Pyokyeong Son
IB Mathematics HL
Mr. Andre Ito
January 9th, 2018

The Mathematics of Deciphering the Enigma machine

I. Introduction

In this math exploration, I will examine the mathematics in the design of the Enigma Machine, an encryption device used by Nazi Germany for military communications, as well as its deciphering by the British military intelligence team in Bletchley Park. I will detail the inner workings of both the Enigma Machine and the British deciphering device Bombe, and use the knowledge to construct a computer program that would replicate the mechanism of the Enigma, and a simplified Bombe machine.

II. Rationale

I took interest in the Enigma machine after watching the movie "Imitation Game," which told the story of the members of Bletchley Park breaking the Enigma. I was always interested in the specifics of encryption algorithms, and I believed that understanding how the first modern encryption device worked and how it was broken, would be a great first step into understanding the fundamentals of encryption.

Encryption is a crucial part in our lives in this age of information. For all practical purposes, however, encryption may seem to be unneeded; Assuming everybody is well-purposed and are "nice" people, there is no reason to hide information.

However, by nature, humans desire privacy, and that desire is one of the foundations of the design of our modern society. Whether it is to keep military secrets, or to hide your embarrassing texts with your ex-boyfriend, encryption, I believe, represents a technology solely developed for humans. Our craving for privacy and secrecy is what makes us human, and encryption is what enables us to do so in this digital world.

Enigma is one of the milestones in the field of encryption. It was developed during when electronic communication was starting to spread, and a period where keeping secrets was vital; World War 2. It signals the start of digital encryption, and was designed with enough mathematical confidence and complexity that the cipher text messages were broadcasted in open channels. It relies on mathematical models, not loyalty or pressure, to keep secrets.

Another significant fact about the machine is that it was defeated. The fact that the first encryption device that was thought to be unbreakable, was broken, reveals how every encryption method is vulnerable to exploits, and reminds us that no algorithm is unbreakable. Modern encryption methods, however complicated, is theoretically breakable; the currently widely-used RSA algorithms can, theoretically, be defeated using a quantum computer, and alternatives are already being designed. This shows that at a fundamental level, encryption is always incomplete—there is no "perfect" algorithm, only a better one.

By understanding the mathematics behind the Enigma and the Bombe, I believe I would gain a better understanding of the fundamental ideas behind encryption; how, but more importantly, why, we design encryption.

III. Details of the Enigma's design

The Enigma machine looks like a normal typewriter, only that in the place of the paper, there is a set of lamps on the top of the machine. The user would press a letter on the keyboard, and a different letter would light up; the letter has been encrypted.

Fig. 3.1. "Enigma (crittografia) - Museo scienza e tecnologia Milano." *Wikimedia Commons.* Accessed 20 Oct. 2016.

Internally, The Enigma machine is a set of many complicated circuits, in which pressing one key on the keyboard completes a circuit, as the signal passes through various ciphers, and lights up a corresponding letter in the lamp array. Figure 3.2 reveals all of the cryptographic circuits of the Enigma.

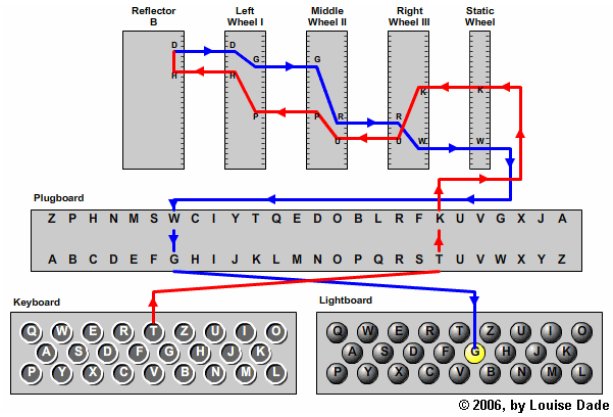


Fig. 3.2. Dade, Louise. The inner workings of an Enigma machine. 2006. Accessed 20 Oct. 2016.

1. Rotors (Wheel)

The rotors are the main security feature of the enigma. They are circular disks placed at the center of the machine, each performing a simple substitution cipher for every alphabetical character; for instance, a rotor might receive a signal in the wire corresponding to "K," and output the signal to the wire for "U."

The rotor can be rotated(thus, the name), which changes the connected input and output. The Enigma typically used by the German military had 3 rotors. Therefore, in this enigma, the rotor by themselves have the following number of combinations:

$$\text{Rotor Combinations} = 26 \cdot 26 \cdot 26 = 17{,}576$$

The rotors also have a feature named “Stepping” and “Turnover.” Stepping describes the actions in which the rightmost rotor rotates one position every time an encryption or decryption is performed. Each rotor also has a notch at its “turnover position.” When a rotor continues stepping and reaches this notch, the rotor left to it would “step” one position, essentially causing a Turnover.

It is simple to imagine the rotors as incrementing a base-26 3-digit number; the rightmost digit would increment from A to Z, and then the center digit would increment by one, resetting the ones digit back to A, and so on.

$$\text{AAA} \rightarrow \text{AAB} \rightarrow \text{AAC} \rightarrow \ldots \rightarrow \text{AAZ} \rightarrow \text{ABA} \rightarrow \text{ABB} \rightarrow \ldots$$

The implementation of stepping and turnover was one of the reasons why some considered the Enigma unbreakable, as the substitution of the rotors changes with every encryption. For example, if we were to encrypt the letters E, E, and E, sequentially, they will be encrypted differently, for example, to A, C and K.

2. Reflector

A reflector receives a letter signal from the rotors, and “reflects” the signal as another letter back into the rotors. The signal travels through all of the rotors again.

Figure 3.3 visualizes the process of the encryption provided by the 3 rotors, stepping and turnover, and the reflector:

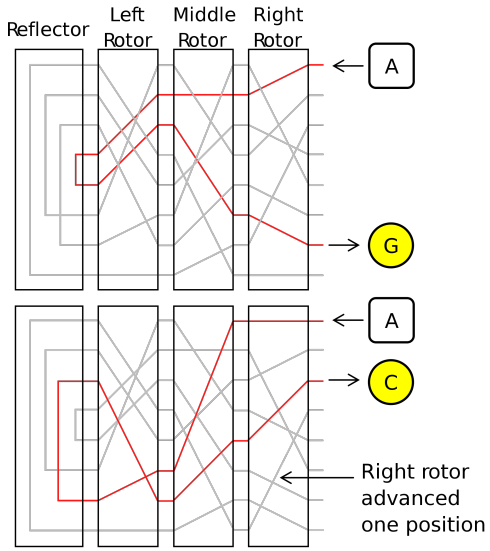


Fig. 3.3. “Enigma-action” Wikimedia Commons. Accessed 20 Oct. 2016

3. Plugboard

The rotors itself would not be to much of a security feature by itself; 17,576 seems like a huge number, but the importance of deciphering this machine was enough to motivate the Polish and British intelligence to hire thousands of workers to try out each combination by hand, which would not take a long time.

To prevent this, the machines used in the military incorporated a component called the plugboard. At the front panel of the machine, the user could connect a wire between two holes representing letters,

swapping them before and after the rotor encryptions. The number of wire settings on a plugboard can be calculated using fairly simple combinatorics:

The number of wires used = $p \quad (1 \leqslant p \leqslant 13)$

(Maximum 13 pairs can be made in a 26 letter-alphabet.)

We can think of choosing 2 letters, regardless of order, from 26 letters; then 2 letters from 24; and so on:

$$\binom{26}{2} \cdot \binom{24}{2} \cdot \binom{22}{2} \cdot \ldots \cdot \binom{26-2(p-1)}{2}$$

However, the order of the plugs inserted does not matter, so we an divide by the number of orders that can be made, which gives the total number of plugboard combinations:

$$\text{Plugboard Combinations} = \frac{\binom{26}{2} \cdot \binom{24}{2} \cdot \binom{22}{2} \cdot \ldots \cdot \binom{26-2(p-1)}{2}}{p!}$$

This can be simplified to:

$$\frac{\frac{26 \cdot 25}{2} \cdot \frac{24 \cdot 23}{2} \cdot \frac{22 \cdot 21}{2} \cdot \ldots}{p!} = \frac{\frac{26!}{(26-2p)! \cdot 2^p}}{p!} = \frac{26!}{(26-2p)! \cdot p! \cdot 2^p}$$

In practice, the German navy swapped 10 pairs of letters. Therefore, the number of combinations will be:

$$\frac{26!}{(26-2 \cdot 10)! \cdot 10! \cdot 2^{10}} = 150{,}738{,}274{,}937{,}250 \text{ Combinations}$$

In total, therefore, a 3-rotor, 10-wire plugboard Enigma would have the following number of combinations:

$$26 \cdot 26 \cdot 26 \cdot \frac{26!}{(26-2 \cdot 10)! \cdot 10! \cdot 2^{10}} = 2{,}649{,}375{,}920{,}297{,}106{,}000$$

By using counting principles like combinations we were able to find out that the Enigma machine had a huge humber of possible keys. This number alone was intimidating for me, and it must have been for the codebreakers in Bletchley Park. The fact that by using simple mechanisms like a rotor, reflector, and a plugboard, a huge number of combinations could be derived, is a fascinating feature of this machine, as well as the mathematical calculation behind it. It seemed to me that breaking such a number would be impossible. However, as it turns out, it is not the number that we care about, but the mechanisms themselves, as detailed in the following sections.

IV. Breaking the Enigma

The mechanics of the Enigma were well known; the codebreakers had an exact replica of the Enigma machine and its rotors. However, they did not know the settings for the rotors or the plugboard.

< Known information >

- Enigma's encryption mechanism, that uses stepping Rotors, a Reflector, and a Plugboard for encryption
- Wiring of the rotors, as shown in table 4.1.

	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Rotor 1	EKMFLGDQVZNTOWYHXUSPAIBRCJ
Rotor 2	AJDKSIRUXBLHWTMCQGZNPYFVOE
Rotor 3	BDFHJLCPRTXVZNYEIWGAKMUSQO

Table. 4.1. Wiring of some of the Enigma Rotors.

The following settings are the "keys" for encrypting or decrypting a message. If we know the key, we can decrypt any cyphertext generated with that key:

< Key >

- Initial positions of the 3 rotors
- Plugboard Combinations

In practice, the Germans had 5 rotors from which they could choose 3, and 3 reflectors from which they would choose 1. But there were inconsistencies in these implementations. Therefore, for simplicity, we will ignore these combinations.

The Enigma's keys change every day, so we have less than a 24 hours to find the key, and use that key to decrypt that day's messages.

1. Using the Crib, exploiting the Reflector

The British codebreakers were able to infer some of the plaintext in the German messages. For example, the British knew the time and the format of the German military's weather report. They might have intercepted the following part of an encrypted message during when the German military would send their daily weather report:

```
Encrypted Message: … A X J K O T V H E U L V T P T …
```

We know that some part of this string decrypts to "W E A T H E R R E P O R T," and we can use this information to help figure out the key. This is known as a known-plaintext attack, in which the attacker uses a known decrypted message and an encrypted message to figure out the key. The plaintext string, "W E A T H E R R E P O R T," is called a **crib**. There are currently 3 possible positions of the crib in the cyphertext.

```
Encrypted Message:    A X J K O T V H E U L V T P T
Position 1:           W E A T H E R R E P O R T
Position 2:             W E A T H E R R E P O R T
Position 3:               W E A T H E R R E P O R T
```

We have understood the reflector's mechanism, and from that, we can deduce that the reflector never sends back the same letter. This can be visualized easily by taking a second look at the diagram of the rotor boxes and the reflector in figure 4.2. Because each press of the letter completes a single circuit, the reflector

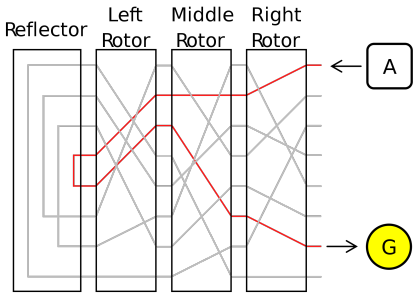


Fig. 4.2. “Enigma-action” Wikimedia Commons. Accessed 20 Oct. 2016

cannot output the same letter as its input; the wire must come in at one point, and come out from another to complete a circuit. (Equally, therefore, in the same rotor position, if A encrypts to G, G would encrypt to A.) In the above diagram, we can see that all input wires inside the reflector forms a pair, and therefore it is not possible to reflect the same letter as it received. Therefore, in a more general sense, we can say that **a letter never encrypts to itself.** This is a limitation of the Enigma being a set of circuits—a circuit has to complete a loop for the signal to flow.

Taking advantage of this flaw, we can eliminate some of the positions where a letter encrypts to itself:

```
Position              0 1 2 3 4 5 6 7 8 9 1011121314
Encrypted Message:    A X J K O T V H E U L V T P T
Position 1:           W E A T H E R R E P O R T
Position 2:             W E A T H E R R E P O R T
Position 3:               W E A T H E R R E P O R T
```

For position 1, the 8th letter “E” and 12the letter “T” are same in the plaintext and cyphertext, so position 1 cannot be the correct position. In the 3rd position, the 5th letter “T” and the 14th letter “T” are same in the plaintext and cyphertext, so it cannot be the correct position. Therefore, we can eliminate all the positional possibilities except position 2:

```
Position              1 2 3 4 5 6 7 8 9 10111213
Encrypted Message:    X J K O T V H E U L V T P
Position 2:           W E A T H E R R E P O R T
```

2. Brute force attack

We will now start to come up with some tactics to find out the key for each of these possibilities. The first thought would be to go through all of the key combinations of the Enigma.

For example, starting from 10 plugboard wirings of:

```
(AB)(CD)(EF)(GH)(IJ)(KL)(MN)(OP)(QR)(ST)
```

the letters in the same parenthesis denotes that they are plugged together, and therefore swapped.

We would try the rotor setting A, A, A for rotor 1, rotor 2, rotor 3 correspondingly, and try to decrypt the excerpted message and see if it matches the plaintext message. If it doesn’t match, we move on to A, A, B, Then A, A, C, etc.

After we have tried every rotor setting and found no match, we move on to another plugboard wiring, for example:

```
(AB)(CD)(EF)(GH)(IJ)(KL)(MN)(OP)(QS)(RT)
```

and start over.

In the worst case, we would have to go through all 2,649,375,920,297,106,000 combinations, for each possibility of the plugboard and the rotor. This type of attack algorithm is known as a brute-force attack,

in which the attacker tries every possible key. We can use computers to speed it up; appended is a program that I wrote to automate the process. Reference Appendix A and B for the code written in Python.

The program chooses the plugboard wirings by, first, choosing 20 letters from the alphabet, and second, arranging them in pairs—somewhat different from the plugboard combination formula from before. This process has the following combinations:

$$\frac{\frac{20!}{2^{10}}}{10!} \cdot \frac{26!}{20! \cdot 6!}$$

We can confirm that we are indeed counting the same number of plugboard combinations by simplifying the terms and seeing that it is same as the plugboard combination calculated before:

$$= \frac{20!}{10! \cdot 2^{10}} \cdot \frac{26!}{20! \cdot 6!} = \frac{26!}{10! \cdot 6! \cdot 2^{10}}$$

For each of these plugboard combinations, the program goes through the 26^3 combinations of the rotors, effectively going through all possible keys of the Enigma.

Here is the program running, outputting the number of combinations of keys it has tried per time period:

```
[redacted]/Math Exploration Enigma Machine/Bombe_Brute_force$ python Bombe.py

# Current Time: 2017-10-23 21:11:39.150671
# Current Testing Plugboard Wiring: [('A', 'B'), ('C', 'D'), ('E', 'F'), ('G', 'H'), ('I', 'J'),
# Current Test Key Count: 439400

# Current Time: 2017-10-23 21:12:06.357505
# Current Testing Plugboard Wiring: [('A', 'B'), ('C', 'D'), ('E', 'F'), ('G', 'H'), ('I', 'J'),
# Current Test Key Count: 878800
```

Fig. 4.3. Command line execution of the Bombe Simulation.

However, after many minutes of waiting, the program still seems to keep on calculating. We can see that the time interval between the key 439400 and 878800 was 27 seconds, meaning that trying 439,400 keys took 27 seconds. To try all 2,649,375,920,297,106,000 combinations, we can extrapolate that it will take:

$$\frac{2{,}649{,}375{,}920{,}297{,}106{,}000}{439{,}400} \cdot 27 = 162{,}797{,}336{,}932{,}230 \text{ seconds} \approx 45{,}221{,}482{,}481 \text{ hours} \approx 5{,}162{,}269 \text{ years}$$

Unfortunately, our program will take at most 5 million years to find out the key. We could be lucky and find the correct key earlier, but it is unrealistic to assume so. Also, as mentioned above, the key changes every 24 hours, and we can't wait for too long. Therefore, **we need a much better algorithm of finding the key.**

It is interesting to see that even an encryption mechanism developed decades ago, cannot easily be defeated by a modern computer that can try thousands of keys a second. This shows that the Enigma is, indeed, a well designed encryption device, as well as the fact that we need to think of better tactics, using mathematics and logic, to find out a way of cracking the device. There is a limit on the brute-force algorithm no matter how much processing power we have, and therefore, we must develop smarter ways.

As the plugboard provides the biggest number of combinations, we would want to reduce this number. We can use logic and mathematics to rule out some plugboard combinations, as developed by the mathematician Alan Turing.

3. Elimination of plugboard settings by mathematical induction

We can think of the Enigma machine as a composite function of the plugboard and the rotors. We assume, for simplicity, that all three rotors and the reflector is one function:

$\texttt{Rotors at position n} = R_n(x)$
$\texttt{Plugboard} = P(x)$

Therefore, the complete Enigma machine would be described as:

$\text{Output} = P \circ R_n \circ P(\text{Input})$

or, as a single function, at rotor position n:

$E_n(x) = P \circ R_n \circ P(x)$

We also know that **the plugboard is a self-inverse function** because of its wiring. For example, a plugboard with the wires connected between A and K, would get A and output K, and equally get K and output A.

$P(x) = P^{-1}(x)$

We have also previously established the fact that due to the property of the reflector, the rotors and the reflectors combined have a pair of letters that are connected together. For example, if A was inputted to the rotors and P was the output, an input of P would output A from the same rotor box at the same position. Therefore, **the rotor and the reflector together is also a self-inverse function.**

$R(x) = R^{-1}(x)$

Using these facts, we can deduce:

$P \circ R_n \circ P(x) = P^{-1} \circ R_n^{-1} \circ P^{-1}(x)$
$\therefore E_n(x) = E_n^{-1}(x)$

Therefore, we can see that **Enigma itself is also a self-inverse function.**

The Enigma as a composite function can also be drawn as a diagram, with the box P representing the Plugboard function, and R_n representing the Rotors and the Reflector at position n:

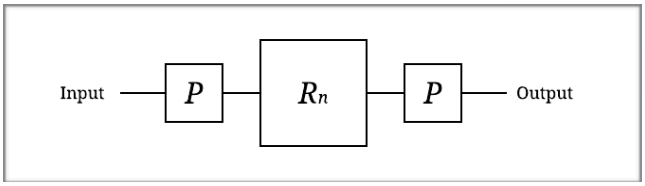


Fig. 4.4.1 The Enigma Machine modeled as functions.

(From now on, this visual representation of the functions will be added next to the written functions for convenience)

Let us assume that the rotor setting of the Engima, is, say, X, E, E. (At rotor position $n = 1$, the rotor setting would be X, E, E, and at rotor position 2, X, E, F, and on.

Let us also assume one setting of the plugboard. For example, that E is connected to U:

$\texttt{U} = P(\texttt{E})$

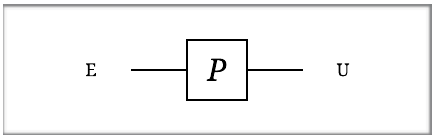


Fig. 4.4.2

We know from the plaintext and cyphertext alignment that at position 2, (Where rotor setting will be XEG) E will encrypt to J.

```
Position                1 2 3 4 5 6 7 8 9 10111213
Encrypted Message:      X J K O T V H E U L V T P
Position 2:             W E A T H E R R E P O R T
```

$J = E_2(E)$

$J = P \circ R_2 \circ P(E)$

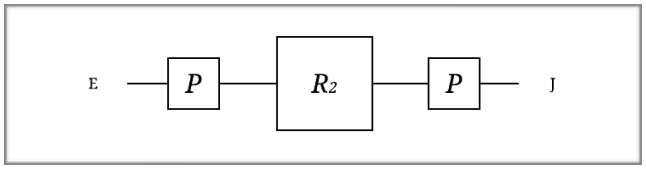


Fig. 4.4.3

According to our assumption of (EU), the output from the first plugboard, and therefore the input letter of function R, will be the letter U.

Since we do have an Enigma machine, we can figure out what U will encrypt to at rotor position 2, by setting our Enigma machine's with our assumed rotor position. I constructed the following table, using code from Appendix A, for easy reference:

Rotor Position	1	2	3	4	5	6	7	8	9	10	11	12	13
Plaintext: U	J	R	F	V	W	T	B	M	S	C	N	V	M

Table 4.4. The results of the enigma encryption at each rotor position.

So, for example, $J = R_1(U)$, $R = R_2(U)$, and so on. We are interested in rotor position 2, and we can see that U becomes R once it passes the rotors and reflectors.

Therefore, we can deduce with this knowledge:

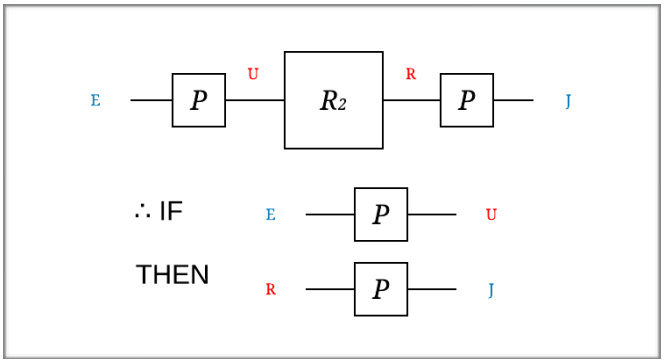
Fig. 4.5.1

$J = P \circ R_2 \circ P(E)$
(Crib)

$U = P(E)$
(Assumption)

$R_2(U) = R$
(Our Enigma machine)

$\therefore J = P(R)$



We have deduced, that if (EU) is correct, then (RJ) is correct as well. In other words, if E is connected to U on the plugboard, then J would be connected to R. We can deduce a few more plugboard settings using the same assumptions:

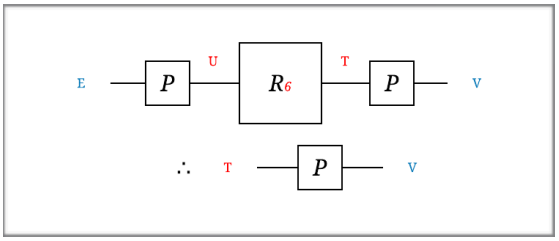


Fig. 4.5.2

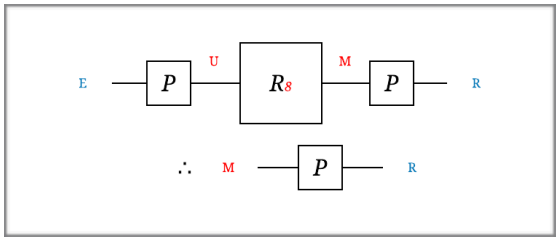


Fig. 4.5.3

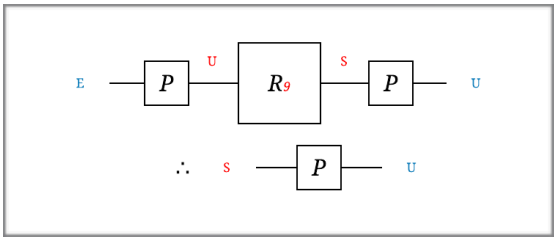


Fig. 4.5.4

However, on the deduction in figure 4.5.4, we have found that (EU) ⇌ (SU). However, U cannot be connected to both S and E. **Our assumption has led to a contradictory conclusion, so this assumption of the plugboard setting is false.**

Also, these propositions goes both ways. If (EU) is false, (RJ), (TV), and (MR) would be false as well.

(EU) ⇌ (RJ)

(EU) ⇌ (TV)

(EU) ⇌ (MR)

(EU) ⇌ (SU)

This means that all of our deductions were false. (EU), (RJ), (TV), (MR), and (SU) are **all eliminated as a possibility.**

4. A Bombe machine's stop

We can use these deductions to rule out the plugboard settings of the Enigma. In the real bombe, there would be 13 simultaneous simulations of the Enigma's rotors each at position 1 to 13, simulating the rotor and reflector, R_1~R_{13}. First we would make our assumption, (EU), and plug the machine accordingly. The circuitry would instantaneously make the deductions of other plugboard settings in the manner that we manually went through.

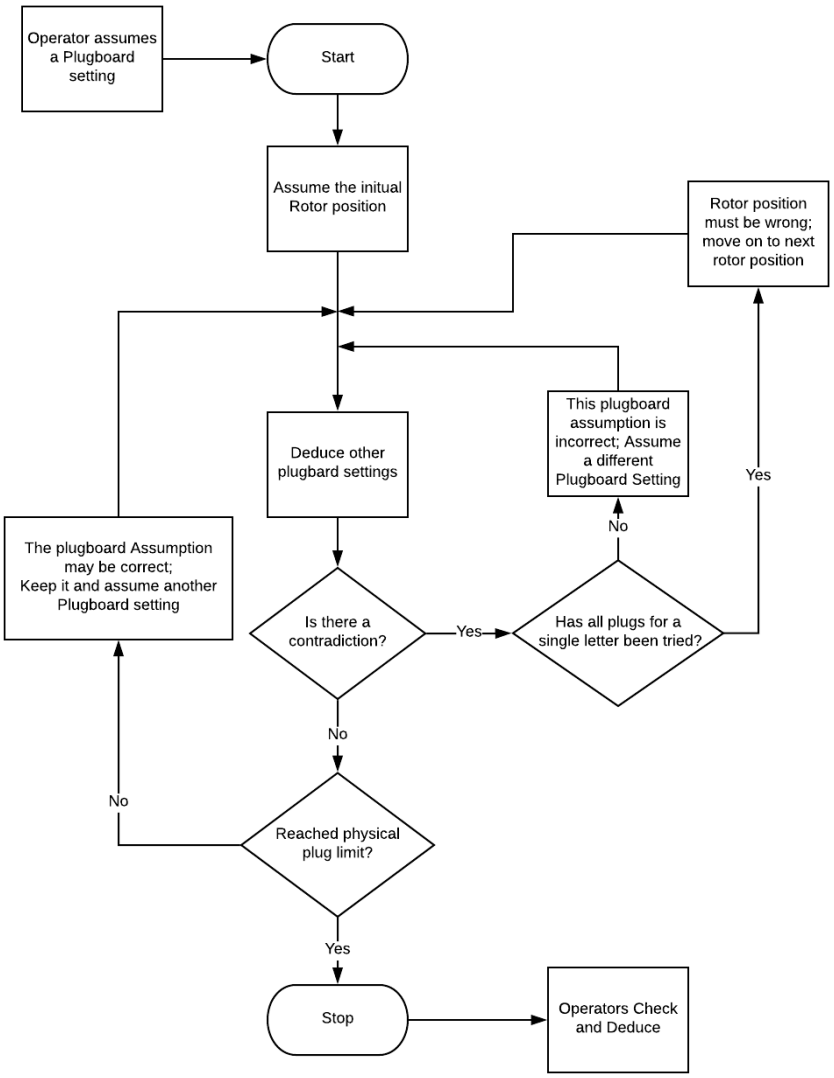


Fig. 4.6 A flowchart describing the operation of a simplified Bombe machine.

If it finds a contradiction, the settings it deduced from the previous assumption would be eliminated. If the machine found that all plugboard settings for a certain letter was all contradictory, (for example, if (EA), (EB), (EC) … (EZ) is all contradictory) this would mean that the rotor setting is clearly incorrect, and it would move on to the next rotor setting, abandon the current plugboard settings, and restart the same process.

Note that our initial assumptions may be abandoned by the machine. The initial assumption from the operator does not have to be correct; the bomb machine will simply find a contradiction, and reject that setting.

If it does not find a contradiction, the assumed plugboard setting would be a correct candidate. It keeps that plugboard setting, and makes another assumption, repeating the process of checking it again. Due to the physical limit of the machine, it would not be able to completely deduce all the plugboard settings. If a candidate for a correct plugboard setting is found, the machine would "stop," and human operators would come to check.

These steps are illustrated in the flowchart, figure 4.6.

The operators would further check if the "stop" is the correct key. It is still possible that the machine stopped simply by coincidence; it is probable that there is a key that does not induce any contradictions, but is not correct. By trying the decrypt the cyphertext using a separate Enigma machine and comparing it to the crib, we can deduce the remainder of the plugboard settings to see if our key decrypts the message correctly.

The following process is an example of how an operator would check the stop, and deduce the plugboard settings. For example, our stop reveal the following possible key:

< Key Candidate 1 >

- Plugboard = (EA)(CF)(GL)(HI)(KP)(MS)(NR)
- Rotor setting (position 1) = X, E, F

Note that the Bombe machine have only deduced 7 of the 10 plugboard settings. We can test if this setting is leading us to the right key, and if it is, deduce the remaining 3 plugs by setting an enigma machine with that key, and trying to decrypt our message:

```
Position              1 2 3 4 5 6 7 8 9 10111213
Encrypted Message:    X J K O T V H E U L V T P

Decrypted using Key1: T E A X O E R R E P O Q T
Crib:                 W E A T H E R R E P O R T
```

The message decrypted with our key reveals that there are matches with the crib at position 2, 3, 6, 7, 8, 9, 10, 11, and 13 suggesting that it is likely part of the correct key.

At position 5, the crib is H. As (HI) is a suggested plugboard setting by the current key candidate, we can trace it back; the value output by the rotors at position 6 would be I. We can use an Enigma with rotors at position 5 (X, E, K), and find that:

$R_5(\mathrm{I}) = \mathrm{W}$

The encrypted letter is T, so therefore, we can deduce that:

$P(\mathrm{W}) = \mathrm{T}$

$\therefore\ P = \ldots(\mathrm{WT})\ldots$

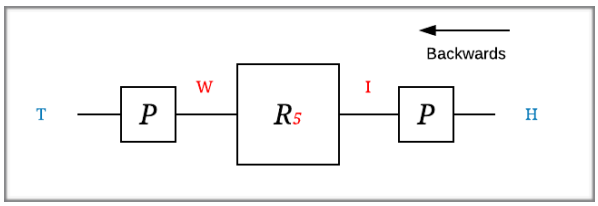


Fig. 4.6.1.

Through such a process, the operators were able to deduce the remainder of the plugboard settings, and, get the correct key.

However, there can be many coincidences where the plugboard setting may not induce a contradiction, but is still wrong. These "false stops" which the operators had to check by hand decreased the speed of the bombe, and therefore, another technique had to be developed.

5. Loops, and self-inverse plugboard function

Alan Turing developed an additional method for reducing the false stops on the bombe machine, using “loops”. Using the crib and the cyphertext, we can draw what was called a menu, which shows which letters encrypt to which, on what rotor position.

For example, in rotor position 2, E encrypts to J, and it can be drawn as in figure 4.7.

Fig. 4.7.

If we draw all of the encryptions, we can make a “menu” diagram, like figure 4.8, which models what each letters encrypt to in what position.

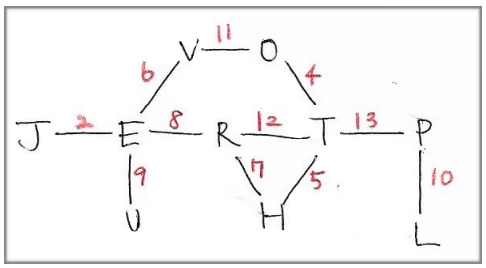


Fig. 4.8. A "Menu” Diagram

Now, we can see that there is a “loop” in R, H, and T. This means that:

$E_7(R) = H$
$E_5(H) = T$
$E_{12}(T) = R$

and therefore:

$\therefore E_{12} \circ E_5 \circ E_7(i) = i$
where $i =$ (a letter variable).

Expanded out, this would look like:

$$P \circ R_{12} \circ P \circ P \circ R_5 \circ P \circ P \circ R_7 \circ P(i) = i$$

We also know that the plugboard is a self-inverse function because of its wiring, as established in section IV-3.

$$P(x) = P^{-1}(x)$$

Therefore, as two self-inverse functions cancel each other out, we can simplify the previous composite function to:

$$P \circ R_{12} \circ R_5 \circ R_7 \circ P(i) = i$$

Apply $P(x)$ to both sides:

$$R_{12} \circ R_5 \circ R_7 \circ P(i) = P(i)$$

We can use this property to greatly reduce the number of false stops, as this condition is much harder to satisfy as a coincidence. The bombe machine would have multiple rows of 13 rotors, interconnected by plugs that model this loop. Even if the plugboard assumptions did not lead to a contradiction, if this loop was not satisfied, the machine would reject this plugboard setting.

When a stop occurs, the operator can then use the previous deduction method to check the bombe machine’s stop, and deduce remaining plugboard settings to get a complete key.

III. Conclusion

In any encryption method, the system is as strong is its weakest link. Breaking the Enigma, just looking at its huge number of combinations, seems like an impossible challenge. Even using a modern computer, it would take millions of years to try every one of them. The key to breaking a system like this lies in using logic and mathematics; the Enigma was not broken because the British built a fast machine; it was because bright minds like Alan Turing or Gordon Welchman, found weaknesses in its implementation such as the fact that a letter never encrypts to itself, and the plugboard being a self-inverse substitution cipher, and used it to develop algorithms that exploited these weaknesses.

I have stated that the purpose of encryption is a uniquely humane thing; that we have developed encryption because we are human, because we value privacy. Breaking encryption, as it seems, is also a uniquely human act. Future computers, ones that are millions of times faster than ones right now, may be expected to crack the encryption algorithms that we use today. However, during the process of exploring how to breaking the Enigma, I have learned that that is not the case; it is the methods that we create that break encryption. The computer, or the Bombe, is merely a tool that is included in the process; the algorithm that humans create, the ingenuity and intuition required to think of them, is what breaks encryption.

> *A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect, a universal machine.*
>
> *— Alan Turing*

IV. Bibliography

Rijmenants, Dirk. “Technical Details of the Enigma Machine.” *Cipher Machines and Cryptology.* users.telenet.be/d.rijmenants/en/enigmatech.htm. Accessed 22 Oct. 2017.

Dade, Louise. “Enigma Machine Emulator.” enigma.louisedade.co.uk. Accessed 22 October 2017.

“Simulation.” *Enigma v.4.3.* enigmaco.de/enigma/enigma.html. Accessed 22 October 2017.

“Bombe: Breaking the Enigma Cipher.” *Crypto Museum*, 30 September 2017. www.cryptomuseum.com/crypto/bombe/. Accessed 22 October 2017.

Smith, Chris. “Cracking the Enigma code: How Turing’s Bombe turned the tide of WWII.” *BT*, 02 November 2017. home.bt.com/tech-gadgets/cracking-the-enigma-code-how-turings-bombe-turned-the-tide-of-wwii-11363990654704. Accessed 22 October 2017.

Baines, Rupert. “How did Alan Turing's Bombe machine work?.” *Quora*, 10 Dec 2013. www.quora.com/How-did-Alan-Turings-Bombe-machine-work. Accessed 22 Oct. 2017

“The Turing Bombe Simulator.” 25 March 2015. www.lysator.liu.se/~koma/turingbombe/ Accessed 22 October 2017.

“The Turing Bombe Simulator: Tutorial.” www.lysator.liu.se/~koma/turingbombe/TuringBombeTutorial.pdf. Accessed 22 Oct. 2017

Oberzalek, Martin. “Breaking the Enigma.” 4 April 2000. www.mlb.co.jp/linux/science/genigma/enigma-referat/node6.html. Accessed. 22 Oct 2017

“The Bombe” www.quadibloc.com/crypto/ro2453.htm. Accessed 22 Oct 2017.

Young, Brad. “The Math That Saved the World: A Mathematical and Historical Analysis of the Cryptographic Attacks on the Nazi Enigma Machine.” *SlideShare*, 22 Apr. 2009. www.slideshare.net/BradYoung/cracking-the-enigma-machine-rejewski-turing-and-the-math-that-saved-the-world. Accessed 22 Oct. 2017

Bellovin, Rebecca. “Cracking the Enigma.” wwwf.imperial.ac.uk/~rbellovi/writings/enigma.pdf. Accessed 22 Oct. 2017

Grime, James. "Maths from the talk Alan Turing and the Enigma Machine.” *Singing Banana*, http://www.singingbanana.com/enigmaproject/maths.pdf. Accessed 22 Oct. 2017.

Armstrong, Sarah, et al. “Puzzled: The Underlying Mathematics of the Enigma.” *Supercomputing Challenge*, 4 April 2007. http://supercomputingchallenge.org/06-07/finalreports/63.pdf. Accessed 22 Oct. 2017

Hosgood, Steven. “All You Ever Wanted to Know About Banburismus but were Afraid to Ask.” *Stoneship*, Jul 2008. http://stoneship.org.uk/~steve/banburismus.html.

Appendix A: The Enigma Machine, simulated in Python

There are 5 classes, and therefore 5 files included in the code. The user is expected to run `Enigma.py`.

- Enigma.py
- Plugboard.py
- RotorBox.py
- Reflector.py
- Rotor.py

< Enigma.py >

```
'''
The Engima class describes an Enigma machine. An Engima machine has an input
mechanism,
an output mechanism, a plugboard, a rotorbox containing 3 rotors, and a
reflector.

@author: Pyokyeong Son
@date: 2017-10

'''

from RotorBox import RotorBox
from Reflector import Reflector
from Plugboard import Plugboard

class Enigma:

    def __init__(self, rotorSettings, plugBoardWiring):
        self.rotorBox1 = RotorBox(rotorSettings)
        self.reflector1 = Reflector("B")
        self.plugboard1 = Plugboard(plugBoardWiring)

    def encrypt(self, inputValue):
        inputValue = self.plugboard1.plugThrough(inputValue)
        self.beforeReflection = self.rotorBox1.getRotorBoxOutput(inputValue)
        self.afterReflection = self.reflector1.reflect(self.beforeReflection)
        return self.plugboard1.plugThrough(
            self.rotorBox1.getInverseRotorBoxOutput(
            self.afterReflection))

def main():
    rotorSetting = [0, 0, 0]
    for i in range(3):
        rotorSetting[i] = int(raw_input("Rotor Settings #" + str(i+1) + " : "))
        '''


    plugBoardWiring = { #Input plugboard as a Dictionary
    }

    enigma1 = Enigma(rotorSetting, plugBoardWiring)
    plainText = raw_input("Input String: ")
    for i in range(len(plainText)): print enigma1.encrypt(plainText[i]),

if __name__ == "__main__": main()
```

<Plugboard.py>

```
'''
The Plugboard class describes an Enigma's plugboard. The wiriing is currently
set to the wiring for 1941-08-17, during Operation Barbarossa, accoring to
http://cryptocellar.org/bgac/HillClimbEnigma.pdf this article.

@author: Pyokyeong Son
@data: 2017-10

'''

class Plugboard:

    def __init__(self, plugBoardWiring):
        self.plugBoardWiring = plugBoardWiring

    # Simple substitution according to the wiring
    def plugThrough(self, inputValue):
        if inputValue in self.plugBoardWiring:
            return self.plugBoardWiring[inputValue]
        else: return inputValue
```

< RotorBox.py >

```
'''
The RotorBox class describes the encryption done by the 3 rotors in the rotor
of the enigma. This only handles the rotors, not the plugboard or the reflector.

@author: Pyokyeong Son
@date: 2017-10

Limitations:
- Can only hold 3 rotors
- Rotor order is fixed
'''

from Rotor import Rotor

class RotorBox:

    alphabet = [
        "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", ]

    def __init__(self, initialStep): # assumes rotor order is I II III
        self.rotorStep = initialStep

        # Make 3 rotors, I, II, and III
        self.rotorA = Rotor(1)
        self.rotorB = Rotor(2)
        self.rotorC = Rotor(3)

    def getRotorBoxOutput(self, inputLetter):
        # Encrypts the letter through 3 forward rotor substitutions
        self.rotorStep[2] += 1

        inputLetter = self.inputTranslator(inputLetter, self.rotorStep[2])
```

```
        self.outputC =
self.outputTranslator(self.rotorC.rotorSubstitution(inputLetter),
self.rotorStep[2])
        self.outputC = self.inputTranslator(self.outputC, self.rotorStep[1])

        self.outputB =
self.outputTranslator(self.rotorB.rotorSubstitution(self.outputC),
self.rotorStep[1])
        self.outputB = self.inputTranslator(self.outputB, self.rotorStep[0])

        self.outputA =
self.outputTranslator(self.rotorA.rotorSubstitution(self.outputB),
self.rotorStep[0])

        return self.outputA

    def getInverseRotorBoxOutput(self, inputLetter): # Encrypts the letter
through 3 backwards rotor substitutions
        inputLetter = self.inputTranslator(inputLetter, self.rotorStep[0])

        self.outputA =
self.outputTranslator(self.rotorA.rotorInverseSubstitution(inputLetter),
self.rotorStep[0])
        self.outputA = self.inputTranslator(self.outputA, self.rotorStep[1])

        self.outputB =
self.outputTranslator(self.rotorB.rotorInverseSubstitution(self.outputA),
self.rotorStep[1])
        self.outputB = self.inputTranslator(self.outputB, self.rotorStep[2])

        self.outputC =
self.outputTranslator(self.rotorC.rotorInverseSubstitution(self.outputB),
self.rotorStep[2])

        return self.outputC

    def outputTranslator(self, inputValue, step): # Corrects the output signal
of the rotors for the Stepping
        index = self.alphabet.index(inputValue) - step
        while index < 0: index += 26 # Handling underflow; e.g. index = -10
        return self.alphabet[index]

    def inputTranslator(self, inputValue, step): # Corrects the input signal of
the rotors for the Stepping
        index = self.alphabet.index(inputValue) + step # Handling overflow; e.g.
index = 36
        while index > 25: index -= 26
        return self.alphabet[index]
```

< Reflector.py >

```
'''
The Reflector class describes the Reflector in the Enigma. There are two types
of reflectors that can be used: the B, or C, both used by the German military.

@author: Pyokyeong Son
@date: 2017-10

'''
```

```
class Reflector:

    reflectorWiring = {
    "B": {
        # reflector B
        "A":"Y", "B":"R", "C":"U", "D":"H", "E":"Q", "F":"S", "G":"L", "H":"D",
        "I":"P", "J":"X", "K":"N", "L":"G", "M":"O", "N":"K", "O":"M", "P":"I",
        "Q":"E", "R":"B", "S":"F", "T":"Z", "U":"C", "V":"W", "W":"V", "X":"J",
        "Y":"A", "Z":"T"},
    "C": {
        # reflector C
        "A":"F", "B":"V", "C":"P", "D":"J", "E":"I", "F":"A", "G":"O", "H":"Y",
        "I":"E", "J":"D", "K":"R", "L":"Z", "M":"X", "N":"W", "O":"G", "P":"C",
        "Q":"T", "R":"K", "S":"U", "T":"Q", "U":"S", "V":"B", "W":"N", "X":"M",
        "Y":"H", "Z":"L"}}

    def __init__(self, reflectorType):
        # Choose between the two reflectors
        self.reflectorDictionary = self.reflectorWiring[reflectorType]

    def reflect(self, inputValue): # Simply reflects
        return self.reflectorDictionary[inputValue]
```

< Rotor.py >

```
'''
The Rotor class describes an Enigma rotor. It can be one of the three rotors
that was used by the German military during WW2.

@author: Pyokyeong Son
@date: 2017-10

Limitations:
- Ignores ring settings
- Only has 3 rotors to choose from

'''

class Rotor:

    rotorWiring = [{
        # M3 Rotor I
        "A":"E", "B":"K", "C":"M", "D":"F", "E":"L", "F":"G", "G":"D", "H":"Q",
        "I":"V", "J":"Z", "K":"N", "L":"T", "M":"O", "N":"W", "O":"Y", "P":"H",
        "Q":"X", "R":"U", "S":"S", "T":"P", "U":"A", "V":"I", "W":"B", "X":"R",
        "Y":"C", "Z":"J"}, {
        # M3 Rotor II
        "A":"A", "B":"J", "C":"D", "D":"K", "E":"S", "F":"I", "G":"R", "H":"U",
        "I":"X", "J":"B", "K":"L", "L":"H", "M":"W", "N":"T", "O":"M", "P":"C",
        "Q":"Q", "R":"G", "S":"Z", "T":"N", "U":"P", "V":"Y", "W":"F", "X":"V",
        "Y":"O", "Z":"E"}, {
        # M3 Rotor III
        "A":"B", "B":"D", "C":"F", "D":"H", "E":"J", "F":"L", "G":"C", "H":"P",
        "I":"R", "J":"T", "K":"X", "L":"V", "M":"Z", "N":"N", "O":"Y", "P":"E",
        "Q":"I", "R":"W", "S":"G", "T":"A", "U":"K", "V":"M", "W":"U", "X":"S",
        "Y":"Q", "Z":"O"
        }
    ]

    def __init__(self, rotorNumber):
        # The rotor's dictionary is set according to what it is
```

```
        self.rotorDictionary = self.rotorWiring[rotorNumber-1]

        # An inverse dictionary, for when the signal is traveling backwards
        self.rotorInverseDictionary = {v: k for k, v in
self.rotorDictionary.iteritems()}

    def rotorSubstitution(self, inputValue): # Performs a forward substitution
        return self.rotorDictionary[inputValue]

    def rotorInverseSubstitution(self, inputValue): # Performs a backward
substitution
        return self.rotorInverseDictionary[inputValue]
```

Appendix B: A simplified, brute force, Enigma deciphering program

This code is intended to be run with all the files in Appendix A.

< BruteForce.py >

```
from Enigma import Enigma
import itertools
import datetime

class BruteForce:

    alphabet = [
        "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", ]

    def __init__(self, cypherText, plainText):
        self.cypherText = cypherText
        self.plainText = plainText
        self.counter = 0

    def iterateRotor(self):
        for i in range(26):
            for j in range(26):
                for k in range(26):
                    enigma1 = Enigma([i, j, k], self.plugBoardWiring)
                    count = 0
                    while True:
                        if enigma1.encrypt(self.plainText[count]) ==
self.cypherText[count]:
                            if count >= (len(self.plainText)-1): return [i, j,
k]
                            count += 1
                            continue
                        else:
                            break
        return -1

    def iteratePlugboard(self):
        choose10 = list(itertools.combinations(self.alphabet, 20))
        for lst in choose10:
            for x in self.all_pairs(list(lst)):
                self.plugBoardWiring = dict(x)
                self.plugBoardWiring.update({v: k for k, v in
self.plugBoardWiring.iteritems()})
                self.iterateRotor()

                self.counter += (26*26*26)
                if (self.counter % 100) == 0:
                    print "\n# Current Time: " + str(datetime.datetime.now())
                    print "# Current Testing Plugboard Wiring: " + str(x)
                    print "# Current Test Key Count: " + str(self.counter)

                isPlugCorrect = self.iterateRotor()

                if isPlugCorrect != -1:
                    return [isPlugCorrect, self.plugBoardWiring]

    def all_pairs(self, lst):
        if len(lst) < 2:
            yield lst
```

```
    return
a = lst[0]
for i in range(1,len(lst)):
    pair = (a,lst[i])
    for rest in self.all_pairs(lst[1:i]+lst[i+1:]):
        yield [pair] + rest
```