The Electronic Journal of Mathematics and Technology, Volume 19, Number 2, ISSN 1933-2823

Cryptanalysis of the Enigma Machine: The Bombe and Beyond

Adam S. Downs
asdowns114@gmail.com

Neil P. Sigmon
npsigmon@radford.edu

Department of Mathematics and Statistics
Radford University
Radford, Virginia 24142
USA

Richard E. Klima
klimare@appstate.edu
Department of Mathematical Sciences
Appalachian State University
Boone, North Carolina 28608
USA

Abstract

The cryptanalysis of the Enigma machine has been recognized as one of the supreme achievements of the human intellect. One of the trickiest parts related to the plugboard, which contributed by far the largest factor to the total number of configurations of the machine. To determine the daily plugboard connections, Allied codebreakers used electromechanical devices called the bombe and the checking machine. After they found the plugboard connections though, they still had to discover additional settings, the difficulty of which varied depending on which branch of the German military had created the messages under attack. The process was significantly more difficult for messages created by the German Navy. In this paper, we will describe some of the procedures involved in recovering this additional information needed to fully cryptanalyze the Enigma. To assist in demonstrating these procedures, technology involving Maplets will be used.

1 Introduction

In 1918, German electrical engineer Arthur Scherbius applied for a patent for a mechanical cipher machine. This machine, later marketed commercially under the name *Enigma*, was designed with

electric current passing through revolving wired wheels. Scherbius offered the machine to the German military. Only after learning that their World War I ciphers had routinely been broken did they adopt it, which they used as their primary resource for encrypted communications throughout World War II.

As we described in [3], Polish codebreakers led by Marian Rejewski cracked the Germans' first implementation of the Enigma. After modifications by German codemakers rendered the Polish techniques no longer viable, British mathematician Alan Turing identified weaknesses in the encryption process using patterns generated by *cribs*, which are small parts of plaintext corresponding to known parts of a ciphertext, made easier for Allied codebreakers to find through the frequent mistaken use of salutations, titles, and addresses by Enigma operators. When a device designed by Turing called the *bombe* became operational, Allied codebreakers again began reading German messages. For messages created by the German Army and Air Force, once a single encrypted message had been broken, every message between any two operators on the same day could be broken due to a deficiency in how operators transmitted their rotor starting positions. German Navy operators used a more secure procedure though for transmitting their positions, which made the codebreaking process more difficult.

In this paper, we will examine some of the challenges and illustrate some of the techniques used by Allied codebreakers to discover daily Enigma machine settings and break Enigma messages sent by operators from all branches of the German military. We will begin with a brief review of the components of an Enigma and the basics of how the bombe was used in the initial cryptanalysis.

2 Components of Enigma

We begin by reviewing the components of an Enigma machine and the challenges it presented to Allied codebreakers. We give more detailed descriptions of these components and challenges in [4]. Figure 1 shows an image of an Enigma with several important components labeled.

Figure 1: Enigma cipher machine.

To encrypt or decrypt a letter, the key labeled with the letter on the keyboard was pressed, which launched electric current representing the letter into the machine. It traveled first through the plugboard, where it would change to another letter if sockets labeled with the letter were connected by a cable to sockets labeled with the other. It then passed through three rotors, in each of which it could change to another letter, and then through a reflector, where it definitely changed to another letter. It then went back through the rotors in the opposite direction, and then back through the plugboard, in each of which it could again change to another letter. Finally, it went to the lampboard, where it lit a lamp labeled with the encrypted or decrypted letter. We describe this process in more detail in [2].

For the plugboard, the Germans settled on using 10 cables to connect 20 letters in pairs. Using a formula that we describe and justify in [4], it can be shown that there are 150,738,274,937,250 ways to choose 20 letters and connect these letters in pairs using 10 cables at the plugboard. Around each of the three rotors in an Enigma, a ring was etched with the 26 letters (or numbers representing the letters) alphabetically clockwise when viewed from the right. Above each rotor slot, a small window was cut to make the letter (or number) at a particular location on the ring visible, called the *window letter*. While each rotor could be rotated into any of 26 different positions inside the machine, just the ring alone could also be rotated while the central part of the rotor was held fixed. A rotation of the entire rotor versus just the ring was distinguished using a number called the *ring setting*. The ring setting essentially indicates a rotation of just the ring, while the window letter indicates a combined rotation of the central part of the rotor with the ring attached. The window letter and the ring setting each contribute a factor of $26^3 = 17{,}576$ to the total number of machine configurations.

The window letter and the ring setting must be considered separately to account for all of the variability in the rotors because the rotors rotated during encryption and decryption, and this rotation was only influenced by the ring, not the central part of the rotor. Encryption and decryption was done one letter at a time, and each time an input letter was pressed on the keyboard, the rightmost rotor (before the current reached it) would rotate one position counterclockwise when viewed from the right. Notches cut into the ring around each rotor then influenced the rotation of a rotor to its left. The German Army and Air Force used rotors labeled **I**–**V**, into each of which one notch was cut. The German Navy also used additional rotors labeled **VI**–**VIII**, into each of which two notches were cut. Since each notch was on the ring, its position in the rotor slot at any time could be identified by the window letter. For each notch, there was one position in the rotor slot, identified by a window letter called the *notch letter*, for which if the rotor rotated one position counterclockwise, the notch would cause a rotor to its left to also rotate one position counterclockwise. The notch letters for the rotors were Q for **I**, E for **II**, V for **III**, J for **IV**, Z for **V**, and M and Z for **VI**, **VII**, and **VIII**.

Since the German Army and Air Force chose from among five rotors for three rotor slots, there were $5 \cdot 4 \cdot 3 = 60$ ways for Army or Air Force operators to arrange rotors in the machine. Since the German Navy chose from among eight rotors though, there were $8 \cdot 7 \cdot 6 = 336$ ways for Navy operators to arrange rotors in the machine. Although rotors could only be situated in the machine with each side facing in a particular direction, current passed through them in both directions, from right to left before the reflector, and then from left to right after the reflector. Like the rotors, reflectors had electrical contacts representing the 26 letters alphabetically clockwise when viewed from the right, though only on one side, since current entered and exited reflectors on the same side. Unlike the rotors (and the

plugboard), letters in reflectors were always fully connected in 13 pairs. Having the reflector in the middle of the symmetric sequence

$$\text{plugboard} \rightarrow \text{rotors (right to left)} \rightarrow \text{reflector} \rightarrow \text{rotors (left to right)} \rightarrow \text{plugboard}$$

also ensured that the machine configuration for encrypting and decrypting the same message was identical, since the decryption of a ciphertext letter was obtained by simply reversing the path through the machine that the corresponding plaintext letter had taken when it was encrypted.

Reflectors labeled B and C with two different electrical wirings were produced for the German Army, Air Force, and Navy. The reflector in use was always placed in the machine in only one way and did not rotate. As such, the number of ways in which a reflector could be chosen was 2. Combining the factors resulting from the plugboard, window letters, ring settings, rotor arrangements, and reflector, the total number of different machine configurations for an Enigma used by the German Army and Air Force was 5.59×10^{24}. In an Enigma used by the German Navy, for which the number of rotor arrangements was 336 rather than 60, the total number of machine configurations was 3.13×10^{25}.

3 Overcoming the Plugboard Factor

We next describe how Allied codebreakers overcame the plugboard factor and the logic behind the Turing bombe. We give a more detailed description of this in [4].

Allied codebreakers discovered that *cribs*, which are small parts of plaintext corresponding to known parts of a ciphertext, could be used to eliminate the plugboard factor for the Enigma. To demonstrate this, suppose the crib `FOLLOW ORDERS TO` was known to encrypt to the part of the ciphertext message `NUENTZERLOHHBTDSHLHIY` that is underlined. In our analysis of this alignment, we will start by labeling the crib and ciphertext letters with position numbers as follows.

Position:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Crib:	F	O	L	L	O	W	O	R	D	E	R	S	T	O
Cipher:	E	N	T	Z	E	R	L	O	H	H	B	T	D	S

Since an Enigma used identical settings for encryption and decryption, for an Enigma in position 1 with plaintext letter `F` encrypted as ciphertext letter `E`, it would also be true that plaintext letter `E` would encrypt as ciphertext letter `F`. Similarly, for the machine in position 2 with plaintext letter `O` encrypted as ciphertext letter `N`, plaintext letter `N` would encrypt as ciphertext letter `O`.

We will now demonstrate a Maplet[1] entitled **Turing Welchman Bombe Menu**, which was written by the authors, and designed to assist in finding plugboard pairs. The source code for this and all of the Maplets demonstrated in this paper, as well as directly usable versions of them, can be downloaded using the links in Section 9. The code is unique to Maple, but could easily be altered for any language. In this Maplet, we enter a crib along with its corresponding ciphertext. By clicking the **Plot Menu** button, Figure 2 shows the crib/ciphertext pairs in what the Allied codebreakers called a *menu*.

[1] A Maplet is like an applet, but requires the engine of the CAS Maple, and is written using Maple functions and syntax.

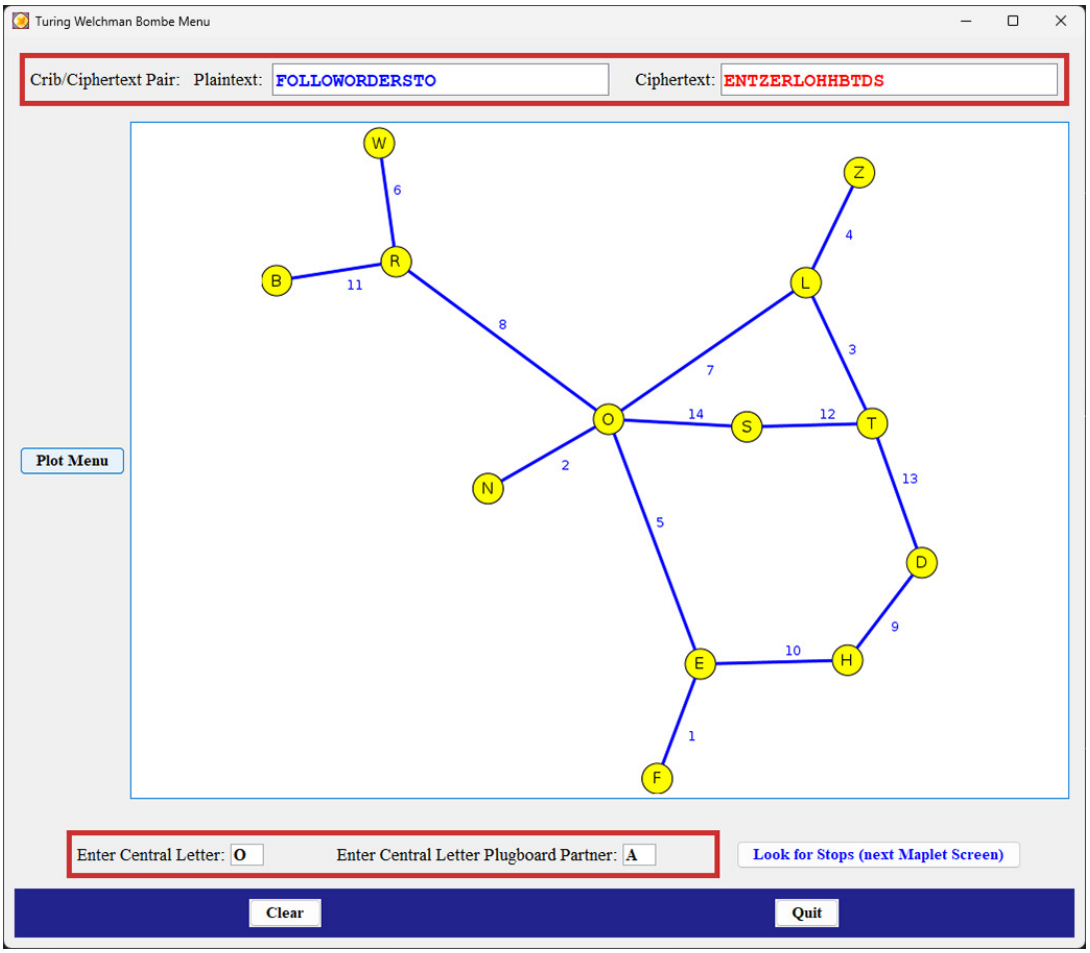

Figure 2: Finding a crib/ciphertext menu.

In the Turing bombe, the plugboard, rotors, and reflector were connected in a "double-ended" fashion. The effect of going through an ordinary Enigma and a double-ended version is the same, except that current in the double-ended travels in the same direction. In the bombe, Enigma rotors were emulated by cylindrical disks called *drums*. We will express the combination of a letter passing through the rotors (drums) in both directions and the reflector as a *double scrambler*. On both sides of the double scrambler was a cable connection containing 26 wires, with each wire representing a single letter. On the bombe, three drums emulating the three rotors in the double scrambler were mounted vertically on a sequence of shafts, with the top drum emulating the rightmost rotor of the Enigma, the middle drum emulating the middle rotor, and the bottom drum emulating the leftmost rotor.

Turing's attack on the plugboard began with the choice of a menu letter, normally one with a large number of links connected to it, called the *central letter*. From the menu in Figure 2, a natural choice for the central letter would be O. Once a central letter was selected, a possible plugboard partner for it was chosen. Then, the locations in the menu where closed loops occurred were noted. In the menu in Figure 2, three sequences of links form closed loops. Starting with the possible plugboard partner of the central letter, the plugboard partners of the letters in the loops were found. From the letters in closed menu loops and their plugboard partners, Turing made a hypothesis about the

reflector, rotor order, and window letters that were used when a given ciphertext was formed. Recall that when a ciphertext was formed, although the rotors rotated within the machine during encryption, the plugboard connections never changed during encryption. This meant that whatever the plugboard partner for a menu letter was at the start of a loop, it had to be the same at the end of the loop.

During the search for machine settings that gave the plugboard pairs, the window letters and ring settings found were usually not the ones that had been used in the actual encryption of the message. Instead, the window letters and ring settings found resulted in the same rotor core starting positions, or *rotor offsets*, that had been used in the actual encryption of the message. To understand this idea better, consider an encryption with initial left-to-right window letters NCS (which correspond to position numbers 14, 3, 19 in the alphabet) and corresponding ring settings 4, 26, 13. The rotor offsets for this encryption would then be 10, 3, 6, since $14-4=10$, and $3-26=-23=3 \bmod 26$, and $19-13=6$. However, if we did the same encryption with these rotors and initial window letters ZZZ (corresponding to 26, 26, 26) and corresponding ring settings 16, 23, 20, the rotor offsets would again be 10, 3, 6, since $26-16=10$, and $26-23=3$, and $26-20=6$. Then, assuming the same reflector and rotor order were used, and that there were no rotations of the middle or left rotors in either scenario, the encryption using these rotors with initial window letters NCS and corresponding ring settings 4, 26, 13 would be the same as with initial window letters ZZZ and corresponding ring settings 16, 23, 20. To test for the correct rotor core starting positions, a setting for the initial window letters, in many cases ZZZ, was chosen, and the ring settings were found by a brute force attack from among 17,576 possibilities, until the correct rotor core starting positions were found. To find a plugboard partner for the central letter that was logically consistent, a combination of a reflector (either B or C), a rotor order (of which there were 60 possibilities for Army and Air Force messages, or 336 for Navy messages), and a ring setting (of which there were 17,576 possibilities) was tested, which eliminated having to test the 17,576 possibilities for the window letters.

Of course, each of the 17,576 possibilities for ring settings had to be considered for each of the 120 possible combinations of a reflector and rotor arrangement of the Army and Air Force Enigma and 672 for the Navy Enigma. This gave a total of $17{,}576 \cdot 120 = 2{,}109{,}120$ configurations for the Army and Air Force Enigma and $17{,}576 \cdot 672 = 11{,}811{,}072$ for the Navy Enigma that could require some level of testing. Although these numbers are significant, the Allied codebreakers found them manageable.

To demonstrate this, consider an Enigma configuration with reflector **C**, rotor order **I**, **V**, **III**, initial window letters ZZZ, and ring settings 16, 23, 18, to be tested on the crib/ciphertext alignment given at the start of this section. With the fact that pressing a key on an Enigma keyboard caused the rightmost rotor to rotate one position before the encryption or decryption of the letter, and assuming no rotation of the middle or leftmost rotors, we can notate this alignment with window letters as follows.

Position:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Window:	ZZA	ZZB	ZZC	ZZD	ZZE	ZZF	ZZG	ZZH	ZZI	ZZJ	ZZK	ZZL	ZZM	ZZN
Crib:	F	O	L	L	O	W	O	R	D	E	R	S	T	O
Cipher:	E	N	T	Z	E	R	L	O	H	H	B	T	D	S

Consider the loop $\mathtt{O} \to \mathtt{S} \to \mathtt{T} \to \mathtt{L} \to \mathtt{O}$ in Figure 2, and suppose we choose A as the plugboard partner of O. Using the **Double Scrambler Calculator** Maplet, which was written by the authors, we

can find the double scrambler output and thus plugboard partners for all of the letters in the loop. In this Maplet, the user first enters the reflector, ring settings, and rotor order. Then, to find the plugboard partner for the letter S linked to O, the user enters the partner A chosen for O and the window letters ZNN at position 14 used to encrypt O to S. By clicking the **Find Output Double Scrambler Letter** button, the plugboard partner G of S in the menu is produced. Figure 3 shows the result.

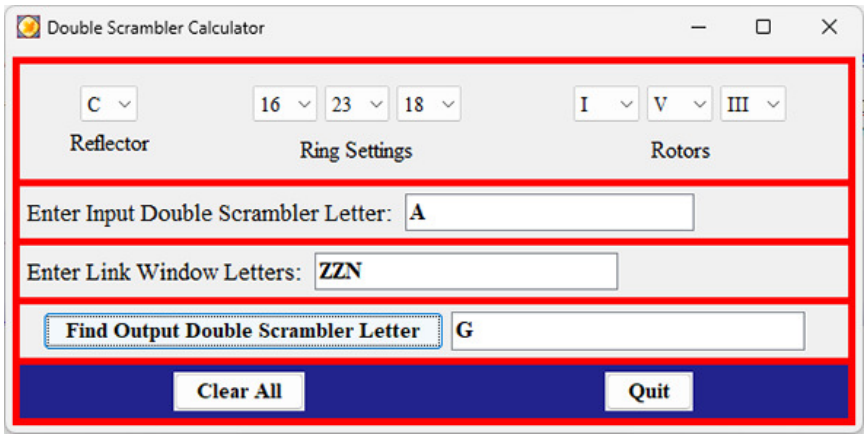

Figure 3: Finding a double scrambler output.

The process can be continued to find the plugboard partners for all of the letters in the loop. In summary of this process, this machine configuration with plugboard partner A of O at the start of the loop would result in the following plugboard partners for all of the letters in the loop.

Drum Setting:	ZZN	ZZL	ZZC	ZZG
Menu Letter:	O ⟶ S	⟶ T	⟶ L	⟶ O
Plug Partner:	A ⟶ G	⟶ P	⟶ W	⟶ Q

However, this gives different plugboard partners of the central letter at the start and end of the loop, a logical inconsistency which shows that the assumed reflector, rotor order, initial window letters, ring settings, and plugboard partner of the central letter cannot all be correct. Continuing with the same reflector, rotor order, initial window letters, and ring settings, choosing S as the plugboard partner of O results in plugboard partners of the letters in the loop in the sequence S → O → N → I → S. With the same plugboard partner of the central letter at the start and end, the settings could all be correct.

Once a chosen letter produced an inconsistency though, instead of searching the rest of the alphabet to find a possible letter that was logically consistent for the given loop, Turing found it more efficient to let the plugboard letter output from the loop for the central letter serve as the next input into the loop, letting this letter proceed through the loop, and repeating this task until a cycle was produced. Each non-plugboard partner was recorded to eliminate possible plugboard partners for the central letter.

For example, recall we first tested the menu loop O → S → T → L → O with initial choice A for the plugboard partner of the central letter O. This resulted in plugboard partners for the menu letters in the loop in the sequence A → Q → P → W → Q, which gave an inconsistency. Using Turing's scheme, next we would test the same loop with Q as the plugboard partner of O. This would result in plugboard

partners for the menu letters in the loop in the sequence Q → V → J → M → L, which again gives an inconsistency. Next we would test the same loop but with L as the plugboard partner of O. Turing's scheme continued in this manner, testing the same loop repeatedly until eventually cycling back to the original choice A for the plugboard partner of O. In this example, this would require testing the loop ten more times, and the choices for the plugboard partner of O would be the following in order.

A ... Q ... L ... H ... Y ... O ... K ... J ... E ... W ... M ... N ... A

For convenience, these non-plugboard partners can be expressed as the *cycle* (AQLHYOKJEWMN). This does not generate all of the letters that are not plugboard partners though. One method to find other non-plugboard partners is to pick another letter not in the cycle (AQLHYOKJEWMN) and see if it generates another cycle with an inconsistency. A more efficient way for menus that have more than one loop though is to generate cycles using other loops and use the loops together to generate more non-plugboard partners. Note that the menu in Figure 2 contains three loops that can be expressed by O → S → T → L → O, and O → S → T → D → H → E → O, and O → L → T → D → H → E → O. By using the **Turing Welchman Bombe Menu** Maplet, we can determine the cycles generated by these loops. After entering a potential plugboard partner A of the central letter O and clicking the **Look for Stops (next Maplet Screen)** button, another Maplet screen appears. In this screen, the reflector **C** and ring settings 16, 23, 18 to be tested are selected. The rotor order **I**, **V**, **III** is then selected in a bottom to top format corresponding to how drums were placed on the Turing bombe representing the rotors in the Enigma machine. Then, assuming initial window letters ZZZ, the cycles produced by the three loops are generated by clicking the **Search for Stops** button. Figure 4 shows the result.

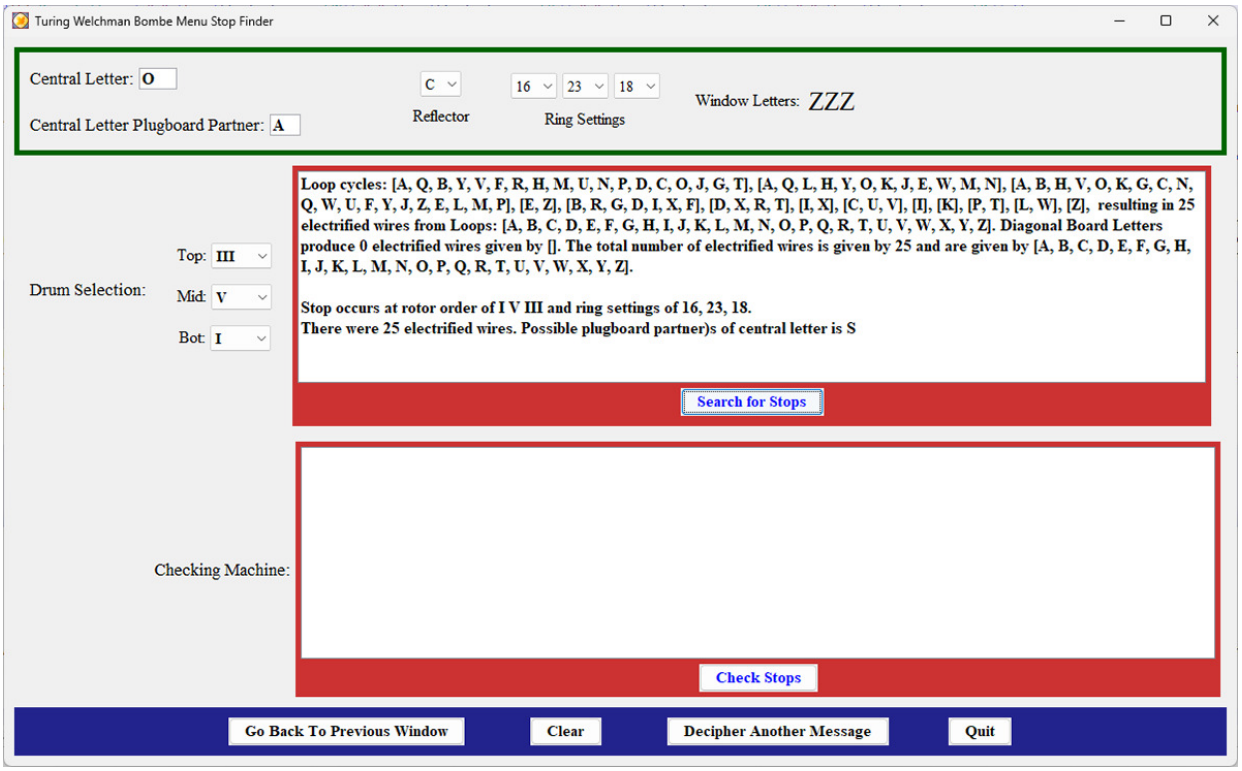

Figure 4: Generating loop cycles.

The Maplet computes the cycles generated by the three loops and takes their union. Any letter not included in the union, which here is only S, is a possible plugboard partner of the central letter. This is only a candidate plugboard partner for the central letter O though. Normally in practice, 10 plugboard cables were used to switch 20 total letters in pairs. Thus, once the first potential plugboard partner of a letter was found, 9 other letters that had plugboard partners had to be found, plus the 6 other letters not swapped by a cable. A device known as the *checking machine* was designed to determine if the potential plugboard partner of the central letter was correct and look for the other plugboard partners. When a potential plugboard partner of the central letter was found, the settings were used for the selection of the drums, ring settings, and reflector on the checking machine. Using the plugboard partner of the central letter producing the stop, the drums would be moved to positions corresponding to the menu link positions required to transform the plugboard letter of the central letter into the plugboard letter of another menu letter. The process was continued for other links to find plugboard partners of all the menu letters. Since usually 10 cables were used, the goal was to find the 20 letters that were plugboard partners. Once they were found, the other 6 letters were assumed to have no plugboard partner. However, the results were only used if no letters with multiple plugboard partners in the menu occurred. If a logical inconsistency occurred, the bombe stop was assumed to be false and the results disregarded. The bombe was restarted to look for other settings that would decrypt the message. If the menu was logically consistent, the message was sent for decryption.

This process of finding other plugboard partners can be done using the **Turing Welchman Bombe Example** Maplet. Continuing the result with a potential partner S of the central letter O, the checking machine is implemented by clicking the **Check Stops** button. Figure 5 shows the result.

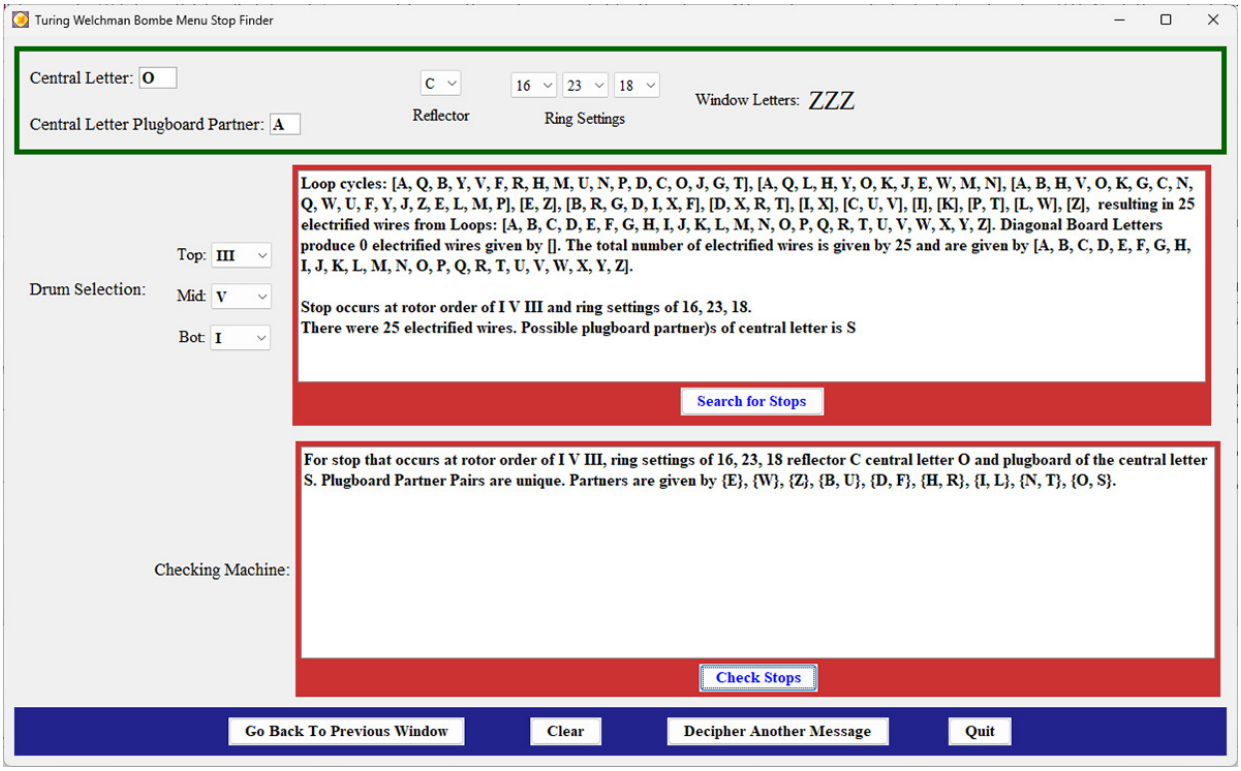

Figure 5: Running the checking machine.

These results give no inconsistencies anywhere in the menu, and indicate, if the Enigma configuration in this example is correct, then in addition to the central letter and its plugboard partner O/S, the letters N/T, R/H, B/U, L/I, and D/F would be plugboard pairs. They also show that the letters W, Z, and E would be unconnected in the plugboard. This does not give the full setup of the plugboard though, since four plugboard pairs would remain to be determined. However, by using the crib/ciphertext encryption plus decrypting other parts of ciphertext with the plugboard settings already discovered, the other plugboard assignments can be found. We describe this process in [4]. Using this process, the ten pairs of letters connected in the plugboard are O/S, N/T, R/H, B/U, L/I, D/F, G/A, Q/P, C/K, and J/Y, leaving W, Z, E, V, M, and X unconnected in the plugboard. This allows the full ciphertext chunk NUENTZERLOHHBTDSHLHIY to be decrypted as GO FOLLOW ORDERS TO QUICK.

Sometimes when a reflector, ring setting, and rotor order were tested, all 26 letters were eliminated as plugboard partners of the central letter. In other cases, a potential plugboard partner was found for the central letter but the checking machine found non-unique plugboard partners for other letters in the menu, resulting in a false stop. Also, a component added later to the Turing bombe called the *diagonal board* was useful in finding plugboard partners for menus without multiple loops. We describe these cases and the diagonal board in [2]. When successful though, this process allowed codebreakers to break a single encrypted message between two Enigma operators on any given day. Breaking all messages sent by any Enigma operators on that same day required more work though.

4 Army and Air Force Encryption

German Army and Air Force Enigma operators were equipped with a three-rotor Enigma, five different rotors labeled **I**–**V**, and a common monthly setting sheet. For each day of the month, the setting sheet would identify to the operator the machine setting, or *key*, for the day. Included in the daily key were the rotors to be installed in the machine, their prescribed order, the choice of reflector between B and C, the ring settings for the rotors, and the ten plugboard letter pairs. When an operator needed to encrypt a message, they set their machine up using the specified daily key. However, when placing a rotor in the machine, each operator would choose their own window letter for it. The window letters they chose for all three rotors, in order from left to right, were known as the *message setting*. Of course, the Enigma operator to whom they were sending the message needed to know the message setting as well. We will describe a procedure for providing this that was used by German Army and Air Force Enigma operators after May 1940. For the purpose of communicating their message setting, the sending operator would choose another three letter group, known as the *indicator setting*, and start by setting up their machine using the indicator setting rather than the message setting. They would then use this setup to encrypt the message setting. The sending operator would then change the window letters to match the message setting, and encrypt the message. They would then transmit the result to the receiving operator by sending the encrypted plaintext message following by a six-letter sequence consisting of the sending operator's indicator setting and encrypted message setting.

We will demonstrate this process using the Maplet **Encrypt/Decrypt Enigma Message**, which was written by the authors, and designed to encrypt and decrypt messages using an Enigma. Suppose the

daily key is rotor order **I**, **V**, **III**, reflector B, ring settings 19, 13, 25, and plugboard pairs O/S, N/T, R/H, B/U, L/I, D/F, G/A, Q/P, C/K, and J/Y. Suppose also that the sending operator chooses message setting TEX, and indicator setting RED. The sender would first set their machine up using the daily key with the indicator setting as the window letters, and encrypt the message setting. This would give output YXS. Suppose now that the sending operator wanted to encrypt TODAY IS THE DAY. They would then change the window letters in the machine to TEX, and encrypt the message. Figure 6 shows the result of using the Maplet to do this, giving the ciphertext HKMQWALFVHABJ.

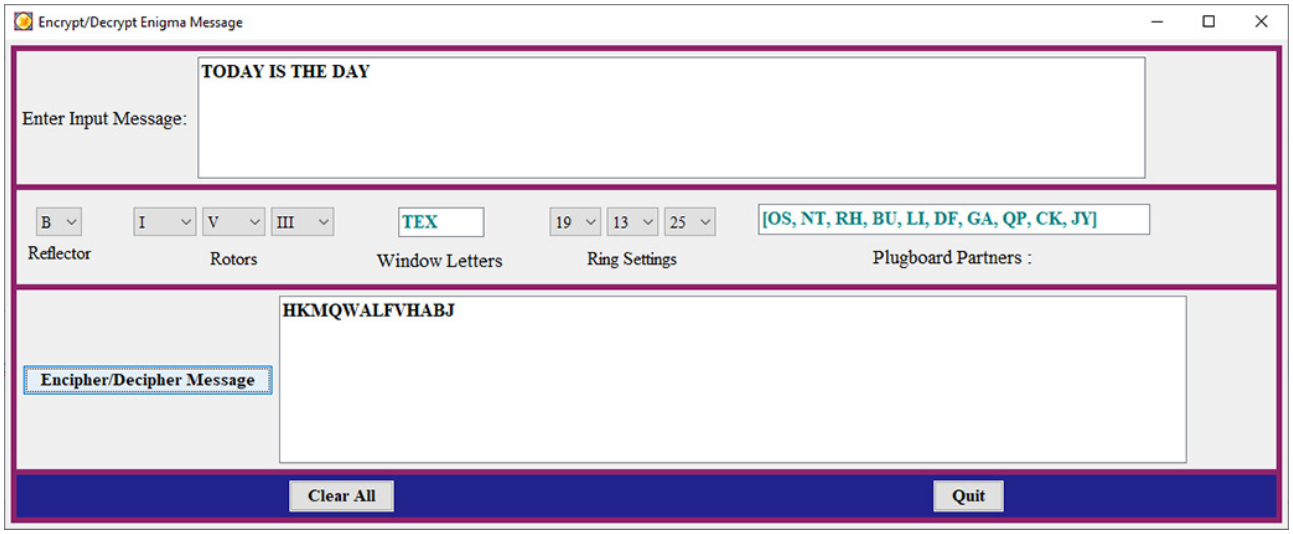

Figure 6: Encrypting a message.

Finally, they would transmit the ciphertext followed by the indicator setting and the encrypted message setting: HKMQWALFVHABJ REDYXS. To decrypt the message, the receiving operator would first set their machine up using the daily key with the indicator setting RED as the window letters, and decrypt the last three letters received. This would result in TEX. The receiving operator would then change the window letters in their machine to TEX, and use this setup to decrypt the message.

5 Army and Air Force Cryptanalysis

Turing's bombe enabled Allied codebreakers to decrypt single messages that had been encrypted by German Army and Air Force Enigma operators. However, there was not enough time to use the bombe to decrypt all messages sent between any two Army or Air Force operators. We will now consider how it was possible for codebreakers to quickly decrypt all messages sent between any two Army or Air Force operators on the same day after decrypting a single message using the bombe.

When an individual Enigma ciphertext was decrypted using the bombe, the process resulted in ring settings that gave rotor core starting positions based on the assumption that the last window letters prior to the crib were ZZZ. When the bombe was successful on one Enigma ciphertext, although it would give the reflector, rotor order, and plugboard connections from the daily key on the day the

ciphertext was formed, it would only give the ring settings from the daily key if the last window letters prior to the crib had actually been ZZZ. Finding the actual ring settings from the daily key was necessary in order to be able to decrypt most or all of the other encrypted messages sent between any two Army or Air Force operators on the same day without having to use the bombe.

The first step in finding the ring settings from the daily key was to determine the actual initial window letter and ring setting of the rightmost rotor. These could be found from knowing where a turnover of the middle rotor first occurred, the notch letter of the rightmost rotor, and the rightmost initial window letter resulting from the assumption that the last window letters prior to crib were ZZZ.

To demonstrate this, we will continue the example that we began in Section 3, with additional ciphertext letters included, in which we assume we know that the crib FOLLOWORDERSTO corresponds to the underlined part of the ciphertext NUENTZERLOHHBTDSHLHIYWEABHTQKC. Using this fact, we determined in Section 3 the following parameters from using the bombe on this one ciphertext: rotor order **I**, **V**, **III**; reflector C; ring settings 16, 23, 18; and plugboard pairs O/S, N/T, R/H, B/U, L/I, D/F, G/A, Q/P, C/K, and J/Y. We will now show how the actual initial window letter and ring setting of the rightmost rotor can be found for this example. We will do this using a Maplet entitled **Clonking**, which was written by the authors, and designed for this purpose. Figure 7 gives the result.

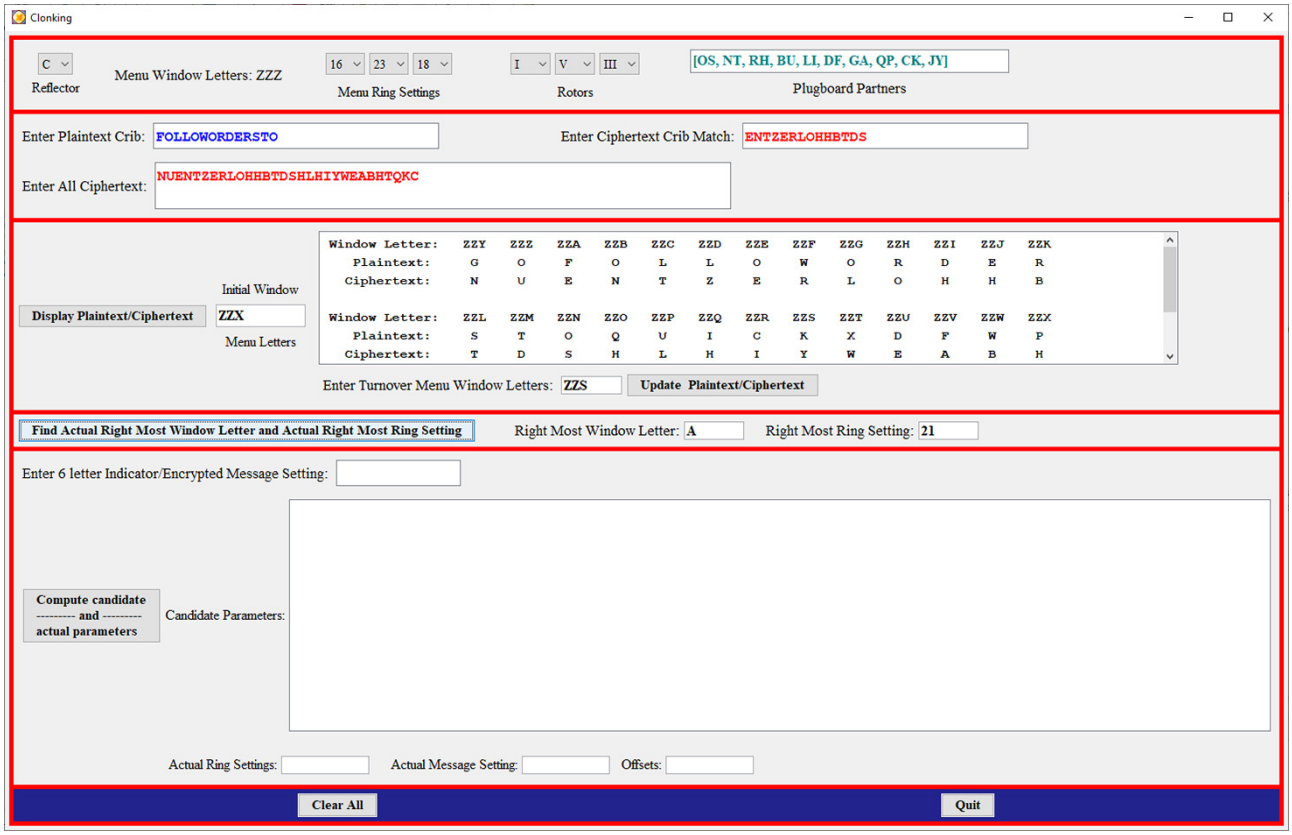

Figure 7: Finding an actual rightmost window letter and ring setting.

In the Maplet, the reflector, ring settings, rotor order, and plugboard pairs discovered using the bombe are entered, as well as the crib, its corresponding part of the ciphertext, and the ciphertext in full.

Clicking the **Display Plaintext/Ciphertext** button then causes the window letters, plaintext, and ciphertext to be displayed under the assumption that the last window letters prior to the crib were ZZZ. From the output shown in the Maplet, we can see that the decrypted message is legible only through the window letters ZZS. The reason for this is very likely due to a turnover of the middle rotor at this point. Since there are two letters in the ciphertext before the start of the crib, the Maplet thus indicates that the actual encryption starts with the window letters ZZX. So to find the actual window letter of the rightmost rotor when the ciphertext was formed, note that since a turnover of the middle rotor first occurred at ZZS and the machine was set to ZZX, then a total of 21 turnovers occurred for the rightmost rotor during encryption. Since the rightmost rotor **III** only has notch letter V, and after 21 turnovers of the rightmost rotor during encryption this notch letter V would have to have been the rightmost window letter, if we go backward on the ring of letters around the rotor (i.e., backward in the alphabet, wrapping from the start of the alphabet to the end if necessary) 21 positions from V, the result would be the actual rightmost initial window letter. Going backward on the ring of letters around the rotor 21 positions from V gives A, which is the actual rightmost initial window letter when the ciphertext was formed. Next, to find the ring setting of the rightmost rotor when the ciphertext was formed, we can use the fact that for the rightmost initial window letter X and ring setting 18 that resulted from the assumption that the last window letters prior to crib were ZZZ, or the actual rightmost initial window letter A and actual ring setting, the rotor core starting positions would have to be the same. For the rightmost initial window letter X and ring setting 18, since X is the 24th letter in the alphabet, the rotor core starting position would be $24-18=6$. Thus, because subtracting the rotor core starting position from the number of the initial window letter would give the ring setting, and the actual rightmost initial window letter A is the first letter in the alphabet, the actual ring setting would be 21, since $1-6=-5=21 \bmod 26$. Clicking the **Find Actual Right Most Window Letter and Actual Right Most Ring Setting** button in the Maplet causes these results to be displayed.

After finding the actual initial window letter and ring setting of the rightmost rotor, codebreakers could then find the actual ring settings of the middle and leftmost rotors by exploiting how they knew Army and Air Force Enigma operators transmitted their message settings as described in Section 4. To demonstrate this, consider an Enigma configured with the correct reflector, rotor order, plugboard connections, and actual initial window letter and ring setting of the rightmost rotor, and with the rotors turned so that the indicator setting shows through the windows. There would then be only $26 \cdot 26 = 676$ different combinations of ring settings of the middle and leftmost rotors. If the letters in the operator's encrypted message setting were decrypted assuming each of these 676 different combinations, only a small number should cause the third letter to decrypt to the actual rightmost initial window letter. It would not be unreasonable to try all 676 different combinations to see which ones caused the third letter in the operator's encrypted message setting to decrypt to the actual rightmost initial window letter, and this is exactly what the codebreakers did. They actually did not even have to decrypt all of the letters in the operator's encrypted message setting for every trial. By rotating the rightmost rotor two positions forward to replicate the rotations that would occur when the first two letters were decrypted, they could initially test a combination by decrypting just the third letter, and then decrypt the first two letters only if the third letter decrypted to the actual rightmost initial window letter.

For example, suppose the sequence `BOXFUW` consisting of the indicator setting and encrypted message setting was intercepted. For the sequence `BOXFUW`, the codebreakers could, with the rotors turned so

that BOZ was showing in the windows, try all 676 different possible combinations of ring settings of the middle and leftmost rotors to see which ones caused W to decrypt as A. Then only for those that did, they could turn the rotors back so BOX was showing in the windows and decrypt FU.

The Allied codebreakers referred to the process we just described as *clonking*. After clonking, from the combinations of ring settings of the middle and leftmost rotors that caused the third letter in the operator's encrypted message setting to decrypt as the actual rightmost initial window letter, they could identify the correct actual ring settings of the middle and leftmost rotors by checking which ones together with the resulting decrypted letters gave the same rotor core starting positions as the initial window letters and ring settings that had worked in cryptanalyzing the individual ciphertext.

As an demonstration of this, consider the example we began in Figure 7, in which we found that the actual initial window letter and ring setting of the rightmost rotor when the ciphertext was formed were A and 21. Clonking for the sequence BOXFUW would involve trying all 676 different combinations of ring settings of the middle and leftmost rotors to see which ones caused FUW to decrypt as ..A, and then for each that did, finding the resulting rotor core starting positions. By entering BOXFUW and clicking the **Compute candidate and actual parameters** button, the Maplet shows all ring settings of the middle and leftmost rotors that decrypt FUW as ..A. Figure 8 shows the results.

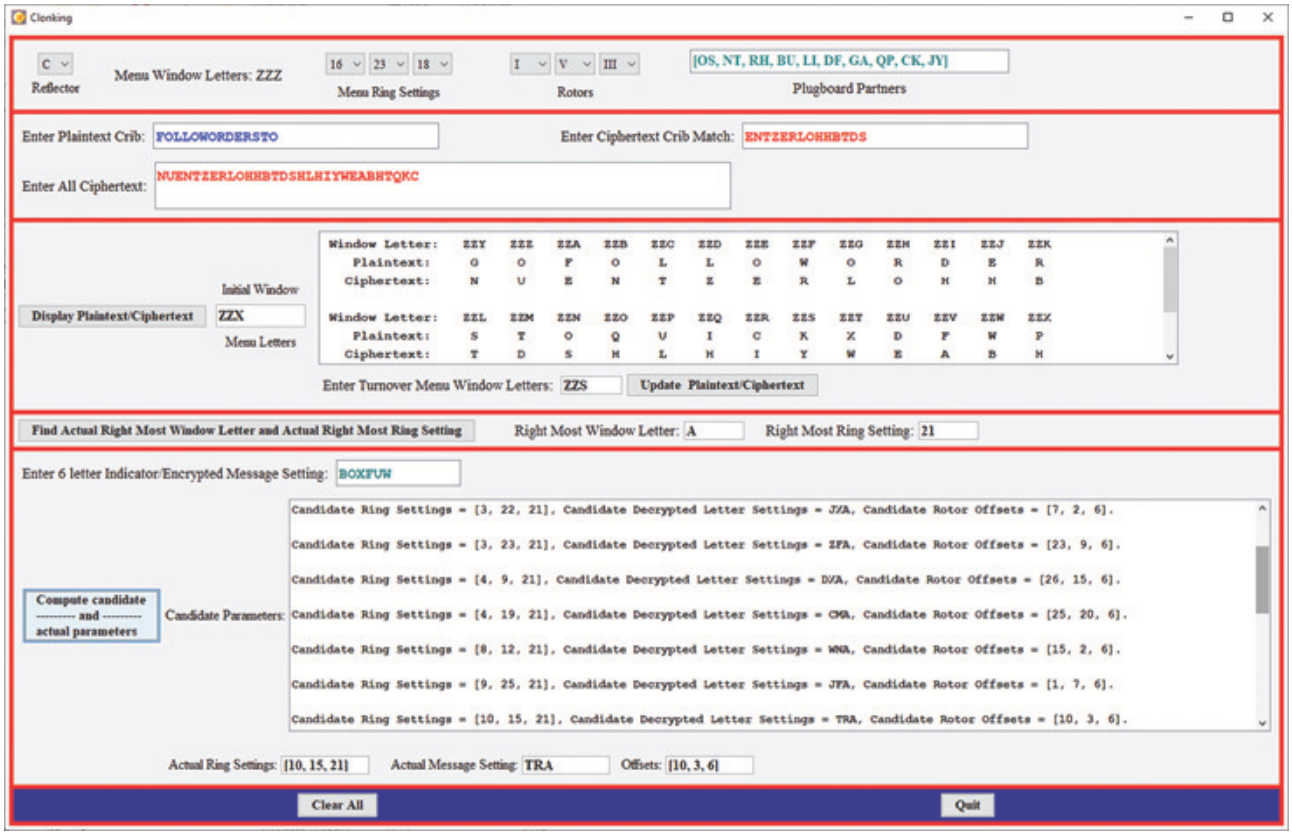

Figure 8: Finding an actual message setting and ring settings.

From this, we see that ring settings 3, 23, 21 cause FUW to decrypt as ZFA. Since ZFA are the letters in positions 26, 6, 1, ring settings 3, 23, 21 with these initial window letters give rotor core starting

positions 23, 9, 6, since $26-3=23$, $6-23=-17=9 \bmod 26$, and $1-21=-20=6 \bmod 26$. The initial window letters and ring settings that had worked in cryptanalyzing the ciphertext in this example were ZZX and 16, 23, 18, respectively. Since ZZX are the alphabet letters in positions 26, 26, 24, the ring settings 16, 23, 18 with these initial window letters result in the rotor core starting positions 10, 3, 6, since $26-16=10$, $26-23=3$, and $24-18=6$. Note that in the Maplet output, the ring settings 10, 15, 21 and decrypted letters `TRA` give these same rotor core starting positions. Thus, the actual initial window letters and ring settings when the ciphertext in this example was formed were TRA and 10, 15, 21. As a result, 10, 15, 21 must be the ring settings in the daily key for all Enigma ciphertexts formed on the same day as the ciphertext in this example. This result is verified in the Maplet. With the correct ring settings found for the daily key for the particular day, codebreakers could use the indicator setting transmitted by each Enigma operator, decrypt their message setting, and then decrypt the ciphertext message sent by the operator.

6 Navy Encryption

At the beginning of World War II, the German Navy used the same three-rotor Enigma machine as the German Army and Air Force. However, in addition to the five rotors used by the Army and Air Force, Navy Enigma operators were equipped with three additional rotors. Like Army and Air Force Enigma operators, Navy operators were issued a monthly setting sheet with instructions on how to configure their machine on any given day. Unlike their Army and Air Force counterparts though, Naval Enigma daily keys were issued in two parts. First, the *inner settings* specified the rotor order and ring settings that were to be used for pairs of days for the month covered by the setting sheet. Then, the *outer settings*, which were changed daily, specified the plugboard pairs and a collection of special positions specified by the rotor window letters known as the *Grundstellung* (in English, *initial position*).

Also in contrast to the Army and Air Force, Navy Enigma operators did not choose their own message settings. Instead, each operator selected a group of three letters called the *procedure indicator group* from a designated section of a printed book called the *K-book*. Then, after configuring their Enigma with the specified rotor order, ring settings, plugboard pairs, and window letters specified by the Grundstellung, the sending operator encrypted the three letters specified by the procedure indicator group. The three letters resulting from this represented the message setting. The process was referred to by Allied codebreakers as *Dolphin*. Suppose, for example, the daily key is the following.

Inner Settings:	Ring Settings:	16, 10, 22
	Rotor Order:	**VIII**, **IV**, **I**
Outer Settings:	Reflector:	B
	Plugboard Pairs:	`O/S`, `N/T`, `R/H`, `B/U`, `L/I`, `D/F`, `G/A`, `Q/P`, `C/K`, `J/Y`
	Grundstellung:	YES

To create the message setting, suppose the operator uses the K-book to look up procedure indicator ATS. The operator would first set up the machine using the inner and outer settings with the Grundstellung as the window letters, and encrypt the procedure indicator. This would result in `SJK`. If the

operator then wished to encrypt TOMORROW IS BETTER, they would change the window letters in the machine to SJK, and encrypt the message. Figure 9 shows the result of using the **Encrypt/Decrypt Enigma Message** Maplet to do this, resulting in the ciphertext CQCLWTIXNNLKFGGS.

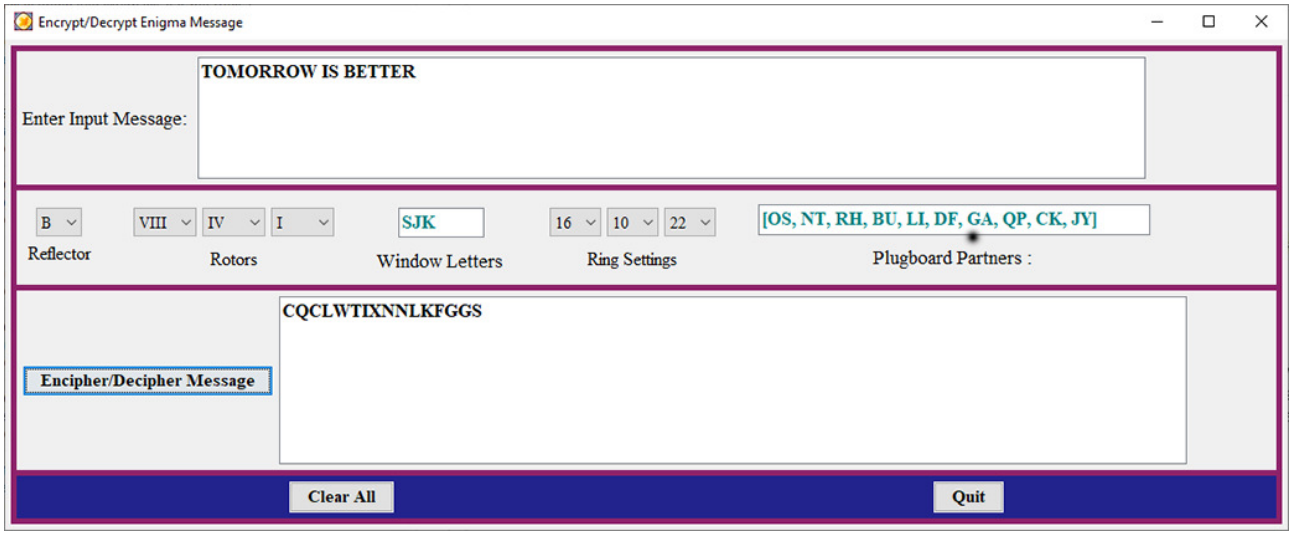

Figure 9: Encrypting a message.

German Navy Enigma operators used a different and more secure method than their counterparts in the Army and Air Force to transmit their message settings and ciphertexts to the receiving operators. The sending operator would encrypt the procedure indicator group by means of a *bi-gram table*. Bi-gram tables, which were arranged in 26×26 grids, were issued in a book to all Navy operators, with instructions for which bi-gram table was to be used on a given day. The **Bigram Table** Maplet, which was written by the authors, is designed to generate a random bi-gram table that results from a given seed. For example, Figure 10 shows the bi-gram table resulting from a seed of 79. Bi-gram tables were used to replace groups of two letters. With a given bi-gram table, the replacement for the bi-gram KA would be the two letter sequence in the position in the table where the row labeled with the letter K intersected the column labeled with the letter A. For the table in Figure 10, the replacement for KA is XD. Note that bi-gram tables were symmetric in the sense that for the two-letter sequence XD, the position in the table where the row labeled X intersects the column labeled D is KA. German Navy Enigma sending operators used bi-gram tables to encrypt their procedure indicator groups. To demonstrate this, as a continuation of the example in this section, recall that the procedure indicator ATS was chosen from the K-book. The operator would first attach a random letter, say J, to the end of this to obtain ATSJ. From the K-book, the sending operator would choose another 3-letter sequence, say PRO, and attach a random letter, say K, to the beginning of it to obtain KPRO. These two four-letter sequences would then be stacked to obtain the following.

K P R O
A T S J

With, for example, the bi-gram table shown in Figure 10, the operator would make the replacements KA $\rightarrow$ XD, PT $\rightarrow$ DV, RS $\rightarrow$ UA, and OJ $\rightarrow$ XK to obtain the following.

X D U X
D V A K

The Enigma sending operator would then transpose this array into the following.

X D D V
U A X K

Finally, the sending operator would string these rows together as XDDVUAXK, and transmit this to the receiving operator twice with the ciphertext CQCLWTIXNNLKFGGS generated in Figure 9 in between: XDDVUAXK CQCLWTIXNNLKFGGS XDDVUAXK.

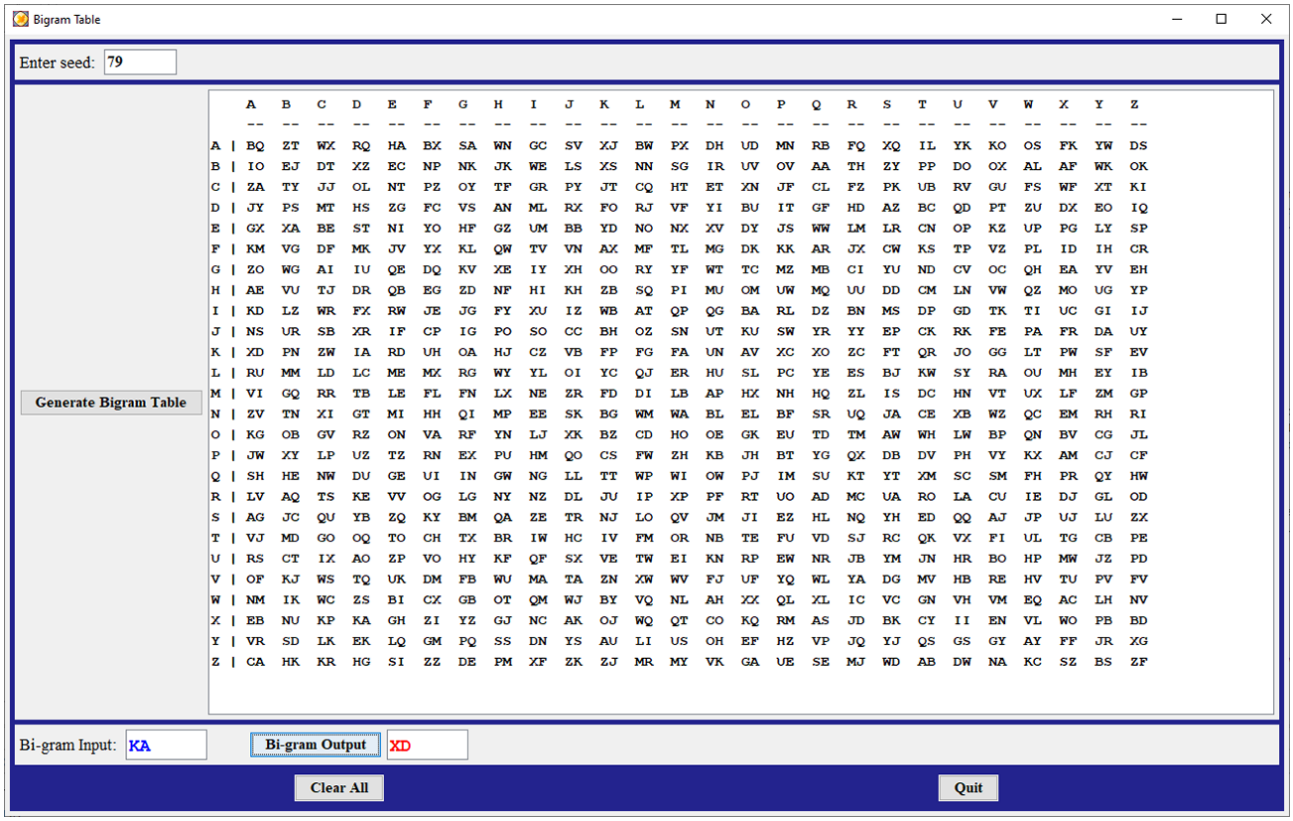

Figure 10: Finding a bi-gram table.

To decrypt the message, the receiving operator would begin by reversing the process of the bi-gram table to recover the procedure indicator group ATS. Then, using the Grundstellung setting of YES and the rest of the machine settings given in the daily key, the receiving operator would encrypt ATS to recover the message setting of SJK. Finally, using this message setting, the receiving operator would decrypt the ciphertext CQCLWTIXNNLKFGGS to recover the plaintext TOMORROW IS BETTER.

7 Navy Cryptanalysis

The fact that the German Navy had 336 possible rotor orders as compared to 60 for the Army and Air Force alone vastly increased the amount of bombe time to discover the initial plugboard pairs,

reflector, rotor order, and ring settings needed to break Enigma messages. Some of this additional challenge was overcome through a technique developed by Alan Turing called *banburismus* which eliminated certain rotor orders. The construction of more bombes to test rotor orders also helped. Even when an Enigma daily key was found with the successful breaking of a single message though, the Navy's use of bi-gram tables to encrypt procedure indicators better prevented the direct recovery of the message settings used by other Enigma operators on a given day. With these added difficulties, it was imperative that the Allies capture monthly setting sheets. The first such capture occurred in 1941, and as the war progressed, other captures occurred. This enabled progress to be made, since it allowed the daily key to be known for a limited number of days. However, without knowledge of the bi-gram tables used by each operator, it was still difficult to recover the message setting used by each operator. This problem was first resolved by forming each day what was known as an *EINS catalog*.

The German word *eins*, which translates in English as *one*, was suspected to occur most often in Navy plaintexts. It was later confirmed that about 90% of German Navy messages contained at least one occurrence of *eins*. Using the daily key provided by the captured setting sheets, a daily catalog of four letter groups obtained by encrypting `EINS` at all of the $26^3 = 17{,}576$ possible window letter position groups was constructed. The daily construction of the catalog was arduous, and was first carried out by hand. Later, a machine known as *the baby* was used, with entries recorded on punch cards.

The **EINS Encryption Finder** Maplet, which was written by the authors, is designed to generate EINS catalogs. It takes daily key parameters as input, and returns via the **Generate EINS Table** button the resulting EINS catalog. Figure 11 shows the catalog table that results from a daily key.

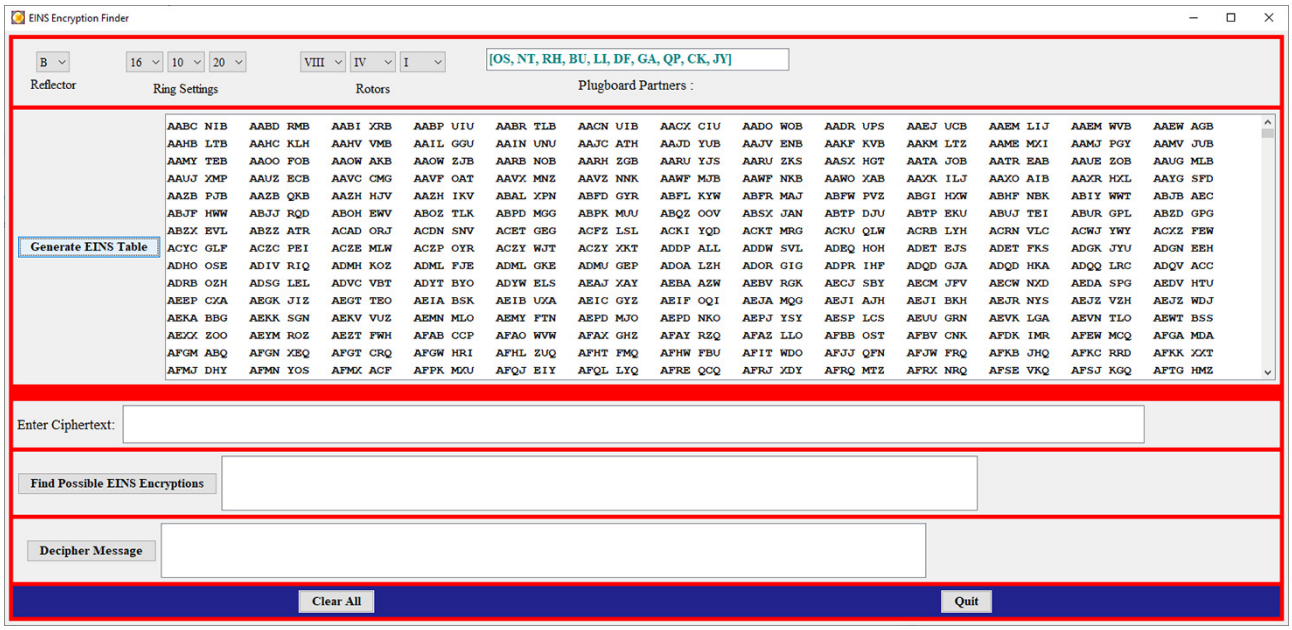

Figure 11: Generating an EINS table.

The table displays all possible ways in which `EINS` could be encrypted with the given daily key followed by the window letters that generate each encryption. For example, the result `AABC NIB` in the upper left of the table indicates that `EINS` encrypts as `AABC` when the window letters are NIB.

Ciphertexts from Navy operators were compared with the entries in the catalog to see if any four letter groups constructed by encrypting EINS occurred in the ciphertexts. For a rotor position where a match occurred, the four characters in the ciphertext would decrypt as EINS. Once a match was located, further letters of the ciphertext would be decrypted to see if legible plaintext was obtained.

As a demonstration of this process, and as a continuation of Figure 11, suppose the ciphertext BPDXHXWOPKPRCDKLWFGXVUTYL was intercepted. After entering this in the **EINS Encryption Finder** Maplet, clicking the **Find Possible EINS Encryptions** button initiates a search for possible encryptions of EINS in the table. Figure 12 shows that there are three matching occurrences in the table, the locations in the ciphertext where each of these encryptions of **EINS** begins, and the window letters that give each one. Clicking the **Decipher Message** button causes the ciphertext to be decrypted with each possible set of window letters, and shows the plaintext as the only legible option: ATCM OCCURS EINS TIME PER YEAR. The Maplet also gives the window letters from the start of the encryption, FBI, found by reverse-rotating the rotors the number of positions in the ciphertext prior to the encryption of EINS from the window letters that produced the encryption of EINS.

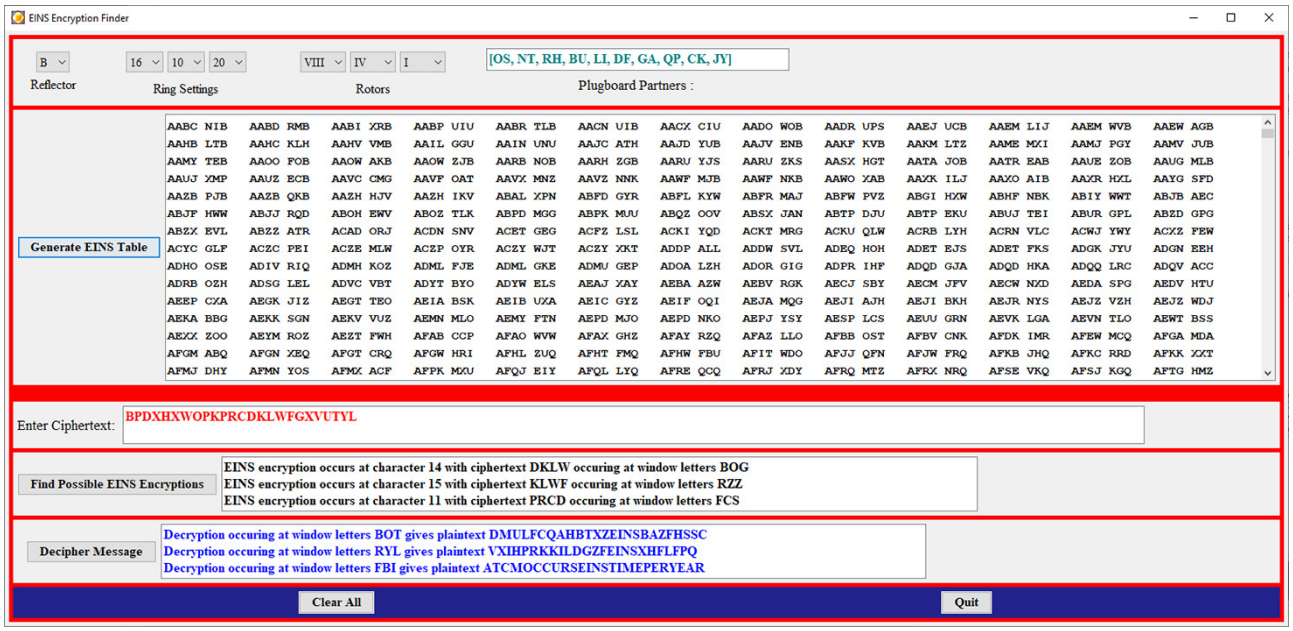

Figure 12: Finding a message setting and decrypting a message.

If approximately 200 message settings were recovered on any given day, it became possible for code-breakers to recover the bi-gram tables used. However, this involved a very laborious procedure.

8 Conclusion

In this paper, we extended our discussion that can be found in [4] to show how the German Army, Air Force, and Navy versions of the three-rotor Enigma could be cryptanalyzed and messages sent by all Enigma operators on any given day read. Due to the more involved method used by Navy

operators to transmit message settings, breaking Navy messages proved a much greater challenge. Indeed, Allied codebreaking efforts were severely complicated in February 1942 by the introduction of a four-rotor version of the machine by the German Navy. Called *Shark* by the Allies, keys for the four-rotor Enigma required further efforts to find, and required faster four-rotor bombes which were developed by the United States in 1943. A description of this process can be found in [1].

9 Supplementary Electronic Materials

[S1] **Turing Welchman Bombe Menu** Maplet illustrated in Figures 2, 4, and 5: https://www.appstate.edu/~klimare/TuringWelchmanBombeExample.maplet.

[S2] **Double Scrambler Calculator** Maplet illustrated in Figure 3: https://www.appstate.edu/~klimare/DoubleScramblerCalculator.maplet.

[S3] **Encrypt/Decrypt Enigma Message** Maplet illustrated in Figures 6 and 9: https://www.appstate.edu/~klimare/EnigmaEncryptionDecryption.maplet.

[S4] **Clonking** Maplet illustrated in Figures 7 and 8: https://www.appstate.edu/~klimare/Clonking.maplet.

[S5] **Bigram Table** Maplet illustrated in Figure 10: https://www.appstate.edu/~klimare/Bigram.maplet.

[S6] **EINS Encryption Finder** Maplet illustrated in Figures 11 and 12: https://www.appstate.edu/~klimare/EINS.maplet.

References

[1] Frank Carter, *Breaking Naval Enigma*, Bletchley Park Trust, Milton Keynes, UK, 2008.

[2] Richard Klima and Neil Sigmon, *Cryptology, Classical and Modern, Second Edition*, Taylor & Francis, Boca Raton, FL, 2019.

[3] Richard Klima and Neil Sigmon, 2021. Recognizing the Polish Efforts in Breaking Enigma. Proceedings of the 25th Asian Technology Conference in Mathematics (ATCM 2020). Available at: https://atcm.mathandtech.org/EP2020/invited/21799.pdf.

[4] Richard Klima and Neil Sigmon, 2018. The Turing Bombe and Its Role in Breaking Enigma. Proceedings of the 22nd Asian Technology Conference in Mathematics (ATCM 2017). Available at: https://atcm.mathandtech.org/EP2017/invited/4202017_21528.pdf.

[5] Neil Sigmon, 2024. Maplet Download Page for Cracking the Enigma Code: Beyond the Bombe. Available at: https://sites.radford.edu/~npsigmon/beyondthebombe/paper.html.