2022 International Conference on Big Data, Information and Computer Network (BDICN) | 978-1-6654-8476-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/BDICN55575.2022.00156

Analysis and Illustration of the Enigma Machine

Jiaxin Guo[1, †], Yichen Hu[2, †] and Yixuan Wang[3, †, *]
[1]School of International Education, Guangdong University of Technology, Guangzhou, Guangdong, 511400, China
[2]Lynbrook High School, San Jose, California, 95129, U.S.
[3]Ningbo Hanvos Kent School, Ningbo, Zhejiang, 315202, China
*Corresponding author's e-mail: guanghua.ren@gecacdemy.cn
[†]These authors contributed equally

Abstract. **The Enigma machine, invented in 1918, the last year of World War I, served a significant role in the German military and history of cryptography. With the emergence of radio transmission, military communication faced the revolution of a new media. However, manually encrypting messages, possibly leaking intelligence, didn't meet the military requirement. Encryption machines, which are way more efficient than enciphering messages manually, were in high demand. The Enigma machine was initially to protect commercial privacy. Nazi Germans developed additional mechanisms to adapt to the battlefield. Breaking down the Enigma machine was among the top priorities for the Allies. Because of its crucial rule in cryptography, it is meaningful to examine and explore the Enigma machine. The Enigma encryption scheme was one of the first encryption schemes involving electromechanics, and learning the structure, principle of it can inspire future encryption schemes. Even though the technology has been progressing tremendously afterwards, the rules for processing cipher traffic have remained the same to a great extent, making a study on the Enigma machine more contributive. The investigation conducted explores the security factors of Enigma machines and possible advances for Enigma machines. By implementing the algorithm for each component of the Enigma machine, the research group developed a simulator of the Enigma machine via Python. Through inspections, the research group concludes that the security of the enigma machine was top at its period. Without procedural errors, it was difficult to break down the Enigma machine.**

Keywords. Enigma Machine, Emulator for the Enigma machine, Improvement for the Enigma machine

I.Introduction

The Enigma machine, Arthur Schebius, used in the mid-twentieth century to protect commercial and military communications, was the peak of the classical cipher. Enigma Machines are composed of three rotors, a reflector, a plugboard, and a keyboard. Its appearance and breakdown had a significant impact on WW2, modern cryptography, and cryptanalysis. In 1926, Enigma's Variants were adopted by the German army, navy, and air force. Enigma fulfilled Pre-war German military needs of protecting their radio signals when implementing their high-mobility tactic, the blitzkrieg. Germans developed variants of the commercial enigma to ensure the safety of their messages. They added a plugboard to substitute the input letter to another one, which significantly increased the complexity, and set up more potential rotors to choose the three rotors. The deeper Germans dug into increasing security, the more revisions and adjustments were made. By examining related encryption schemes and constructing an emulator, a thorough study was developed to explore the security, breakdown of enigma machines, and improvements of its mechanism.

The main aspects of this work can be summarized as follows:

1. The physical construction of the Enigma machine and the corresponding encryption scheme.
2. An emulator for Enigma Machine based on Python.
3. Security analysis of the Enigma Machine.
4. How the Polish broke the German Enigma encryption scheme.
5. How Alan Turing broke the Enigma Machine.
6. Improvement on the Enigma Machine.

For the rest of the report, the rest of this paper is organized as follows. Section 2 elaborates the constitution of the Enigma machine and the German encryption scheme. Section 3 analyzes the security of the Enigma machine. Section 4 demonstrated 2 kinds of cryptanalysis for the Enigma machine. Section 5 introduces a simulator for the Enigma machine. Section 6 shows a possible improvement for the Enigma machine. Finally, Section 7 concludes this paper.

II.Constitution of the Enigma machine

The 'Enigma machine' mentioned in this section specifically refers to the Enigma I, developed in 1927/29 by Chiffriermaschinen AG, with 3 rotor slots and 2 extra rotors. It has been one of the most commonly talked-about variants of Enigma [1]. As Figure 1 shows, the Enigma machine consists of several parts [2]:

1. A keyboard with 26 letter keys as the switches of the electric circuits.
2. A lamp board with 26 lettered lamps.
3. A plugboard that contains 0 to 10 dual-wired cables.
4. A reflector to revert the input back to another contact on the same side of the wheel.
5. Three rotors (selected out of five) internally wired input contacts to output contacts.

In this section, all components above are explained thoroughly.

978-1-6654-8476-3/22/$31.00 ©2022 IEEE
DOI 10.1109/BDICN55575.2022.00156

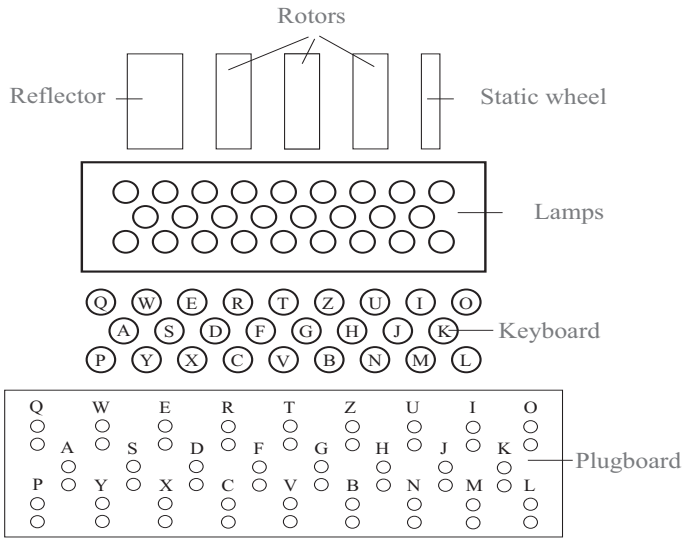

Figure 1. Components of an Enigma machine.

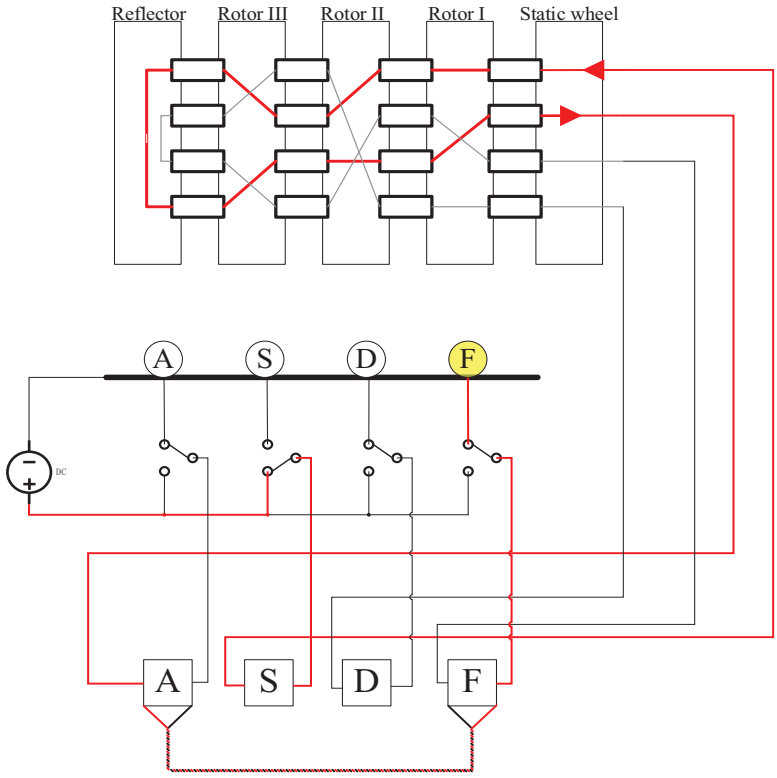

Figure 2. Simplified internal circuits of the Enigma machine.

A.Keyboard & lamps

There are 26 keys for each letter of the alphabet on the keyboard. These keys act as switches in the electric circuits. As the user presses any key, the electric circuit will enclose so that a 'random' lettered lamp will be lit. It has the arrangement as follows:

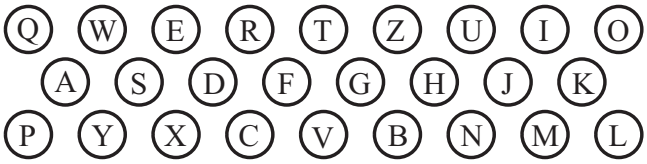

Figure 3. The keyboard layout of the Enigma I.

B.Rotors, stepping and turnover

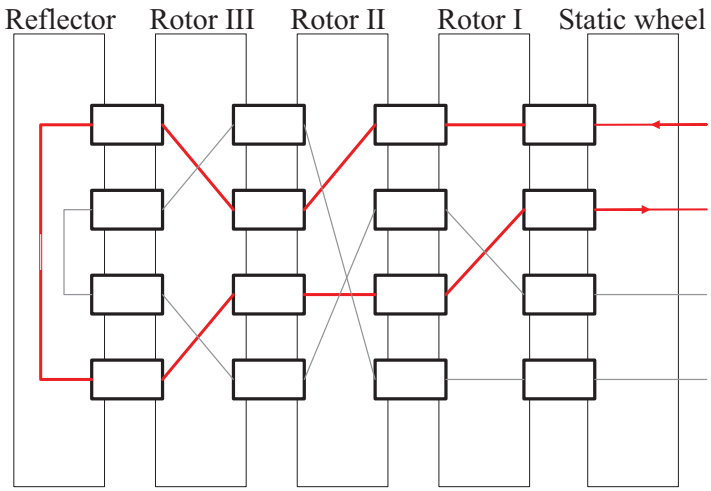

Figure 4. Intrinsic electric connections in the rotors.

Rotors (alternatively wheels or drums, Walzen in German [3]) are coded wheels arranged along a spindle under the lid of the machine. There are 26 crisscrossed wires inside every rotor to transfer the electric currents from one to another. Accordingly, there are 26 electric contact pins on one end of the wires or one side of the rotor. And at the other side of the rotor, there are 26 electric contact pads. As complicated as it is, before the Polish and British got access to an actual machine, discover the internal wiring was considered insoluble especially by pure cryptanalytical means [4]. There are 26 indentations or a ratchet with 26 teeth on the flange of every rotor. And the operator can choose any starting position by pressing the protruding edges of the rotors. The rotor will rotate once (1/26 full rotation) after any key is pressed and before the next electrical connection is made. This mechanism is called 'stepping'. The intrinsic quality of the Enigma cipher machine is a kind of substitution cipher. Stepping prevents implementing a single mapping table of Substitution Cipher, which is easy to solve with means like frequency analysis etc. Every rotation is equivalent to changing a new mapping table. By passing electric currents through the mechanism, it produces highly scrambled 26 letters of alphabet [4].

There is a notch on every rotor which allows the advancement of itself, except the one closest to the reflector. For every specific rotor, the positions of the notches remain the same. This process is called 'turnover'.

C.Plugboard

If the front flap of the machine is folded down, the plugboard and its 26 pairs of sockets can be seen. The plugboard (also known as steckerboard or "cross-plugging board") was introduced to the German Army in 1928 [5]. This front panel allowed the operator to reconfigure the wiring of the connection by swapping pairs of letters. Each Enigma machine comes with 12 cables but only 10 of them can be used. Each cable can be connected to any 2 letter socket with both its ends so as to swap those to letter in the electric circuits. Not all 10 (but up to 10) cables are required to be plugged. If there were no swaps required by the operator, the plugboard can be left completely unplugged.

Figure 5. Plug Cable.

The plugs are 2-pin. On each plug, the upper pin is thicker and the other is relatively thinner, which means they cannot be

plugged upside down (see Figure 5 [6]). The thick pin is cross-connected to the thin pin at the other end of the cable [1].

"The purpose of the steckerboard is to vary the interconnections between the in-out terminals of the scrambler unit and the letter- keys and lamps [4]."

D.Reflector

The reflector,umkehrwalze in German and called the "turn-around wheel" by the people in the Hut Six [4], is considered a half-rotor because it doesn't rotate [2]. What it does is sending electric currents coming from the right, which means the rotors, back to the rotors and the static wheel by different routes.

The explanation of its detailed structure is left to Section 6.

III.Security analysis of the Enigma machine

In this section, we are going to analyze the security of the Enigma machine from the aspects below:

1. The number of configurations.

2. Other factors that might affect the security.

A.Mathematical analysis

It is known that the Enigma machine was huge trouble for the Allies due to its massive amount of possibilities for its configurations and output. And the reason for it is the variables of the components.

1)Plugboard. Variables of the plugboard needed to be considered are as follows:

1. The number of cables used.

2. Socket selections.

3. Group combinations in the selected sockets.

There are 26 sockets on the plugboard and each cable occupies 2 sockets. Thus, 13 cables at most can be used simultaneously. However, most Enigma variants only came with 12 cables, the spare 2 cables included. Given the number of x cables ($0 \le x \le 12$)plugged into the sockets at the same time and x cables take up $2x$ sockets. Thus, there are C_{26}^{2x} or C_{26}^{26-2x} combinations for socket selections in total.

For the first cable, there are C_{2x}^{2} choices, which means $2x$ sockets available because the plugboard is idle. After the first cable is inserted, there will be $2x$-2 socket left. And for the next cable, the number of free sockets is $2x$-4 so on and so forth. For the last cable, only 1 choice is going to be left. Therefore, the number of possible groups, N_1, can be formulated as follows:

$$N_1 = \frac{C_{2x}^{2} \times C_{2x-2}^{2} \times \cdots C_{2}^{2}}{A_x^x} \tag{1}$$

The denominator of this math expression is A_x^x, or $x!$ because the permutations for any x pairs of letter-sockets should not be taken into consideration.

Given $\boldsymbol{x}$ cables inserted into the plugboard, the amount of different connections, which a German operator in WW2 could have chosen from by setting the plugboard, is the product of the 2 expressions, which can be formulated as follows:

$$\begin{aligned} C_{26}^{2x} \times N_1 &= C_{26}^{2x} \times \frac{C_{2x}^{2} \times C_{2x-2}^{2} \times \cdots C_{2}^{2}}{A_x^x} \\ &= \frac{26!}{x! \times (26-2x)! \times 2^x} \end{aligned} \tag{2}$$

The math expression (2) can also be interpreted in another way [7]: The number of combinations of 26 letters is 26!. Nevertheless, some elements, such as the permutations of the x pairs of sockets, the order of the 2 sockets in all x pairs and also the arrangement of the left 26-2x sockets, are required to be excluded.

In Table 1 [2], it shows that the Combinations are even more complicated if 2 cables are left unplugged.

Table 1. Combinations according to different value of x.

x	Combinations	x	Combinations
0	1	7	1,305,093,289,500
1	325	8	10,767,019,638,375
2	44,850	9	53,835,098,191,875
3	3,453,450	10	150,738,274,937,250
4	164,038,875	11	205,552,193,096,250
5	5,019,589,575	12	102,776,096,548,125
6	100,391,791,500	13	7,905,853,580,625

2)Rotors. Variables of rotors that needed to be considered are as follows:

1. Starting position.

2. Rotor selection.

It was mentioned that there are 26 protruding flanges in the rotor and the German operator could change the starting point by pressing them. Each rotor has 26 starting positions which means there are 26×26×26 or 26^3 configurations by changing the starting positions.

The Enigma I came with five rotors. The operator could choose any three of them to insert into the machine. For the left-hand side slot, there are 5 choices. For the middle slot, there are 4 left for one was selected. Hence, for the last slot, 3 rotors will still be free. For the rotor selections, the number is 5×4×3, which is 60.

Therefore, the number of configurations that can be generated by setting starting position and selecting rotors, N_2, can be formulated as below:

$$N_2 = 5 \times 4 \times 3 \times 26^3 = 1{,}054{,}560 \tag{3}$$

3)The number of configurations. The final result should be combined with different variables of the machine, which are

rotors and the plugboard. Theoretically, the plugboard allowed the German operator to plug 13 cables but 10 was the maximum number in real practice. So here we assume the value of x is 10. The number of configurations for the Enigma machine can be presented as follows:

$$C_{26}^{20} \times N_1 \times N_2 = \frac{26!}{10! \times (26-2\times 10)! \times 2^{10}} \times 5 \times 4 \times 3 \times 26^3 = 158{,}962{,}555{,}217{,}826{,}360{,}000 \approx 2^{67.1} \quad (4)$$

B.Other factors that might affect the security

Even though we have already acquired the number of configurations that every variable component can make, at the time the Bletchley Park (the Allied code-breaking center in England, UK during WWII) did not have any information of how the rotors were forged. The first input contact could have been connected to any one of the 26 output contact and 25 for the second input contact so on and so forth, which would have made 26! possibilities for any rotors. The total number of rotor combinations could have been $26!^3$, an unacceptable number.

There were also some characteristics of the Enigma machine that made it less secure and could be exploited by the Enigma machine:

1. The connection is reciprocal, which might seem to be innocent.

2. A letter can never be encrypted to any letter but itself.

The first feature might seem innocent but in fact, it was how Gordon Welchman came up with the idea of the *diagonal board*, later introduced to the Bombe. The second one was the feature exploited by Alan Turing to create the Bombe.

IV.Breaking of the Enigma machine

A.How the Polish broke the Enigma cipher

The Enigma machine had great security during its time, but the Polish still found a way to break it. To be more specific, three conditions need to be met, which are knowing the internal structure of the Enigma machine, Germany's operation, and the ability to eliminate the plugboard influence. In this part, all conditions needed will be analyzed in detail, except for the inner structure of the Enigma machine that has already been analyzed in Section two.

1)The operation rules for German soldiers. Germany had a lot of rules for operating the Enigma machine to make it tough for their adversaries to break down. To begin with, the German operator had to set up the Enigma machine according to the monthly sheet which told the setting of the day. Each rotor would be assigned to one number, and the operator would put them in the right order, such as "1-2-3", "2-1-3", "3-1-2"… Then, the operator should adjust 3 rotors so that each rotor had the right initial letters. For example, turning the first rotor to position 'O', the second to position 'P', the third one to 'V'. The last step was to connect 6 pairs of exchanged letters in the plugboard, like 'A' connected to 'F', 'B' connected to 'G', etc.

Besides, the German operators would "randomly" think of 3 letters as the information key of a message. After that, he would start to type on the Enigma machine. The operator would type the character string of the 3 letters twice at the very beginning of the message. Later the 3 letters would become the new key for the rest of the message. For instance, if the operator decided to use "ASK" as the key, he would type "ASK" twice, and the first 6 letters encrypted by the machine would become "QLOKFI" which was the information key of this piece of the message (figure 1). The operator would change the settings to "A-S-K", and start to type the cipher-text. Accordingly, the receiver who had the same settings would first decrypt the "QLOKFI" and see the information key, and then decrypt the ciphertext with the initial rotor position of "A-S-K".

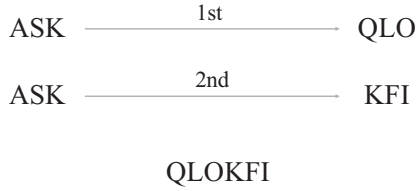

Figure 6. This figure shows how the information key of 3 letters becomes 6 different letters after 2 rounds of typing.

2)The breaking of the Polish. However, the Polish who didn't have the same Enigma settings couldn't use the method mentioned above to break it. Instead, they found their own way. Although they didn't know what those three information key letters were, they did know that the first letter "Q" and the fourth letter "K" in the output "QLOKFI" were encrypted results of the same letter. Now they suppose that the information key was XYZ. They set 'A1' as the symbol for the first rotation and 'A2' for the second, all the way to 'A5'. So they got "X(A0)=Q Y(A1)=L Z(A2)=O X(A3)=K Y(A4)=F Z(A5)=I" (figure 7). Based on the property that the connection of the Enigma machine is reciprocal, "X(A0)(A0)=X". "X(A3)=K" was equivalent to "X(A0)(A0)(A3)=K". Since "X(A0)=Q", it could be simplified to "Q(A0)(A3)=K".

It was decrypted very soon after clarifying the process of achieving the last condition, eliminating the negative influence caused by the plugboard.

Step 1	Set the information key as X Y Z
Step 2	X(A0)=Q Y(A1)=L Z(A2)=O X(A3)=K Y(A4)=F Z(A5)=I
Step 3	X(A0)(A0)=X
Step 4	"X(A3)=K"="X(A0)(A0)(A3)"=K
Step 5	Q(A0)(A3)=K

Figure 7. This figure shows steps of rewriting the information key.

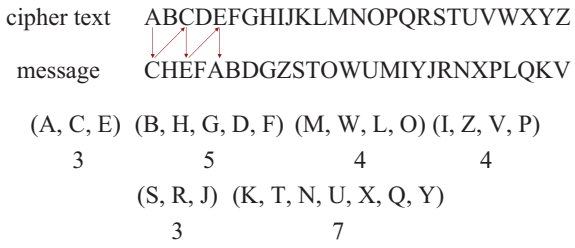

Figure 8. This figure shows the process of Analyzing the eigenvalues

To make it easier to understand, this part will be introduced with a specific example. Assume that there is a row of message "ABCDEFGHIJKLMNOPQRSTUVWXYZ", and the accordant cipher-text is "CHEFABDGZSTOWUMIYJRNXPLQKV". 'A' is matched to 'C'. 'C' is matched to E. And 'E' is matched back to 'A'. In this case, it can be written as "(A, C, E)" form. And 3 is the eigenvalue of the string because it has 3 letters. Analyzing the rest of the letters, we will get "(B,H,G,D,F)", "(M,W,L,O)", "(I,Z,V,P)", "(S,R,J)", "(K,T,N,U,X,Q,Y)" and their eigenvalues are "3, 5, 4, 4, 3, 7" (figure 8) [8] .

If we apply the same method to rewrite "Q(A0)(A3)=K", it will be "(A,C,M,Y,N,Z,Q,D,P,B,O)", "(E,U,I,S,G)", "(F,T,H,R)", "(K,W)", "(J,W)", "(L,X)", and the eigenvalues will be "11, 5, 4, 2, 2, 2" (figure 9). In this way, the Polish classified all letter chains by their eigenvalues and made them a list book, which has around one hundred thousand combinations [9].

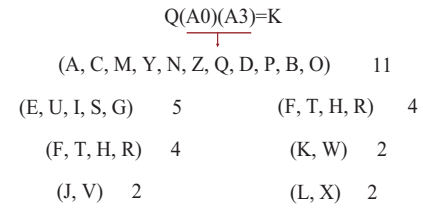

Figure 9. This figure shows the converted form with eigenvalues of the equation, which is "Q(A0)(A3)".

To sum up, the Polish had 3 steps to break down the messages by the Enigma machine, which was decrypting the information key, finding all combinations that met the requirements from the book, and finally brute-forcing all possibilities found from the list book.

B.How the British broke the Enigma machine

"In July 1939 representatives of British and French intelligence were invited to a meeting in Warsaw [10]." Polish shared their breaking with the British and the French and gave them each an Enigma machine.

1)The flaws used in the breaking of the British. After receiving Polish breaking, the British found flaws in the Enigma machine and some certain expressions which frequently showed up in the messages were exploited to break the machine.

The Enigma machine had a very important attribute which allowed the British to break the Enigma machine. "Due to reflector, a letter can never turn into itself, this feature makes easier cryptanalysis [11]." Based on it, the British made use of this attribute to come up with the "crib". The word "crib" means cheats. A crib referred to a certain expression which always showed up in the messages. It could be the name of a place, the salutation or weather forecast and so on.

There were certain patterns in the ciphertext. Numerous messages had the same sentences, such as "keine besonderen Ereignisse" (Nothing to report [12]), "Heil Hitler". By matching these specific words and sentences with ciphertext, the British found a clear relationship between plaintext and ciphertext.

2)The breaking of the Enigma machine. The breaking began with the weather forecast every morning in the ciphertext. The forecast, which was sent at 6 am, had the same words in the beginning. One of them is "wetter", which means weather. The following table demonstrates the "wetter" 's crib in the ciphertext.

Table 2. The "wetter" 's crib

Rotor position			0	1	2	3	4	5
Plaintext 1	W	E	*T*	T	E	R		
Plaintext 2		W	E	T	T	*E*	R	
Plaintext 3			W	E	T	T	E	R
Ciphertext	Q	A	*T*	B	M	*E*	W	C

The letter couldn't be encrypted into itself. Hence only the third guess in the table was possible. 'W' would be encrypted to 'T' though position 0, so on and so forth.

Figure 10 is a simplification of the encryption process. Assume that the letter 'W' becomes V_1 after going through the plugboard. It would go through the set of 3 rotors at position 0 and becomes V_2. It would output 'T' after going through the plugboard.

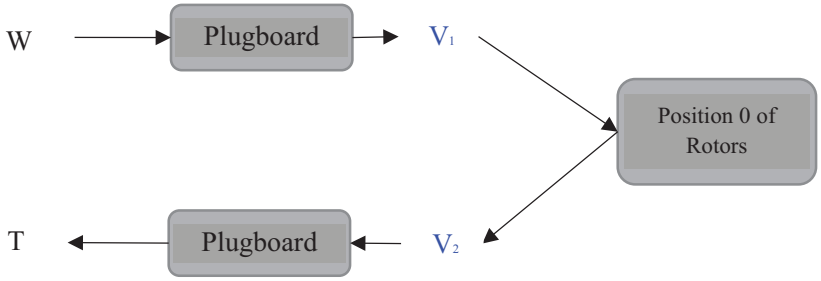

Figure 10. The simplification of the encryption process.

The gain of the Polish from the perspective of the structure of the Enigma machine can be demonstrated as follows.

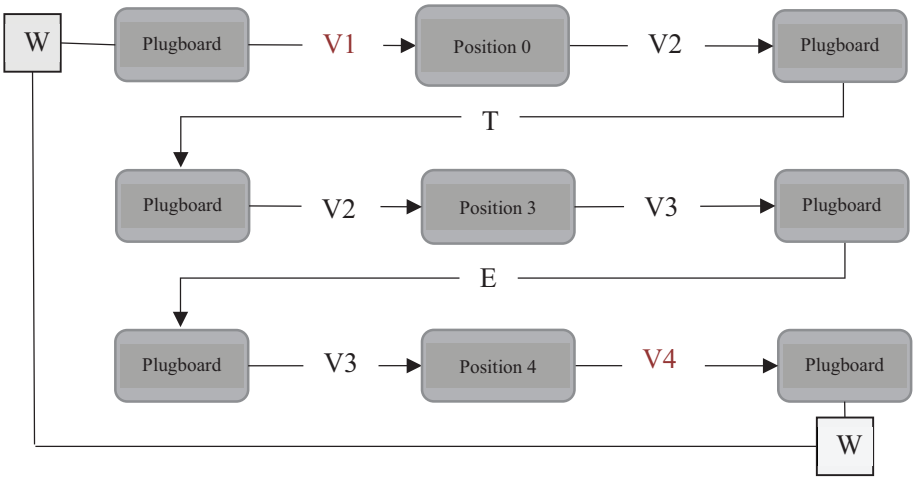

Figure 11. Encryption process.

The plugboard can't be changed in the whole encryption process and the output is the same as the input letter, which is 'W'. Thus, V_1 should be equal to V_4. Three rotor positions in Figure 11 are simulated by three Enigma machines respectively. If V_1 is not equal to V_4, the current guess is incorrect. The British needed to try different guesses until V_1 is equal to V_4. If any guess doesn't fit this pattern, this guess would be incorrect.

Precluding the interference of the plugboard, Figure 11 can be simplified as follows.

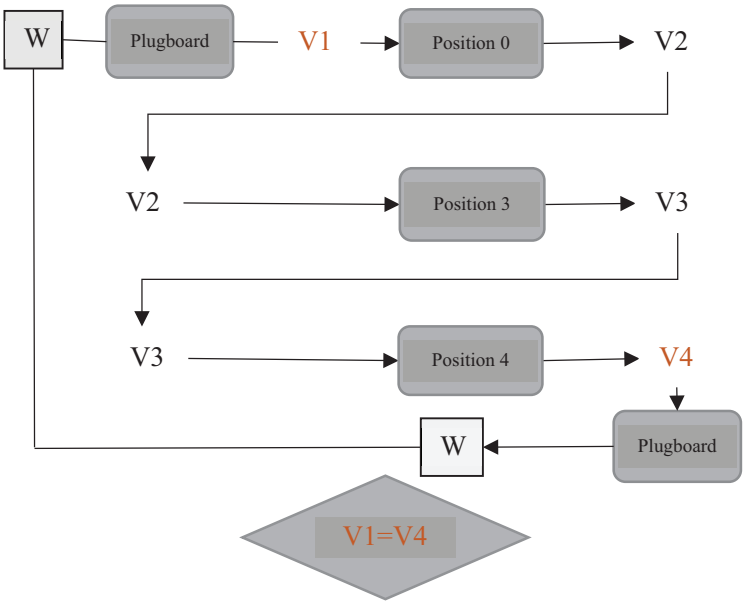

Figure 12. Simplified Figure 2 without the interference of the plugboard.

The process in Figure 12 could still be complicated if it was done manually since there were 26 possibilities for the input V_1 and numerous possibilities for the configurations of the 3 Enigma machines. To conquer that, Alan Turing decided to make the whole process automated and the solution was called the 'Bombe'. Figure 13 is what a Bombe looks like [13].

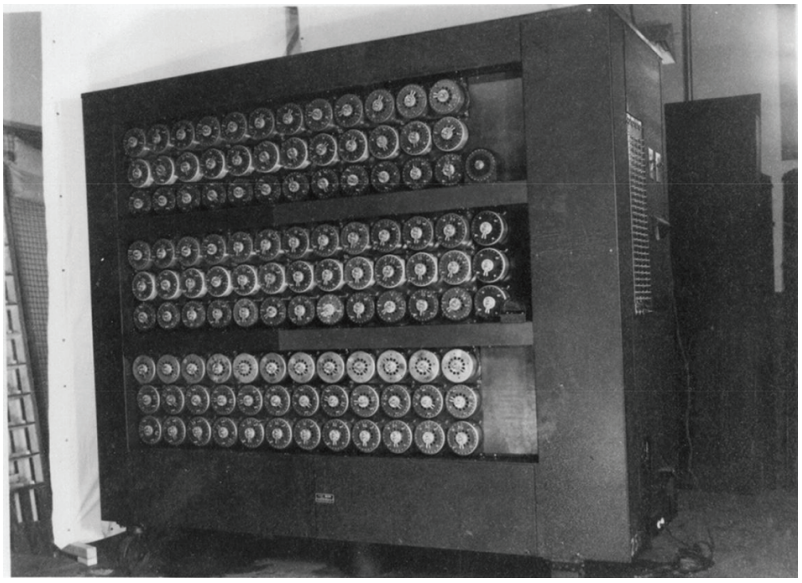

Figure 13. Bombe.

There are 36 rotors, each acting as a rotor in an Enigma machine. Each group of rotors acts as an Enigma machine. The British entered the cribs into the Bombe and it would brute-force it based on the input. If it met a possible solution, it would stop and the result would be examined.

V. Explanation of the Enigma simulator

The emulator developed by python can perform the same task as an enigma machine, but to improve the overall ability and User-friendliness of the stimulator, it also has additional functionalities to alter its rotors. In this section, the program will be broken down into sections and analyzed separately. These sections include its basic methods, rotor classes, encryption algorithm, and additional functions.

A.Fundamental functions of the simulator

```
#rearrange all letter in string A-Z to create a rotor
def rotor_creating():
    string1="ABCDEFGHIJKLMNOPQRSTUVWXYZ" #default string
    string2=""  #new string
    while len(string2)<26: #repeat it 26 times
        string2+=random_pick(string1)  #on new string add a random letter from default string
        string1=string1.replace(string2[len(string2)-1],"")  #remove the added char from default string
    return string2

def reflector_creating():
    string1="ABCDEFGHIJKLMNOPQRSTUVWXYZ"   #default string
    string2=""     #new string
    while len(string2)<13:     #repeat 13 times
        string2+=random_pick(string1)     #on new string add a random letter from default string
        string1=string1.replace(string2[len(string2)-1],"")    #remove the added char from default string
    temp_string=""   #final string
    temp_list = list()    #temp for first half
    temp_list2 = list()   #temp for second half
    for i in range(0,len(string2)):    #put all char in the new string into the two list, and two list are same at this time
        temp_list.append(string2[i])
        temp_list2.append(string2[i])
    random.shuffle(temp_list)     #shuffle two list, and now they are different orders
    random.shuffle(temp_list2)
    for i in temp_list:      #copy the item in the list1 to fianl string
        temp_string+=i
    for i in temp_list2:     #copy the item in the list2 to fianl string
        temp_string+=i
    return temp_string
```

Figure 14(a). Fractions of codes for the simulator

The simulator creates randomized rotor strings, which are randomized strings that contain all letters from A to Z. A random_pick helper function is created to model the random selection process. With the program importing the random module at the start, the random_pick takes a string as the parameter. It first chooses a random number between zero to the length of the string minus one inclusive, then picks and returns the character from the string at that index.

The helper function provides the precondition for the implementation of the rotor_creating algorithm. It has a default string containing letters from A to Z and an empty string. While the empty string's length is smaller than 26, a random letter from the first string is picked with the helper function. Then, that letter is removed from the first string and added to the end of the second string. Each time the first string will drop a letter while the second string gains a letter. In the end, the first string will be empty, and string two contains all letters from A to Z but in random order. At this time, the second string is returned as the rotor string.

Like the rotor_creating function, the reflector_creating procedure goes through the same operation as the rotor_creating function, but it only creates a new string with 13 random letters from A to Z. Then, it utilizes string temp as the final string and lists temp1 and temp2 as the two parts of the reflector. Temp1 and temp2 are set to the same characters as the previous 13 letter string. Then, both lists shuffle with the shuffle method in the Random module. Temp string combines the two lists characters and is returned as the final reflector.

B.Components of the rotor functions

```
class rotor3:
    def __init__(self,original_string,switched_string):     #first column string, second column string, name of the rotor
        self.original_string=original_string
        self.switched_string=switched_string

    def encrypt(self,text):     #forward encryption
        new_text=""     #new string to keep track
        for i in range(0,len(text)):
            if text[i]==" ":
                new_text+=" "
                continue
            switch_num = self.switched_string.find(text[i])   #find the input char in switch string
            temp_text = reflector[switch_num]     #find the corresponding char in reflector
            if switch_num>=13:          #determine which way to reflect
                temp_reflector = reflector[0:13]
            if switch_num<13:
                temp_reflector = reflector[13:26]
            reflected_num = temp_reflector.find(temp_text)     #get the number of reflection
            if switch_num<13:
                reflected_num+=13
            new_text+=rotor_3.switched_string[reflected_num]     #add the corresponding char in rotor3 switched string to new text
        return new_text

    def backward_encrypt(self,text):    #backward encryption
        new_text=""     #new string to keep track
        for i in range(0,len(text)):
            if text[i]==" ":
                new_text+=" "
                continue
            switch_num = self.original_string.find(text[i])     #find the input char in original string
            new_text+=rotor_2.switched_string[switch_num]       #add coresponding string in rotor2 switched string to new text
        return new_text
```

Figure 14(b). Fractions of codes for the simulator

Each rotor class has two attributes: original and switched string, representing the rotor's alphabet tire position before and after passing the rotor. Before moving on to the rotor object methods, the emulator constructs the shift_front and shift_back function that takes a rotor object as its parameter. With string slicing, the shift_front algorithm places the first letter of the rotor's original and switched string to the end of itself, while the shift_back function moves the last letter of the original and switched string to the start of the string. Each rotor has a encrypt and a backward-encrypt method. Despite having the same functionalities, taking a text string as its parameter, and having a variable new_text to keep track of the ciphertext, the method for each rotor is different. For all rotor's front and backward encryption, they all have one common detection algorithm: when it detects white space in the text, it adds the whitespace to its new_text and jumps its current processing iteration.

For rotor1's encrypt method, for each character in the text string, if it is not a letter, it calls shift_front to change its position before exiting the loop. Otherwise, the rotor first shift_front itself, then in its original string, finds the location of the text character in the default A to Z string, and uses switched_char to record the letter at the same position in its original_string. The program finds the letter in the original_string of the second rotor at the same position as the switched_char's location in rotor1's switched string. Rotor2's encrypt function starts with a loop like rotor1's encrypt method. If it passes whitespace detection, it finds the position of the text in rotor2's switched_string, then records the corresponding letter in rotor3's original_string in new_text. Unlike Rotor1 and Rotor2, Rotor3's method involves the reflector, making it extra complicated. It will first find the new_text in its switched_string, then find the corresponding index in the reflector. If the location is the first part of the reflector, it will search the second part of the reflector, vise versa. In the end, it will relocate the corresponding letter in rotor3's switched_string and return it.

Despite backward_encrypt may sound the same as the encryption method, it is entirely different. Rotor1's strings reset before rotor3's backward_encryption by using shift_back. All the shift_back resetting is done in the encryption function, which will be covered later. Rotor3's backward_encrypt is first called, which first locates the new_text in its original_string, then finds the corresponding text in rotor2's switched_string, then returns the letter. Then, for rotor2, its backward_encrypt method will call shift_front each time to set up the strings in rotor1. It then finds the text in its original_string, corresponds to rotor1's switched string, and returns the correspondence. After rotor2's backward_encryption, rotor1's strings are greeted by shift_back again. Rotor1's backward_encrypt is

very similar to rotor2's. It first calls shift_front, finds the new_text in rotor1's original_string, then returns the corresponding letter in the default A - Z string.

C.Initialization and encryption method

```
def enigma_encrypt(text):      #process of enigma encrypting
    a = text.upper()      #change text to all upper case
    b = rotor_1.encrypt(a)      #go through rotor1
    c = rotor_2.encrypt(b)      #go through rotor2
    d = rotor_3.encrypt(c)      #go through rotor3 and reflector

    for i in range(0,len(a)):   #reset rotor1 from its shift for furthur uses, correspond to backward encrypt of rotor 2
        shift_back(rotor_1)

    e = rotor_3.backward_encrypt(d)      #go through rotor3 second time
    f = rotor_2.backward_encrypt(e)      #go through rotor2 second time

    for i in range(0,len(a)):   #reset rotor1 from its shift for furthur uses, correspond to backward encrypt of rotor 2
        shift_back(rotor_1)

    g = rotor_1.backward_encrypt(f)     #go through rotor1 second time, get final output

    for i in range(0,len(a)):       #final reset of the rotor1 after use
        shift_back(rotor_1)

    return g

def enigma_display():
    if check_valid(plain_text.get().upper())==True:
        cipher_text=enigma_encrypt(plain_text.get())
        if enigma_encrypt(cipher_text)==plain_text.get().upper():
            cipher_text_label.configure(text = enigma_encrypt(plain_text.get().upper()))
        else:
            change_rotor()
    else:
        cipher_text_label.configure(text = "Invalid input, please re-enter")
```

Figure 14(c). Fractions of codes for the simulator

After implementing all the encryption functions in the rotor classes, the actual initialization and encryption method becomes very easy to write. Before everything, the rotors and reflector must be initialized, and for each of them, we set their original_string and switched_string and reflector by calling reflector_creating and rotor_creating function, which we implemented earlier.

To make the GUI portion, Tkinter is imported, and a new window is created. We create Tk labels for each component of the displayed window, including each rotor's title and grid for their strings. Then the input box and encryption button are placed. Meanwhile, the check_valid function is made to check on input. It takes three parameters: text, length (optional, default 0), and rotor. (optional, default false) It utilizes a temporary variable check, which is initialized as True. The function first checks the length of the text. If it doesn't match the length parameter while the length parameter is used, it will return false. Otherwise, if the length matches, it will first check if the rotor parameter is used. If it is used, the input must contain all letters in A to Z exactly once, or the program returns false. At last, it checks if all letters and only returns True if all characters are spaces or letters.

In addition, the set_rotor, shift_rotor, and change_rotor buttons are added, and these algorithms will be explained in the next section. The enimga_encrypt method first takes a text as its parameter and changes it to all uppercase letters. It goes through rotor1 to rotor3's encryption and the backward encryption with the opposite order. On its way to encrypt backward, rotor1 is reset before rotor3, rotor1's backward encryption, and after rotor1's encryption. In the end, it returns the final ciphertext. The ciphertext and input will be checked and displayed by the enigma_display function. It first sees if the input string is valid with the check_valid method. If it is invalid, it will give out a warning. Then it checks if the original text is equal to the cipher_test encrypted again. If they are the same, then the rotor set works, then the ciphertext is displayed. Otherwise, it will change the rotor with the change_rotor function.

D.Other functions and future improvement

```
def change_rotor():       #change rotor

    initiate_rotor()      #re-initiate rotor

    rotor_1_original_string_label.configure(text=rotor_1.original_string)      #change displays
    rotor_2_original_string_label.configure(text=rotor_2.original_string)
    rotor_3_original_string_label.configure(text=rotor_3.original_string)

    rotor_1_switched_string_label.configure(text=rotor_1.switched_string)
    rotor_2_switched_string_label.configure(text=rotor_2.switched_string)
    rotor_3_switched_string_label.configure(text=rotor_3.switched_string)

def shift_rotor():
    word = shift_word.get().upper()
    if check_valid(word,3):
        while rotor_1.switched_string[0]!=word[0]:   #shift rotor 1
            shift_front(rotor_1)
        while rotor_2.switched_string[0]!=word[1]:   #shift rotor 2
            shift_front(rotor_2)
        while rotor_3.switched_string[0]!=word[2]:   #shift rotor 3
            shift_front(rotor_3)

        rotor_1_original_string_label.configure(text=rotor_1.original_string)      #change displays
        rotor_2_original_string_label.configure(text=rotor_2.original_string)
        rotor_3_original_string_label.configure(text=rotor_3.original_string)

        rotor_1_switched_string_label.configure(text=rotor_1.switched_string)
        rotor_2_switched_string_label.configure(text=rotor_2.switched_string)
        rotor_3_switched_string_label.configure(text=rotor_3.switched_string)
    else:
        cipher_text_label.configure(text = "Invalid shift word, please re-enter")
```

Figure 14(d). Fractions of codes for the simulator

Other than the main encryption functions, different additional methods are built to boost the overall capability of the simulator. First, the change_rotor function, which ranges all the rotors but not the reflector, calls the initiate_rotor function to reset all the rotors, then changes the displayed rotor strings with the configure method adopted from Tkinter. The shift_rotor algorithm is implemented to set the first letters of the switch strings to a specific three-letter word. It first takes input from its corresponding input box, then calls the check_valid function with length parameter set to three to check if the input string is valid. If valid, each switch_string will be shifted using shift_front until the first letter of the switch_string matches the corresponding letter in the three-letter word. The rotor display labels will be changed after the rotors. If the input is invalid, then “invalid input” will be displayed to warn the user. The set_rotor function will also allow the user to set each of the rotor strings themselves. When its button is clicked, set_rotor will pop out a new window with six input boxes, each representing one of the rotor strings. With inputs ready, the user can click the confirm button to set the rotors, which calls the set_rotor_function to check the input. The set_rotor_function calls check_valid function on each of the six input strings, with length parameter 26 and rotor parameter as True, which examine if the strings are valid rotor candidates. If all inputs are correct, it changes the rotors, rotor display sections and shuts down the new window. Otherwise, an “invalid input” message will be displayed on the screen. The current stimulator can be further refined to reduce the line of codes, calculation amount, and running time. Some repeated code may be combined, or rewritten as a new function. Some components may be revised to reduce redundancy, and adding more detail to the documentation may help inspectors break down the code better.

VI.Improvement of the Enigma machine

The encryption scheme of Nazi Germany had been broken for times even before Alan Turing's Bombe. However, those were the flaws of the scheme instead of the machine per se. To improve the machine, the design deficiency has to be found out and solved.

A. A potential problems

There were inspirations in the previous breaking: Alan Turing's Cribs method and Bombe took advantage of the feature that input letters never output the same letters. This feature is the fundamental feature of the Enigma machine that allowed German soldiers to encrypt any message on both sides. To improve it we need to know how it happens inside the machine.

1)How the flaw happens. The crucial component in the machine, called the 'reflector', is the reason why it happens. There are wires inside the reflector that connect different pairs of contacts on one side of the wheel. When an electric current goes into the rotor through any contact of the wheel, it will always come out from the other contact of the same pair and go back to the rotors. The electric currents never make their way back to the same contacts so that the input doesn't yield the same output letter as the input letter.

B. The Improved methods

A modification for the reflector was done by the search group as follows.

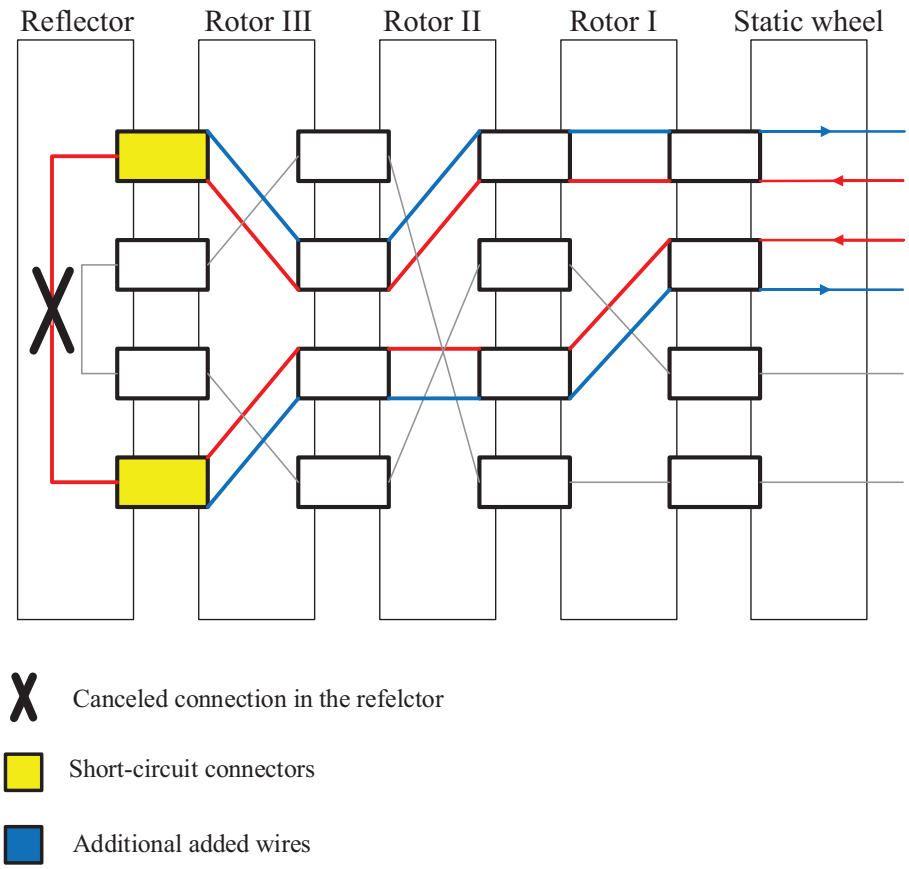

Figure 15. the Improved Rotor Circuit.

The machine has to yield the same output letter as the input letter to be improved. Modifications have been done to the reflector and the electric circuit of the machine to solve the underlying issue.

The added wires provide paths for electric currents to flow back and make their way to the corresponding lamps. And of course, there are similar modifications done in other parts of the machine, which means the keyboard, the lamps, and the plugboard.

To make the electrons sometimes revert to another wire of the same pair, a connection is canceled in the reflector. And the pair of detached contacts are made larger to short circuit the pair of wires connected to them. Now it allows the machine to sometimes yields the input letter itself.

VII. Conclusion

In the future, a series of meaningful work can be conducted subsequently. Based on the current work, we can further develop the machine so that it can be used more widely. For instance, the Enigma machine combined with modern encryption algorithms can result in new methods of encrypting and decrypting information, and we can develop an app designed to make the public able to encrypt and decrypt their messages conveniently whenever and wherever they want to. Further research combined with other subjects can be conducted for a higher level of security such as a quantum optical Enigma machine, which is a combination of Optics, Quantum, and Cryptography. It is capable of functioning in high levels of loss while the previous tolerance of loss is at most 50%.

In the future, a series of meaningful work can be conducted subsequently. Based on the current work, we can further develop the machine so that it can be used more widely. For instance, the Enigma machine combined with modern encryption algorithms can result in new methods of encrypting and decrypting information, and we can develop an app designed to make the public able to encrypt and decrypt their messages conveniently whenever and wherever they want to. Further research combined with other subjects can be conducted for a higher level of security such as a quantum optical Enigma machine, which is a combination of Optics, Quantum, and Cryptography. It is capable of being operated normally under severe noise and loss.

References

[1] Cryptomuseum. (2009) Enigma I. https://www.cryptomuseum.com/crypto/enigma/i/index.htm.

[2] Miller, A.R. (2010) THE CRYPTOGRAPHIC MATHEMATICS OF ENIGMA. CRYPTOLOGIA., 19: 65-80.

[3] en-academics. (2010) Enigma machine. https://en-academic.com/dic.nsf/enwiki/5372#Rotors.

[4] Welchman, W.G. (1982) The Hut Six Story - Breaking the Enigma Code. Penguin Books Ltd, London.

[5] Bauer, C.P. (2013) Secret History - The Story of Cryptology. CRC Press, Boca Raton.

[6] Chiffriermaschine. (2020) Enigma Stecker Cable. https://www.chiffriermaschine.com/enigma/enigma-stecker-cable.

[7] Haran, B.J. (2013) Flaw in the Enigma Code - Numberphile. https://www.youtube.com/watch?v=V4V2bpZlqx8.

[8] David P. Mowry. (2014) German Cipher Machines of World War II. National Security Agency, Fort Meade.

[9] Alexander, C. (1945) Cryptographic History of Work on the German Naval Enigma. The National Archives, Richmond.

[10] Plimmer, B. (1998) Machines invented for WW II code breaking. ACM SIGCSE Bulletin., 30: 37–40, ISSN: 0097-8418.

[11] Deniz, E., Berna, O. (2015) Implementation of Enigma machine using verilog on an FPGA. In: 9th International Conference on Electrical and Electronics Engineering (ELECO). Bursa. pp. 945–948.

[12] Hinsley, F.H., Stripp, A. (1993) Codebreakers: The Inside Story of Bletchley Park. Oxford University Press, Oxford.

[13] Marks, P. (2018) Mr. Twinn's bombes.CRYPTOLOGIA., 42: 1-80.