the user must be able to make minor modifications to the program, as is usually necessary to change the rotors, reflector, and plugboard.

The code written allows the user to choose from the rotors and reflectors available for the commercial and *Wehrmacht* Enigmas and to specify the plugboard connections (Figures 19). Those, along with the ring and settings, gives the user complete control over the initial configuration of the machine. Once the program is run, it will return the encrypted message (Figure 20). If an encrypted message was provided, the decrypted message will be returned.

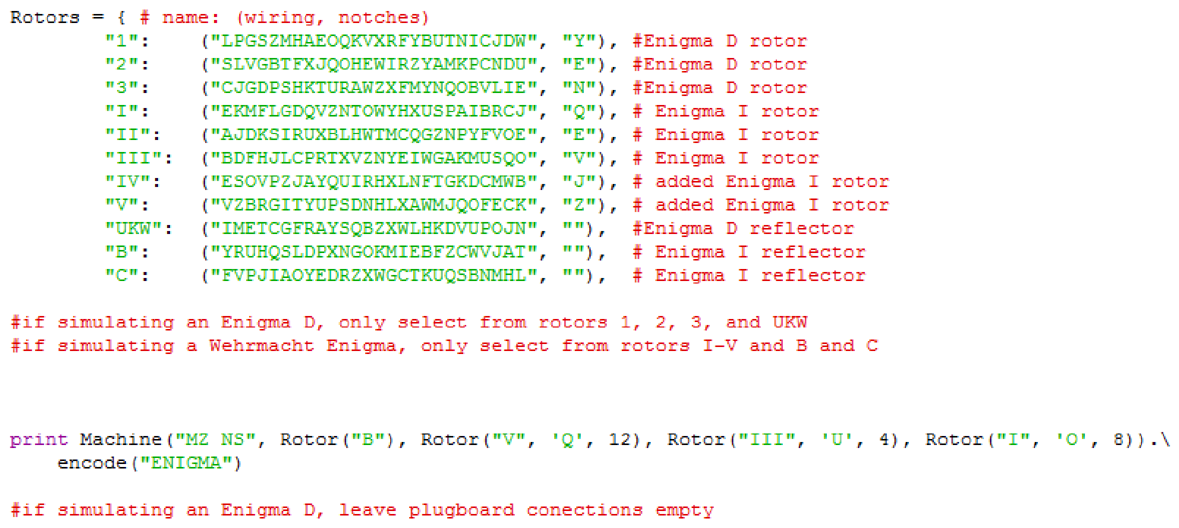
The best simulators combine the three-dimensional and code to produce a physical replica of an Enigma machine as it would have been used. S&T GeoTronics [8] provides a kit, the Enigma Mark 4, from which a working model of an Enigma machine can be built. Several other companies produce similar kits. The Enigma Mark 4 is run by an Arduino board and a USB power source, but otherwise it accurately simulates both a *Wehrmacht* and *Kriegsmarine* Enigma. For this project, an Enigma Mark 4 kit was assembled (Figures 21, 22). Its major flaw, like the coded simulators, is that it does not show the workings of the rotors and reflectors.

There is also a considerable amount of technical skill required to assemble an Enigma model; the Mark 4 requires a knowledge of soldering, programming, and woodworking to be fully assembled. Although pre-assembled kits are available, the cost can be prohibitive.

Additional photographs of the Mark 4 Enigma during assembly can be seen in Appendix B.

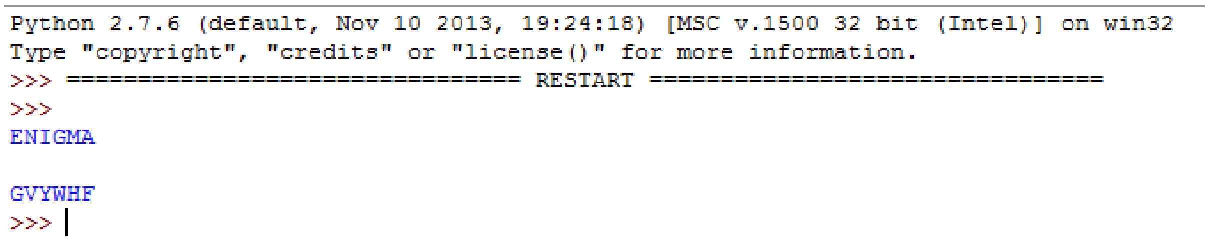
```
class Machine:
    def __init__(self, plugboard, *rotors):
        self.plugboard=range(26)
        for pair in plugboard.split():
        #swaps letters if connected in plugboard
            a, b=map(Ord,pair)
            self.plugboard[a]=b
            self.plugboard[b]=a
        self.rotors=rotors
    def encode(self, message, expected=None):
        out=""
        #begins encoded message as empty string
        for ch in message:
            left, middle, right=self.rotors[-3:]
            #advances the rotors
            if middle.knocks():
                left.advance()
                middle.advance()
            elif right.knocks():
                middle.advance()
            right.advance()
            ofs=0
            ch=self.plugboard[Ord(ch)]
            #through the plugboard
            for rotor in self.rotors[::-1]:
            #through the rotors right to left and then reflector
                ch, ofs=rotor.enter(ch, ofs), rotor.ofs
            for rotor in self.rotors[1:]:
            #back through the rotors left to right
                ch, ofs=rotor.exit(ch, ofs), rotor.ofs
            ch=(ch-rotor.ofs)%26
            ch=self.plugboard[ch]
            #back through the plugboard
            out+=Chr(ch)
            #concatenates message
        return out
```

Figure 18: *Portion of the code used in the Python Enigma simulator; code based off of that produced on the blog William Edwards, Coder* [16]



```
Rotors = { # name: (wiring, notches)
        "1":    ("LPGSZMHAEOQKVXRFYBUTNICJDW", "Y"), #Enigma D rotor
        "2":    ("SLVGBTFXJQOHEWIRZYAMKPCNDU", "E"), #Enigma D rotor
        "3":    ("CJGDPSHKTURAWZXFMYNQOBVLIE", "N"), #Enigma D rotor
        "I":    ("EKMFLGDQVZNTOWYHXUSPAIBRCJ", "Q"), # Enigma I rotor
        "II":   ("AJDKSIRUXBLHWTMCQGZNPYFVOE", "E"), # Enigma I rotor
        "III":  ("BDFHJLCPRTXVZNYEIWGAKMUSQO", "V"), # Enigma I rotor
        "IV":   ("ESOVPZJAYQUIRHXLNFTGKDCMWB", "J"), # added Enigma I rotor
        "V":    ("VZBRGITYUPSDNHLXAWMJQOFECK", "Z"), # added Enigma I rotor
        "UKW":  ("IMETCGFRAYSQBZXWLHKDVUPOJN", ""),  #Enigma D reflector
        "B":    ("YRUHQSLDPXNGOKMIEBFZCWVJAT", ""),  # Enigma I reflector
        "C":    ("FVPJIAOYEDRZXWGCTKUQSBNMHL", ""),  # Enigma I reflector

#if simulating an Enigma D, only select from rotors 1, 2, 3, and UKW
#if simulating a Wehrmacht Enigma, only select from rotors I-V and B and C



print Machine("MZ NS", Rotor("B"), Rotor("V", 'Q', 12), Rotor("III", 'U', 4), Rotor("I", 'O', 8)).\
    encode("ENIGMA")

#if simulating an Enigma D, leave plugboard conections empty
```

Figure 19: *Python Enigma inputs; image taken by the author*



```
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
ENIGMA

GVYWHF
>>> 
```

Figure 20: *Python Enigma outputs; image taken by the author*

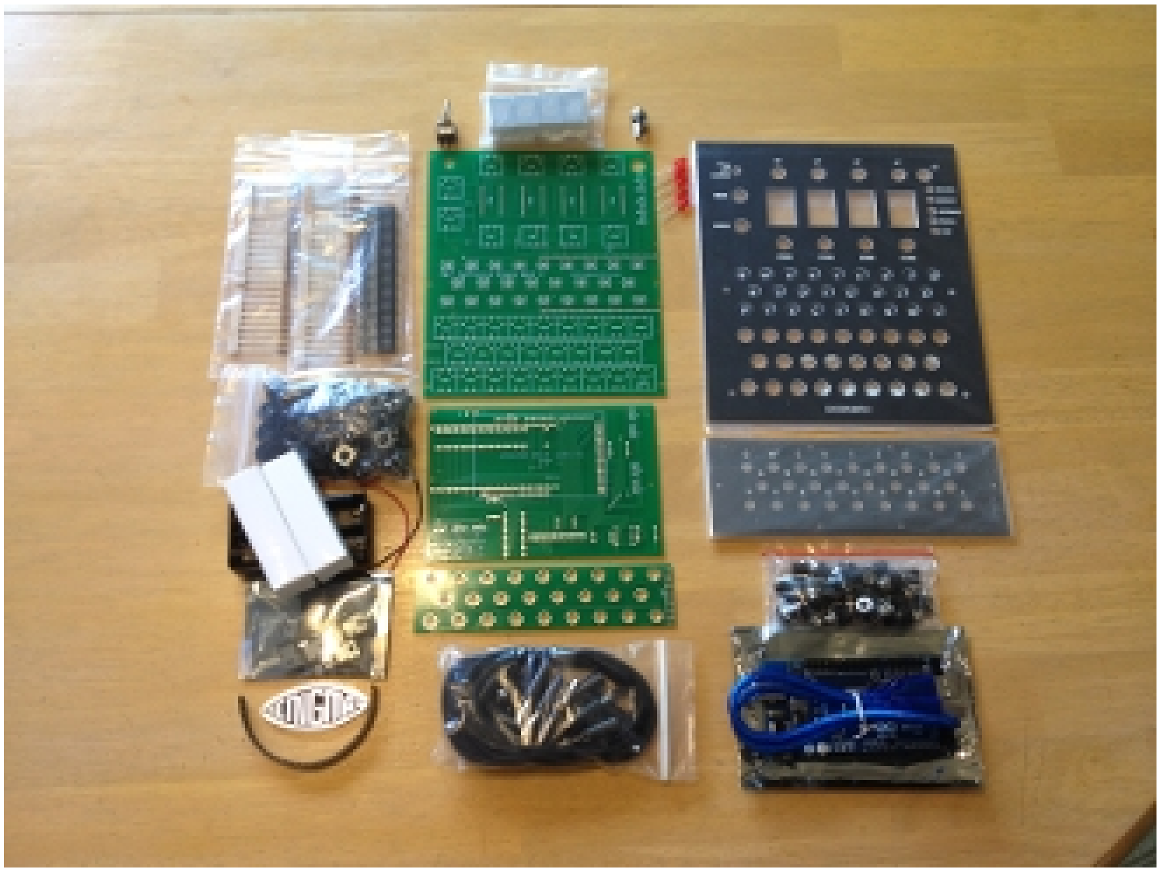


Figure 21: *Mark 4 Enigma kit, pre-assembly; image taken by the author*

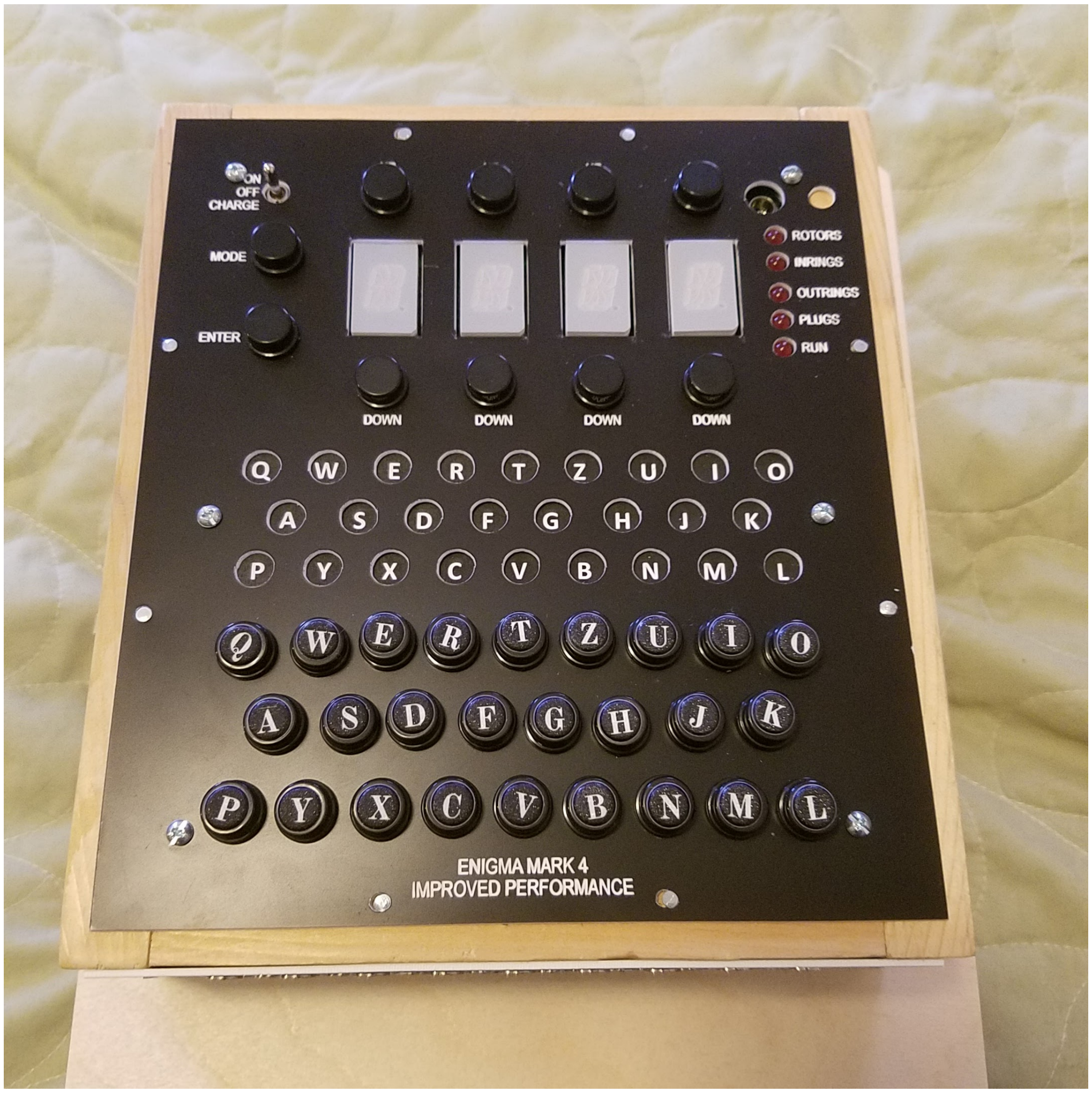


Figure 22: *Fully-assembled Enigma Mark 4 kit; image taken by the author*

Acknowledgments

Thank you very much to my advisor, Dr. Richard Klima, for all of his help and guidance throughout my project. I would also like to thank Dr. Donna L. Lillian for her insights and advice.

Thank you as well to the many professors in the Department of Mathematical Sciences who helped me with various aspects of this project.

I also must thank S&T Geotronics, whose technical assistance and support in assembling the Mark 4 was much appreciated, and William Edwards, whose code provided the framework for the Python simulator presented.

Thank you to the Office of Student Research, the Honors College, and the Department of Mathematical Sciences for their generous support, without which this project would not have been possible.

Finally, thank you again to the Honors College. The Chancellor's Scholarship provided by the Honors College made all of this possible.

Bibliography

[1] Appalachian State University, *Recommended Computer Specifications*, https://support.appstate.edu/services/technology-support-center/recommended-computer-specifications.

[2] Bletchley Park Trust, *Bletchley Park*, https://www.bletchleypark.org.uk/.

[3] Franklin Heath Ltd., *Enigma/Paper Enigma*, http://wiki.franklinheath.co.uk/index.php/Enigma/Paper_Enigma.

[4] Richard Klima and Neil Sigmon, *Cryptology: Classical and Modern with Maplets*, CRC Press, 2012.

[5] Paul Lunde, *The Secrets of Codes: Understanding the World of Hidden Messages*, Weldon Owen, 2012.

[6] The National WWII Museum, *Enigma Archives*, http://www.nww2m.com/tag/enigma/.

[7] Paul Reuvers and Marc Simons, *Enigma Cipher Machines*, http://www.cryptomuseum.com/crypto/enigma/index.htm.

[8] S&TGeotronics, *Enigma Replica*, http://www.stgeotronics.com/Enigma-Replica_c3.htm.

[9] Wikimedia Foundation Inc., *CDC 6600*, https://en.wikipedia.org/wiki/CDC_6600.

[10] ______, *Cryptanalysis of the Enigma*, https://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma.

[11] ______, *ENIAC*, https://en.wikipedia.org/wiki/ENIAC.

[12] ______, *History of Supercomputing*, https://en.wikipedia.org/wiki/History_of_supercomputing.

[13] ______, *Instructions per second*, https://en.wikipedia.org/wiki/Instructions_per_second.

[14] ______, *Marian Rejewski*, https://en.wikipedia.org/wiki/Marian_Rejewski.

[15] ______, *TI-83 series*, https://en.wikipedia.org/wiki/TI-83_series.

[16] William Edwards, *William Edwards, Coder*, https://williamedwardscoder.tumblr.com/.

Appendix A

```
def Ord(ch): return ord(ch)-ord('A')            #convert A-Z to 0-25

def Chr(ch): return chr(ch+ord('A'))            #convert 0-25 to A-Z

def Text(s): return "".join(ch for ch in s if ch in "ABCDEFGHIJKLMNOPQRSTUVWXYZ")


Rotors = { # name: (wiring, notches)

        "1":    ("LPGSZMHAEOQKVXRFYBUTNICJDW", "Y"), #Enigma D rotor

        "2":    ("SLVGBTFXJQOHEWIRZYAMKPCNDU", "E"), #Enigma D rotor

        "3":    ("CJGDPSHKTURAWZXFMYNQOBVLIE", "N"), #Enigma D rotor

        "I":    ("EKMFLGDQVZNTOWYHXUSPAIBRCJ", "Q"), # Enigma I rotor

        "II":   ("AJDKSIRUXBLHWTMCQGZNPYFVOE", "E"), # Enigma I rotor

        "III":  ("BDFHJLCPRTXVZNYEIWGAKMUSQO", "V"), # Enigma I rotor

        "IV":   ("ESOVPZJAYQUIRHXLNFTGKDCMWB", "J"), # added Enigma I rotor

        "V":    ("VZBRGITYUPSDNHLXAWMJQOFECK", "Z"), # added Enigma I rotor

        "UKW":  ("IMETCGFRAYSQBZXWLHKDVUPOJN", ""),  #Enigma D reflector

        "B":    ("YRUHQSLDPXNGOKMIEBFZCWVJAT", ""),  # Enigma I reflector

        "C":    ("FVPJIAOYEDRZXWGCTKUQSBNMHL", ""),  # Enigma I reflector


#if simulating an Enigma D, only select from rotors 1, 2, 3, and UKW

#if simulating a Wehrmacht Enigma, only select from rotors I-V and B and C


class Rotor:

    def __init__(self, table, ofs='A', ring=1):         #intializes rotor

        self.rotor, (table, notches) = table, Rotors[table]

        self.table=map(Ord, table)

        self.reciprocal=[self.table.index(i) for i in range(26)]

        self.notches=[(n-ring+1)%26 for n in map(Ord, notches)]

        self.ofs=Ord(ofs)-ring+1

        self.ring=ring-1

    def knocks(self): return (self.ofs%26) in self.notches

    def advance(self): self.ofs+=1

    def enter(self, ch, ofs): return self.table[(ch+self.ofs-ofs)%26]           #letter enters rotor

    def exit(self, ch, ofs): return self.reciprocal[(ch+self.ofs-ofs)%26]           #letter exits rotor


class Machine:

    def __init__(self, plugboard, *rotors):

        self.plugboard=range(26)

        for pair in plugboard.split():          #swaps letters if connected in plugboard

            a, b=map(Ord,pair)

            self.plugboard[a]=b
            self.plugboard[b]=a
        self.rotors=rotors
    def encode(self, message, expected=None):
        out=""          #begins encoded message as empty string
        for ch in message:
            left, middle, right=self.rotors[-3:]            #advances the rotors
            if middle.knocks():
                left.advance()
                middle.advance()
            elif right.knocks():
                middle.advance()
            right.advance()
            ofs=0
            ch=self.plugboard[Ord(ch)]          #through the plugboard
            for rotor in self.rotors[::-1]:             #through the rotors right to left and then reflector
                ch, ofs=rotor.enter(ch, ofs), rotor.ofs
            for rotor in self.rotors[1:]:           #back through the rotors left to right
                ch, ofs=rotor.exit(ch, ofs), rotor.ofs
            ch=(ch-rotor.ofs)%26
            ch=self.plugboard[ch]           #back through the plugboard
            out+=Chr(ch)            #concatenates message
        return out

print Machine("MZ NS", Rotor("B"), Rotor("V", ’Q’, 12), Rotor("III", ’U’, 4), Rotor("I", ’O’, 8)).\
    encode("ENIGMA")

#if simulating an Enigma D, leave plugboard connections empty
```

Appendix B

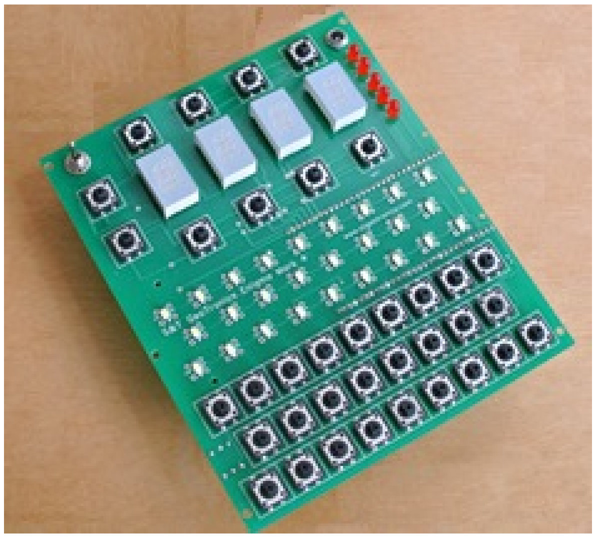


Figure 23: *Mark 4 Enigma kit during circuit-board assembly; image taken by the author*

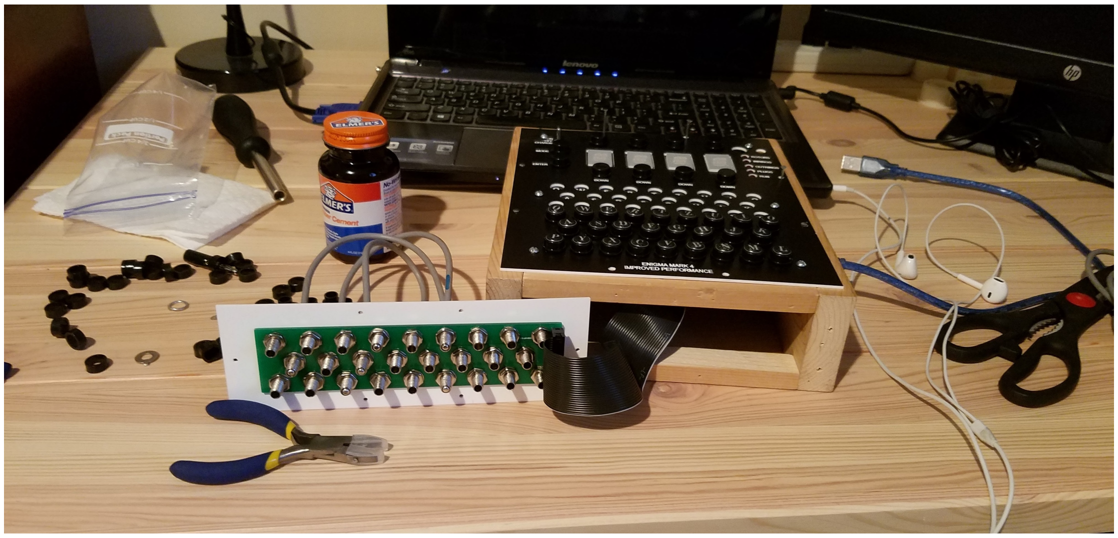


Figure 24: *Mark 4 Enigma kit during case assembly; image taken by the author*