Java vs Enigma

Term Project Report

Trevor Ward and Jacky Chen

Introduction to Computer Programing for Scientists and Engineers

Submitted May 22nd, 2019

Dawson College

For our term project, we wanted to simulate the Enigma encryption machine used by German forces in WW2. 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. What is particular about this encryption machine is that the combination of different plugboard settings, rotor chosen and presets (being the initial positions of the rotors) would generate around possibilities. Moreover, a similar input (letter) would have different outputs. Therefore, no patterns can be found in the encrypted message. Back in WW2, it was a true feat to break the code as it was a man vs machine problem. We wanted to tackle this problem to test how modern technology would fare against the German encryption code; how difficult would it be to generate that machine nowadays using Java. To verify whether our code works or not, we would use a Enigma simulator and compare the inputs and outputs of a given message given the plugboard, rotor and presets settings.

Q W E R T Y U I O P

A S D F G H J K L

Z X C V B N M

Q W E R T Y U I O P

A S D F G H J K L

Z X C V B N M

Q W E R T Y U I O P

A S D F G H J K L

Z X C V B N M

Keyboard

Lampboard

Reflector

Left

Rotor

Middle

Reflector

Right

Reflector

Static Wheel

P

D

P

S

D

R

S

R

Q

V

Q

V

K

H

K

H

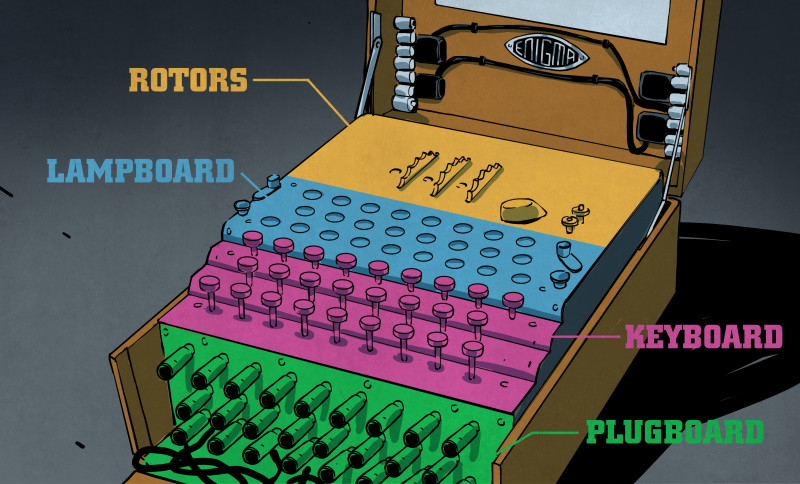
Plugboard

How does Enigma work?

To decrypt the Enigma machine, we must first understand how it works; how does a message get encrypted or decrypted. The decryption works by inputting the crypted message with the same presets and plugboard and rotor combination. Therefore, we would have to both simulate Enigma and decrypt it by exploiting its flaws.

As briefly explained above, before a message is even decided, a person would choose from a possibility of presets made up by the plugboard and the rotors. When a key is pressed, an electric signal is sent through the plugboard and rotors. Then, a deflector reflects the signal back into the rotors, back into the plugboard once more and the result is displayed by a lightbulb lighting under the correct letter.

The plugboard consists of two letters connected to each other at a time to scramble the message. Say “a” is connected to “s”; when pressed, the letter “a” becomes “s” before moving onto the next encryption step. A similar process happens once more before the result is displayed by the light bulbs. Note that there are only ten plugs connecting two letters together. If a letter is not connected, it doesn't change before going through the machine or to the lamp board. 10 plugs connecting two wires means a possible settings, equaling 150,738,274,937,250 possibilities.

 The next step of encryption are the different rotors that scrambles the letters further. Each rotor has 26 contacts (one for each letter) on each side connected to another on the other side by a wire allowing the letter to be scrambled. In terms of possible combinations, there are five rotors to choose from to encrypt a message given a specific preset. Every 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 so on for the other rotor. The reflector doesn't and can't be moved; it is the only constant part in Enigma. 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 plugboard 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 263 possible position presets. Therefore, 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.

Simulation

Before attempting to crack the code, we thought it would be better to write a code that simulates an actual Enigma machine. This way we can work off of the simulation code when writing our decrypting code. The simulation lets the user choose their own Enigma settings (plugboard configuration, rotor arrangement and rotor presets). Once the settings are selected, they are then prompted to enter their message that they wish to encode. PB connections are stored in a character array, while we made hashmaps for all 5 rotors, as well as their inverse and the reflector, and stored them in methods which would call for whatever letter is being coded as well as the preset of the rotor. The code breaks down the message and runs each letter at a time, increasing the rotor count each time. If the user chooses to run the code again with the same settings and this time wrote the encoded message, the code would return his original message because of the reflector rotor. We can test our code against online Enigma simulators to make sure it works.

Methods of Solving

Like any code, to break Enigma we need a key; we need to know the output of a specific input. Recall that every time a letter is pressed, the rotor moves. Thus, the key changes. Any letter can be typed for any amount of times and a different output without any pattern would show. Although this was a strength, it was also a flaw, being a letter could never reappear as itself. Hence, we would need predictive a text (or “crib”) 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 must contain the word “Wetterbericht”. Coded messages were also written as individual 4 letter words. For example, if we receive a coded message and our crib is “COMPUTER”, we must align our crib with the coded message 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 the user to type in a coded message to get the original so long as the presets remain the same, 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”. In our solving code, the user would input both the message and the crib, and character arrays would sort each line of text, while a while loop would find where along the message does the crib not match any letters, determining the correct configuration.

The next step requires a lot of trial and error. We essentially must test every preset and find the one that works, find the one that will give us our crib. We start with the first of our 60 rotor arrangements and set the counters to 1-1-1. Therefore, having a Java coded simulation of the Enigma machine comes in handy. We then start with our first coded letter “A” and assume 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.

A

PB

R

PB

C

B

H

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. Thus, when we type the next letter of the message, T, and again, whatever letter appears from the rotors must be connected on the PB to the second letter or the crib.

T

PB

R

PB

O

T

P

If we continue with our example, we can now see that typing T at preset 1-1-2 gives us P from the rotors but must be O according to the crib. We can now connect P and O on the PB and keeps going. The rotor counter moves to 1-1-3 and we keep typing out the message until we run into a contradiction:

R

PB

R

PB

M

R

O

After typing R we discover that O connects to M on the PB, but we had already determined that O connected to P in our second step. Since O cannot be connected to two letters, we can conclude that our assumption that A connects to B on the PB was wrong, so we assume 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 263 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. In our code, we would have a loop with a boolean condition set at false, and whenever a contradiction occurs the loop would restart, while updating the initial PB connection, and possibly rotor preset and rotor arrangement. The solving code would run through the simulation code to determine the outcome of letters once typed into the machine.

Finally, when the settings are found, they are printed out to the user, who can confirm by writing the same message in the simulator to get the crib as the encryption.

Results

While we were able to write a code that simulated Enigma, we did not however manage to finish writing a code that would solve it, even though all our general ideas or methods to solve it are known. We compared our Java simulator to an online simulator and ours was able to decrypt messages made with the online version.

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 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.

Discussion

What can be concluded is that we have successfully managed to simulate an Enigma machine using Java. By understanding how the Enigma machine works and how did the codebreakers during WW2 solve Enigma helped us create our own version of the Enigma machine on Java. Based on the different flaws in Enigma, we would be able to write a code that works by assumptions and use an eliminative process to narrow down the answer. Another method would be using brute force with more powerful computational power. What could be interesting is to try to decrypt using brute force and see how efficient our computers are compared to the bombe machine used in WW2. We can also compare brute force decrypting to exploiting the flaw and see how much more time efficient it would be.

References

1- Quick history of the Enigma machine:

<https://www.iwm.org.uk/history/how-alan-turing-cracked-the-enigma-code>

2- Enigma simulator

<http://users.telenet.be/d.rijmenants/en/enigmasim.htm>

3- Enigma message flow and machine design

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.692.9386&rep=rep1&type=pdf>

4-Plugboard image

<https://www.google.com/search?q=enigma+machine&safe=off&rlz=1C1CHZL_enCA725CA725&source=lnms&tbm=isch&sa=X&ved=0ahUKEwi0usrJz6viAhWOnFkKHakEBQMQ_AUIDigB&biw=1366&bih=625#imgrc=jx6rEkh_XWQwFM:>

5- Enigma diagram

<https://www.google.com/search?q=enigma+machine&safe=off&rlz=1C1CHZL_enCA725CA725&source=lnms&tbm=isch&sa=X&ved=0ahUKEwi0usrJz6viAhWOnFkKHakEBQMQ_AUIDigB&biw=1366&bih=625#imgrc=9qIlrlcK5NZh0M:>

6- How to solve Enigma

<https://calculate.org.au/2015/02/03/crack-enigma-code/>