Java vs Enigma

Term Project Draft Report

Trevor Ward and Jacky Chen
Introduction to Computer Programing for Scientists and Engineers
Submitted April 24th, 2019
Dawson College

Introduction

It wouldn't be an exaggeration for someone to say that scientists or engineers are glorified problem-solvers. Maybe we're saying that because most, if not all of our time in the science program is spent learning scientific material and answering questions based on said material. Despite that, you cannot deny that a big part of a scientist's job is to find the solutions to mankind's questions. These questions can vary from field to fields, such as biology and chemistry. However, for our CE project, we decided to take a look at solving a man vs machine problem that is at the roots of machine-learning; how fitting for this course. It is the Enigma encryption machine used by German forces in WW2 (Figure 1).

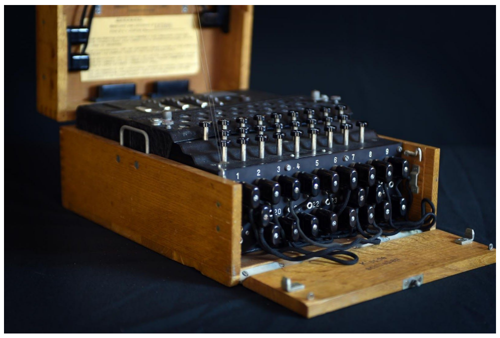


Figure 1

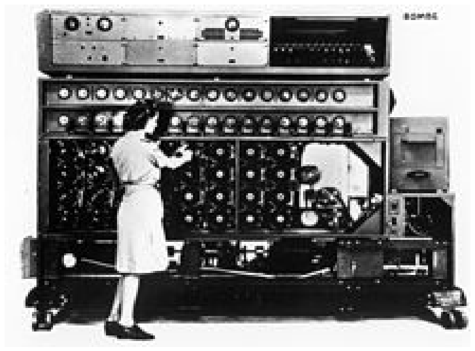


Figure 2

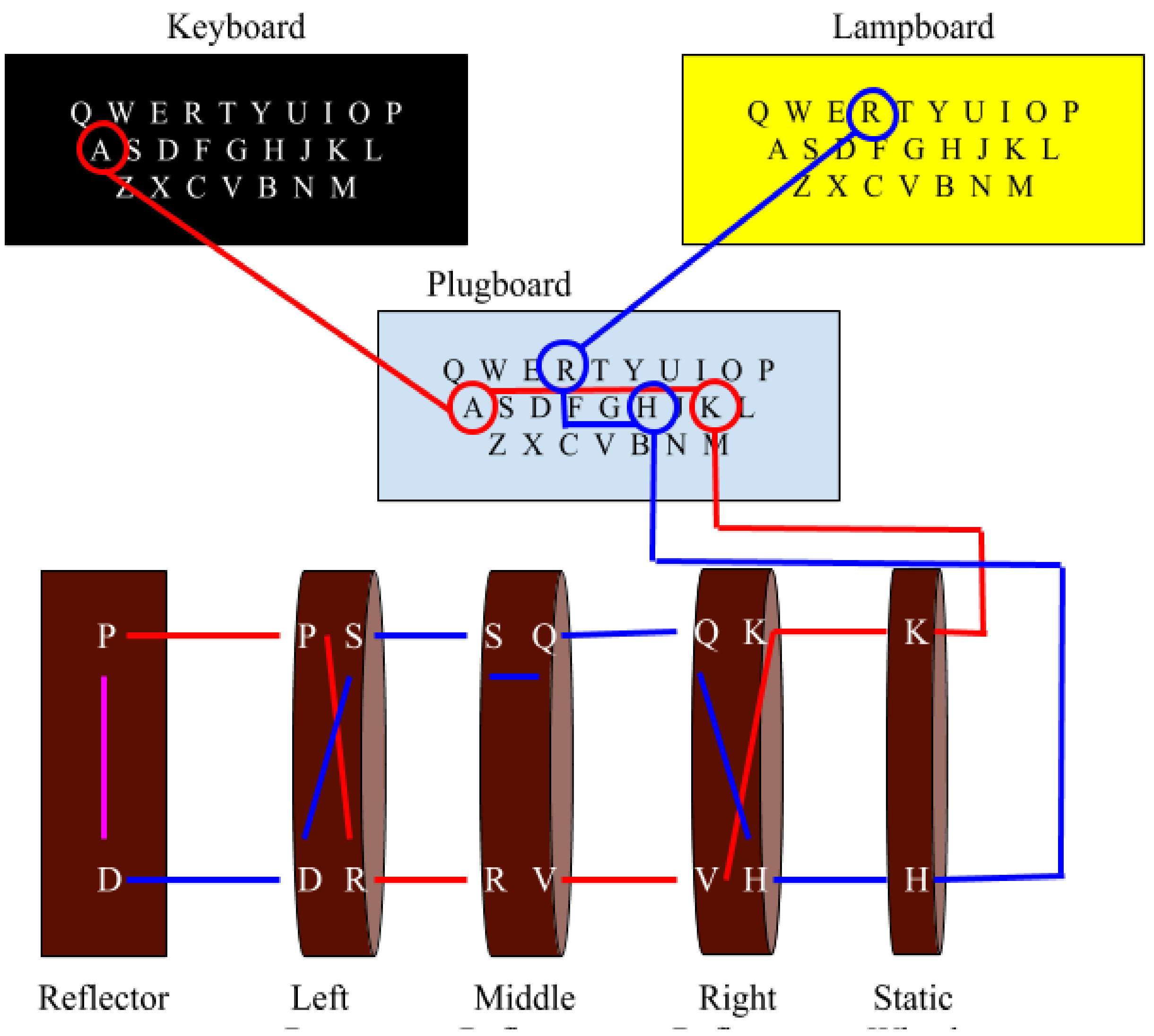
Enigma is essentially an electromechanical typewriter that encodes a message and was one of the first encryption codes that would scramble its keys while typing a message. This means that typing the same letter would give different outputs each time. Add that to the fact that there are many presets to choose from and you get is a code that is nearly impossible to crack (around 156 * 10^18 possibilities). Fortunately, allied forces were able to crack the code using a machine of their own (figure 2), which would take some 20 minutes to crack the code on a good day. Our question for our term project is is it possible to write a Java code capable of solving Enigma?

Understanding Enigma

The way the Enigma machine works is quite simple; a person would choose from a possibility of 156 * 10^18 presets, type a letter in the keyboard, and the encrypted letter appears on the lampboard. The beauty of the machine is that if an encrypted message is typed into the keyboard with the same presets, the result on the lampboard will be the original message. During the war, the encrypted messages were sent by telegraph where someone else with the same presets on their Enigma machine would type in the encrypted message and get the real thing. Our task becomes writing a Java code that can decipher an encrypted message and tell us what are the correct presets to unscramble the message. The components - each with their own presets - that make Enigma so hard to crack are the plugboard and the five rotors.

Message Flow

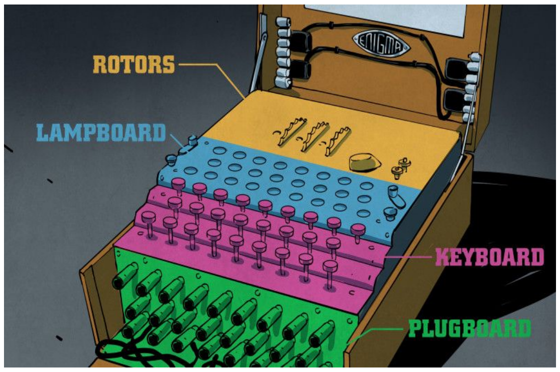
The purpose of the plugboard and rotors is to scramble and change the designation of the typed letter. As the electric current goes went from the keyboard to the lampboard, the letter changes or stays the same 9 times. A typed message flows from the keyboard, through the plugboard to the 3 rotors, through a reflector which sends the message back across the 3 rotors, back through the plugboard and finally to the lampboard as the encrypted message.



In this example above, the letter A is typed into the KB, and after passing through the PB, the 3 rotors and back, the letter R appears on the LB.

Plugboard

The plugboard works by connecting two letters together by a plug. For example, if the letter “a” is connected to the letter “s” on the PB, then when “a” is typed, it becomes encrypted as “s” before continuing through the machine to be further encrypted. Also, if the typed letter somehow becomes encrypted as “s” after going through the machine, then the final encrypted letter becomes “a” since “a” and “s” are connected. There are ten plugs connecting two letters together. If a letter is not connected, it doesn't change before going through the machine or to the lampboard. 10 plugs connecting two wires means a possible 26! / (6!*10!*2^10) settings, equal to 150,738,274,937,250 settings.

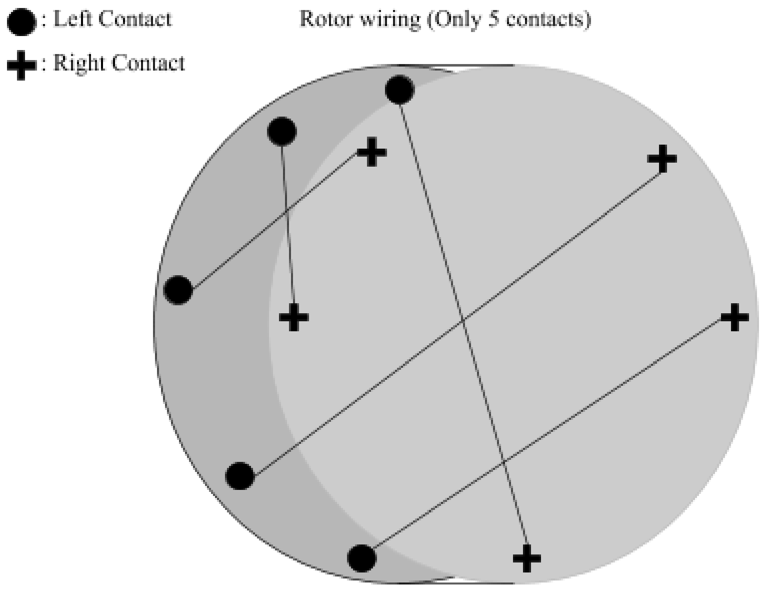


Plugboard

Rotors

Each rotor has 26 contacts (one for each letter) on each side. Each contact on one side is connected to another on the other side by a wire. This is what allows the letter to be scrambled. 3 rotors go in the machine and there are 5 to choose from, each with their own specific inner wiring and contact configuration. Each rotor has a position counter, from 1 to 26, and the position of each rotor is shown next to it on the Enigma machine. Therefore the position of each rotor can be decided before writing a message. Each time a letter is typed, the right rotor rotates and its position counter increases by 1. When the right rotor’s counter passes from 26 to 1, the middle rotor’s counter increases by 1, and when the middle rotor passes from 26 to 1, the left rotor’s counter increases by 1. The reflector doesn't and can't be moved. It has 26 contacts and 13 wires that connect 2 contacts together so the electrical current can go back through the rotors. The static rotor also doesn't move and serves to take whatever letter it receives from the PB and assign it to its letter contact and vice versa.

If there are 5 rotors to choose from, that's 60 possible rotor arrangements, and each rotor has 26 start positions, which is 26^3 possible position presets. Therefore, 60 * 26^3 = 1,054,560 possible rotor presets. Multiplied by the number of possible plugboard arrangements gives us a total of 158,962,555,217,826,360,000 possible presets. Now we can see why Enigma is so hard to crack, but maybe not with the right Java code and an HP laptop.



Methods of Solving

Like any code, to break Enigma we need a key, that is, we need to know what input gives which output when we type it. What makes Enigma special is that each time you type a letter, the rotor moves, and the key changes. You could type "a a a a a" into the KB and "f h j f o" would appear on the LB, however, a letter typed will never appear as itself on the PB, and that is the flaw in Enigma that is key in cracking the code. We do however, need predictive text to crack to code. The codebreakers at Bletchley Park during WW2 knew that the Germans sent out a weather report every morning at 6:00 am with the word "Wetterbericht" (Weather report in German) as the first coded word. All German communications also ended with "Heil Hitler". Wetterbericht was their predictive text. If we put ourselves in the shoes of original codebreakers, we know that whatever the first line of the encrypted message comes from the telegraph has to contain the word "Wetterbericht". Coded messages were also written as individual 4 letter words. For example, if we receive a coded message and our predictive text is "COMPUTER", we have to align our predictive text with the coded word and make sure none of the aligned letters match (this being the Enigma's flaw). If the line of coded message is " CZAT RUSZ TBET", we align "computer" with the message without any matches. Example:

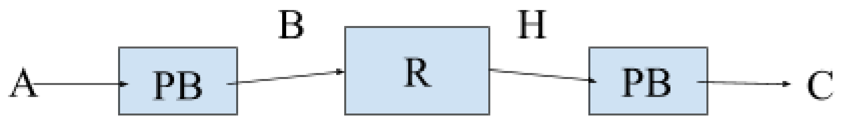
```
CZAT  RUSZ  TBET  - Here C aligns with C so it's no good.
COMP UTER

CZAT  RUSZ  TBET  - Here T aligns with T. still no good.
   C OMPU  TER
```

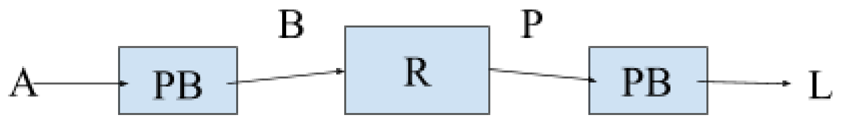
CZAT RUSZ TBET
CO MPUT ER

This is the only case that works. No letters match. Since Enigma also lets you type in a coded message to get the original so long as you have the same presets, we can determine that with certain presets, typing "A T R U S Z T B" will give us "C O M P U T E R".

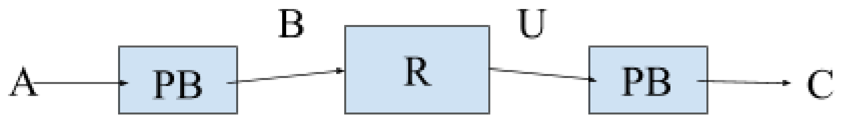
The next step requires a lot of trial and error. We essentially have to test every preset and find the one that works, find the one that will give us our predictive text. We start with the first of our 60 rotor arrangements and set the counters to 1-1-1. We then start with our first coded letter "A" and make an assumption as to which letter it is connected to on the PB. In our case, we'll start with "B". Since we have an exact copy of our counterpart's Enigma machine, we know how of the five the rotor contacts are wired, therefore when we hit "A" plugged into "B" and at rotor position 1-1-1 with arrangement 1 of 60, we can determine what letter will come out of the rotors and therefore which two other letters are connected on the PB.



In our example above, typing A gives us B which turns into H after going through the rotors, and since we know that typing A gives us C, we can determine that H is connected to C. After tying A the rotor position moves to 1-1-2. Therefore when we type A again, we should get a different result from the rotors.



If we continue with our example, we can now see that typing A plugged into B at preset 1-1-2 gives us P from the rotors and L from the LB, meaning P and L are connected. The rotor counter moves to 1-1-3 and we keep typing A until we run into an error, for example:



After typing A a certain amount of times we discover that U connects to C on the PB, but we had already determined that H connected to C in our first step. Since C cannot be connected to two letters, we can conclude that our assumption that A connects to B on the PB was wrong, so we make the assumption that A connects to the next letter, C, reset the counter to 1-1-1 and try again. We go through every possible connection for A, even A not connected to anything and if none of those PB settings work, then it means that our initial rotor preset of 1-1-1 was wrong, so we start all over again with an initial rotor preset of 1-1-2. If none of the 26^3 rotor presets work, it means that our rotor arrangement assumption was wrong, and we go to arrangement 2 of the possible 60. Eventually we get the correct rotor arrangement and preset as well as PB settings that match the one that encrypted the code.

Solving Enigma

In order to solve Enigma, we would need the presets, at least a database containing what the presets are.

Comparing the answer

There is most likely no answer key to an Enigma decryptor code. The only way to make sure that our code works is by making sure that any given input gives out the expected output given the correct presets. We can either work it out by hand on paper or check our answers with a reliable and functional code that decrypts Enigma. We would use the code online or a Enigma machine simulator and set it to the specific presets. Then, by imputing the initial letters of our message, we would get the encrypted message. If we reverse the process; imputing the encrypted message back into the Enigma simulator with the correct we should get the the initial message back. Thus, to make sure that our code works, we should obtain the same encryption/decryption as the online simulator given a specific preset.