

Software Implementation of the Enigma Machine in Python

Mathew Ellison¹, Frank Stomp²

Navajo Technical University
P.O Box 849 Crownpoint, NM 87313, USA

¹mattellison00@gmail.com

²fstomp@navajotech.edu

Abstract During World War II the German military employed an electromechanical, typewriter-like machine to encrypt and decrypt messages. Encryption converted the *plaintext* into gibberish text, the so-called *ciphertext*. The plaintext could be quickly recovered from the ciphertext when some secret information was known. The German Enigma machine was a rather sophisticated machine for the time to ensure secure communication. The current paper presents a description of a software implementation using Python 3.

Keywords: Cryptography, Enigma, Python

1. Introduction

During WW2, the German military employed an electromechanical, typewriter-like machine to ensure secure communication. Dubbed the German Enigma machine, it converted the original text – the *plaintext* – into unintelligible text – the *ciphertext*. The cipher text could be easily transformed back into the original text, provided some secret information was known. Without that secret information obtaining the plaintext from the ciphertext was hard. This method of secure communication is an example of symmetric key encryption (Schneier, 2015) because the sender and receiver must share secret information, called the *key*, to securely communicate. The current paper presents a description of an implementation of (a version of) the German Enigma Machine in Python 3.

The Enigma machine had many different versions and was created in the 1900s for use in the banking and railway industries. It was patented in London in the 1920's (TheNaziGermany1945, 2013). The German military saw potential use for the Enigma Machine in the military for secure communication. For its time, the underlying encryption techniques were quite sophisticated. A picture of the Enigma Machine can be found in (London, 2017). A description of how it worked can be found in (Rijmenants, 2021) and in (Kahn, 1996).

The version of the German Enigma machine we consider simulating in software is one that had to be operated using three rotors, see (Sale, p. 2), chosen

from a collection of five rotors. These rotors could be rearranged in different orders and set in different character positions depending on what the operator decided to use. (Extending our implementation to more rotors is straightforward.) During the war, every morning new instructions were used to set up the Enigma Machine. These instructions were on a setting sheet that had the day, order of the rotors, and the ring setting of each rotor. Each ring setting described the initial turn of that rotor. The instructions also included the plugboard set-up. The plugboard offered an extra layer of security by transposing characters. (More details are provided in Section 2 below.) Sale (Sale, p. 3) states that the total number of combinations is (even for the simplest version of the German Enigma Machine) at least of the order of 15,000,000,000,000,000.

It was uncrackable until Alan Turing and his colleagues found a way to analyze the ciphered messages. For example, Alan Turing and his colleagues used a machine called the Bombe to decipher messages produced by the Enigma Machine. They realized that the Enigma Machine could not encrypt a character with itself. For example, an 'A' could not be encrypted as an 'A'. This flaw reduced the number of potential character conversions dramatically. Another flaw that caused the Enigma Machine to be cracked was the so-called double indicator. This was a technique to encode initial settings of the rotor, by an operator, and its transmission twice. (Repetition in encryption is not good.)

2. Enigma Machine properties

The Enigma machine uses rotors that contained the 26 letters of the alphabet that move in a counterclockwise manner. Each rotor had wire ends at each character position around the rotor that sent the electric current through a path, depending on which wire was connected. The machine has three rotors connected. The Enigma Machine also used a plugboard to add even more security. Its function is to transpose letters. For example, if 'A' and 'X' were paired in the plugboard, then 'X' would be the output if 'A' were the input, and if 'X' were the input, 'A' would be the output.

When a key on the keyboard was pressed, in essence, the right rotor turned counterclockwise. Sometimes it also caused the middle rotor, and even the left rotor, to be turned counterclockwise. Thereafter, that letter is replaced by another letter according to the plugboard setting. The letter is replaced by another letter by the wiring of the right rotor. This process is continued to the middle rotor and then to the left rotor. The resulting character serves as input to the reflector. The reflector, in essence, permutes characters. Then a similar process happens in the reverse direction. Eventually, the output from the right rotor (in the reverse direction) then passes through the plugboard again. Thus, a plaintext letter is converted into a ciphertext letter.

Each component converts the input character into an output character by means of an electric current. Once the cyphertext character has been produced, the current travel through the lamp board and lights up the bulb that is associated with the ciphertext character. For the machine to decrypt the ciphertext and return the original plaintext, both parties need to use the same initial settings.

The Enigma Machine settings were not the only security features. Instead of creating messages in normal form, four letters were grouped together followed by a space. Each row, except possibly the last one, consisted of ten groups. The reason was to prevent outsiders from determining the length of words.

3. Implementation

Our implementation in Python 3 can be found at (github.com/mattellis101/enigma). The rotors, reflectors, and plugboard were modeled using strings. The rotor file can instantiate any of the five rotors. A

string consisting of 26 characters represented each actual rotor. (The characters are capital letters only.) The strings were taken from (Karonen, 2019). We also associate a notch character with each rotor, as used in the Enigma Machine. The notch, once passed, allowed the next rotor to move one step. (Cf. clock hands when reaching the top of the hour.) The reflector file can instantiate any of three reflectors. Each reflector is modeled using strings. We associated them with the label's 'A', 'B', and 'C'.

We also created a file for the plugboard. The plugboard transposes 20 of the 26 characters of the alphabet. The six remaining characters were left as is. The class Plugboard is described in file `plugboard.py`. The actual permutations are in the `plugboard.txt` file. (The `plugboard.txt` file can be modified by the user.)

The user input file contains methods that allow users to choose the Enigma Machine settings. The user can choose three distinct rotors out of the five rotors. Next the user chooses the order of the rotors. The user then chooses which character position (ring setting) each rotor is set to. Once the rotors have been set, the user must pick which reflector to use. Finally, the user is asked if they are encrypting or decrypting a message.

The Enigma file `enigma.py` is the main file. We import the other Python files into the main file. In the Enigma file we created methods that encrypt and decrypt the message. The message is in a text file called `plain.txt`. We also created another text file in the Enigma main file called `cipher.txt`, this is the text file that will contain the encrypted message. When encrypted, the plaintext file is left unmodified but the ciphertext file is overwritten with a new ciphertext. Similarly, when decrypting, the ciphertext file is left unmodified but the plaintext is overwritten with a new plaintext. There are other software implementations of the Enigma machine. For example, an implementation in Python can be found in (Enigma Encoder, 2019). In our implementation of the Enigma machine, we created a more practical way for the user to input the Enigma settings. With the `user.py` file, user input is efficient and Enigma settings are easy to adjust. In our implementation we created classes and followed the object-oriented methodology, which is not present in the code found at (Enigma Encoder, 2019). Another difference between the two implementations is that we organized the code in separate files, called modules in Python.

4. Running the program

To run the Enigma Machine, open the plaintext file on your computer. Write a message with four characters in a group, with ten groups in each row, except possibly the last row. Next run the enigma.py file and the user will be asked a set of Enigma input settings. This is shown in the following figure:

```
Provide the right rotor (1 through 5): 1
Provide the middle rotor (1 through 5): 3
Provide the left rotor (1 through 5): 5
Give me the initial position of the right rotor (A through Z): F
Give me the initial position of the middle rotor (A through Z): D
Give me the initial position of the left rotor (A through Z): V
Provide a reflector (A , B, or C): C
Do you want to encrypt or decrypt? 'e' for encrypt / 'd' for decrypt: e
```

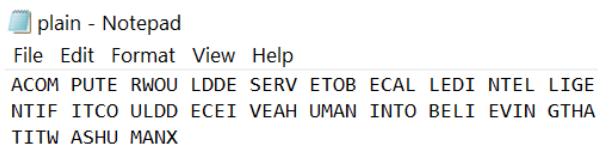
The Enigma program can be run on Python 3.7.4 or a later version. When setting up the Enigma Machine, each setting has a certain limitation. If, for instance, a rotor was chosen that was not one of the five rotors created, a message will let the user know that the option was invalid and to choose from the options listed. Once all the settings are configured, the last question the user must answer is either to encrypt or decrypt a message.

If the user chooses to encrypt a message, the program modifies the cipher.txt file. The user then opens this file and will see the encrypted message. If the user chooses to decrypt a message, the program modifies the plain.txt file.

5. Experimental Results

In order to obtain the original plaintext message from the ciphertext, by decryption, one must use the same initial settings as used during the encryption process. Using different rotor, plugboard, or reflector settings, one will not get the original plaintext back.

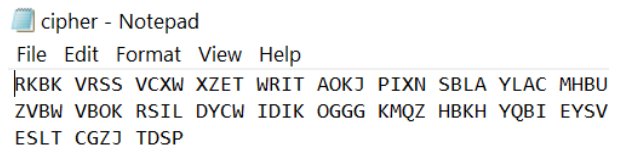
For example, if we encrypt the Turing Test quote by Alan Turing, "A computer would deserve to be called intelligent if it could deceive a human into believing that it was human". (The last character 'X' below indicated a period.)



```
plain - Notepad
File Edit Format View Help
ACOM PUTE RWOU LDDE SERV ETOB ECAL LEDI NTEL LIGE
NTIF ITCO ULDD ECEI VEAH UMAN INTO BELI EVIN GTHA
TITW ASHU MANX
```

```
Provide the right rotor (1 through 5): 5
Provide the middle rotor (1 through 5): 2
Provide the left rotor (1 through 5): 4
Give me the initial position of the right rotor (A through Z): F
Give me the initial position of the middle rotor (A through Z): P
Give me the initial position of the left rotor (A through Z): Q
Provide a reflector (A , B, or C): A
Do you want to encrypt or decrypt? 'e' for encrypt / 'd' for decrypt: e
```

The encrypted message shown below is the result of the Enigma settings chosen above.



```
cipher - Notepad
File Edit Format View Help
RKBK VRSS VCXW XZET WRIT AOKJ PIXN SBLA YLAC MHBV
ZVBW VBOK RSIL DYCW IDIK OGGG KMQZ HBKH YQBI EYSV
ESLT CGZJ TDSP
```

To decrypt the message, the exact Enigma settings used in the encrypting of the message must be used to decipher the message. Therefore, we must enter the same Enigma Settings when decrypting the message.

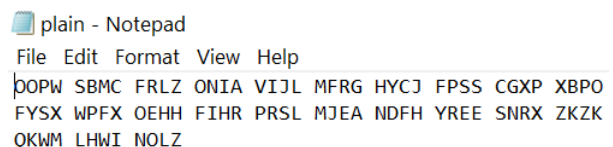
```
Provide the right rotor (1 through 5): 5
Provide the middle rotor (1 through 5): 2
Provide the left rotor (1 through 5): 4
Give me the initial position of the right rotor (A through Z): F
Give me the initial position of the middle rotor (A through Z): P
Give me the initial position of the left rotor (A through Z): Q
Provide a reflector (A , B, or C): A
Do you want to encrypt or decrypt? 'e' for encrypt / 'd' for decrypt: d
```

The output of the decryption process produces the original message that was in the plain.txt file.

The settings in the next example differ in one aspect to the previous example. The initial position of the middle rotor is different. Instead of setting the middle rotor to the character 'P', the rotor is set to the character 'Y'

```
Provide the right rotor (1 through 5): 5
Provide the middle rotor (1 through 5): 2
Provide the left rotor (1 through 5): 4
Give me the initial position of the right rotor (A through Z): F
Give me the initial position of the middle rotor (A through Z): Y
Give me the initial position of the left rotor (A through Z): Q
Provide a reflector (A , B, or C): A
Do you want to encrypt or decrypt? 'e' for encrypt / 'd' for decrypt: d
```

The cipher text was decrypted but the result did not match the plaintext. Therefore, the text produced did not match the original plaintext. This is shown in the following:



```
plain - Notepad
File Edit Format View Help
POPW SBMC FRLZ ONIA VIJL MFRG HYCJ FPSS CGXP XBPO
FYXW WPFX OEHH FIHR PRSL MJEANDFH YREE SNRX ZKZK
QKWM LHVI NOLZ
```

6. Conclusion

Our Enigma program in Python implements one version of the actual Enigma Machines used during WW2. The Python program models the Enigma Machine faithfully.

As sophisticated as the Enigma Machine was in the 1900s, the Enigma Machine was flawed (Enigma, 2009). With today's techniques the Enigma Machine can be cracked in minutes as shown in (London, 2017).

Our implementation can easily be extended to different versions of the German Enigma machine. For example, we used three rotors, but this can easily be modified to add a fourth or fifth rotor. Another part of the Enigma Machine software that can be modified is the plugboard. Instead of permuted ten pairs of characters this can be modified up to thirteen pairs of characters.

Acknowledgements:

This work was supported by grants from the Department of Energy #DE-NA0003946. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Department of Energy.

References:

1. Hodges, Andrew. "Alan Turing: The Enigma" November 2014.
2. Kahn, David. "The Code-Breakers- The Comprehensive History of Secret Communication from Ancient Times to the Internet", 1996
3. Karonen, Ilmari. "Enigma Rotation Example." Crypto.stackexchange.com, June 2019, crypto.stackexchange.com/questions/71231/enigma-rotation-example. Accessed 1 March 2020.
4. London, Andrew. "We Watched an AI Crack the Enigma Code in Just over 10 Minutes." TechRadar, 1 Dec. 2017, www.techradar.com/news/we-watched-an-ai-crack-the-enigma-code-in-just-over-ten-minutes. Accessed 31 May 2021
5. Rijmenants, Dirk, "Technical Details of the Enigma Machine", 2021, users.telenet.be/d.rijmenants/en/enigmattech.htm, Accessed 6 July 2021
6. Sale, Tony. "The Enigma Cipher Machine", www.codesandciphers.org.uk/enigma, Accessed 14 June 2021.
7. Schneier, Bruce. "Applied Cryptography: Protocols, Algorithms and Source Code in C. 20th Anniversary Edition", 2015.
8. TheNaziGermany1945. "Greatest Mysteries of World War 2 Hitler's Engima (sic)." YouTube, 22 June 2013, www.youtube.com/watch?v=y_BLOM08-B4, Accessed 2 Jan. 2020.
9. "Enigma", 07, July 2009, www.cryptomuseum.com/crypto/enigma/working.htm, Accessed 19 July 2021
10. "Enigma Encoder", 29 May 2019, www.101computing.net, Accessed 5 Jan. 2020