Enigma and BOMBE Simulation

Student Name: A.L. Gillies

Supervisor Name: M. Johnson

Submitted as part of the degree of MEng Computer Science to the

Board of Examiners in the School of Engineering and Computing Sciences, Durham University

Abstract —

Context/Background - The Enigma formed the basis of communication throughout the Axis powers. The BOMBE was the device developed by the British to break it. Technology has advanced since then, but how much? This will be the foundation of this paper, how much faster are we now, than then?

Aims - Using the known execution speed of the antique Enigma and BOMBE, the aim of this paper is to show, through implementation and testing of a modern interpretation of the Enigma and BOMBE, the magnitude of speed up that has occurred since the inception of both.

Method - Both machines have been re-created using modern technology and techniques, they have been timed, and this measurement will be compared to the same measurement on the original machine. Both machines will be brought to the modern day through recreations of them being created in C++ code, the BOMBE will then have further developments applied through the use of parallel computing techniques.

Results - The modern versions of both are far faster than their antiquated counterparts, the BOMBE being roughly 700 times faster, the exact magnitude of which can be seen in the results section.

Conclusions - The significant leaps that have been made in computing in the last 70 years cannot be overstated, The development from electro-mechanical devices to the silicon based computers we have today is astounding. Using the BOMBE as a benchmark, a single core processor forms the basis of 700 times improvement on performance between what was the cutting edge and the current mid-tier commercial offerings.

Keywords — Enigma, BOMBE, Parallel, Modern, Computation, Evaluation, Reinterpretation, Comparison, C++, Antiquated.

I INTRODUCTION

Since Alan Turing's era, the quantum leaps of computational power cannot be understated. Electromechanical devices that have a single function have developed into electronic devices capable of renders and simulations of electromechanical devices within themselves. The BOMBE is an example of an electromechanical device which can be rendered and simulated within a modern mobile device. The BOMBE was, at its time, at the forefront of computer development. The advancements within this field mean we are able to study models of the universe, and other academic pursuits inconceivable at the time of Turing.

Turing and his team are credited with the creation of the BOMBE, which in turn is credited with breaking the German Enigma codes. These codes were encrypted using the Enigma machine, another electromechanical device with a single function; it was used to encrypt messages then sent to another enigma machine which then decrypted the message. All German military orders were relayed in this fashion and the breaking of this code is said to have shortened the war by several years and saved innumerable lives.

The aims of this project are two fold; the first is to show the individual effectiveness of both machines, the Enigma and its British counterpart, the BOMBE. This will be done via the implementation of both in a modern setting. The second aim is to outline the developments that have occurred since the inception of the Enigma and BOMBE by showing the significant increase in computational efficiency. An analogue for this would be how fast the BOMBE breaks the Enigma code in a modern setting. The most simple solution will be a BOMBE representation using modern coding techniques without any parallel improvement or other modern techniques. The representation will be simply a copy of the Enigma in C++ code. The project will outline the improvements generated by sequential electronic devices, common since the 1980s, over an electromechanical device. The second, more complex, solution will incorporate parallel programming, introduced in the early 21st century, and still an improving technology.

A Deliverables

Unique ID	Deliverable	Description
DL1	Simulate the Enigma Machine	Have a working Enigma machine that can take in settings and plaintext and return ciphertext.
DL2	Simulate the BOMBE	A working BOMBE that takes in settings and ciphertext and returns decoded plaintext.
DL3	Show the correctness of the Enigma Machine	Show through a series of tests and validations that the plaintext fed to the Enigma Machine and the Ciphertext returned are paired through the settings of the Enigma.
DL4	Show the correctness of the BOMBE	Show that the known plaintext of the ciphertext input into the BOMBE is exactly the same as the plaintext that is returned by the BOMBE
DL5	Evaluate average computation time over set inputs for both Enigma and BOMBE	Using a wide range of known plaintext - ciphertext pairs find the average computation time, the time taken on average to do a single encode or decode for both simulations

DL6	Parallelise the BOMBE and evaluate average computation time	Using known parallelisation techniques increase the speed at which the BOMBE will decode a given ciphertext.

Table 1: Deliverables

II RELATED WORK

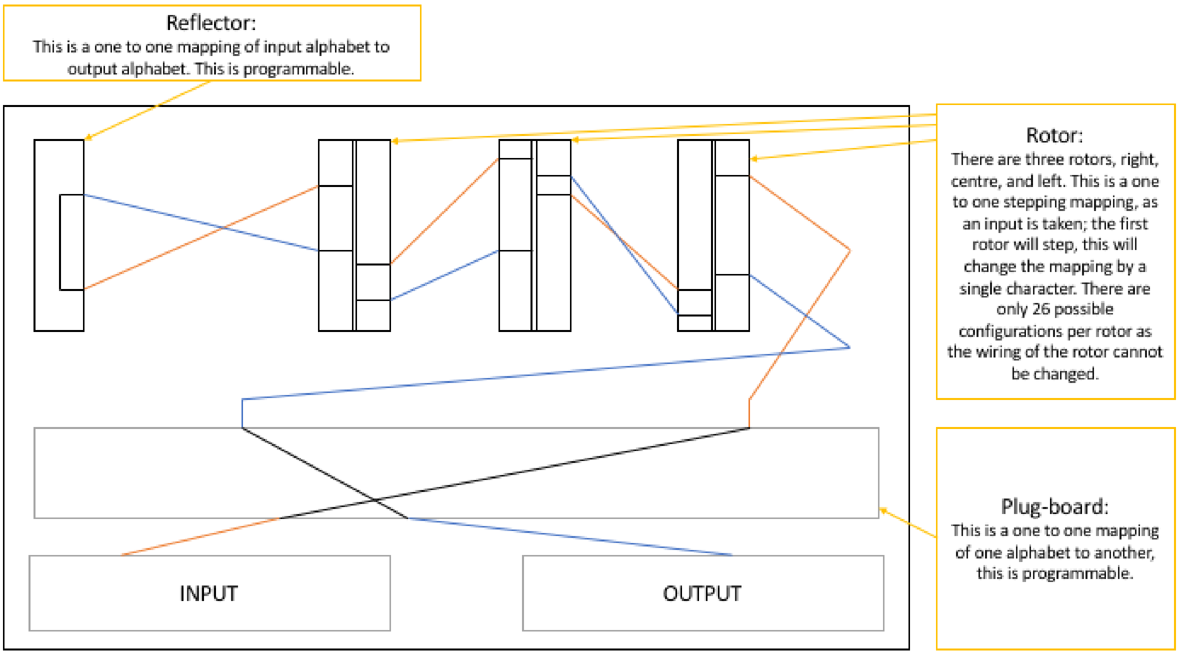
The electromechanical device known as the Enigma machine is best known for its use by Nazi Germany. Designed by a company called Scherbius & Ritter in 1918, it was then sold to the German armed forces after a few incremental upgrades. The machine is designed to allow secure cryptographic messages between two of these machines; the first will encode the message using decided settings, then the second will, using the same settings, decode the message. A set of alphabet mapping is done which provides a one-to-one relationship between one alphabet and another, one letter is mapped to another and this relationship is bidirectional. There is a caveat with the Enigma that a letter from one alphabet cannot map to the same letter in another. The heart of the Enigma is based around the changing mappings that occur through its rotors, which shall be explained in technical detail later in this paper.

Although the Enigma machine is made up of 7 parts, only 5 important parts plus an additional component and some wiring will make it work. The important parts are the plug-board, the three rotors, and the reflector. The other parts are a converter that allows all the separate components to work together, along with a lamp-board to show the encrypted letter, and the final component wiring to connect them all together.

The plug-board is a mapping from one alphabet to another that is a manual process of re-plugging 26 wires that map from one to the other. This does not change for each letter that is input into the device, unlike some of the components we will see later. The reflector is much the same as the plug-board, except that it cannot map a letter to itself. It is a simple one-to-one mapping that does not change over time. In the earlier designs of the Enigma machine it could not be rewired, and instead had to be swapped out for another reflector with different wiring. In the same way to that of the rotors, as we will discuss later in the paper, with later developments came the ability to rewire it. The rotors are the core of the machine. These are three modular circular mappings that are altered each time a letter is encrypted by the machine. The inner ring and outer ring each have contacts on them; when the outer ring is rotated then the mapping will change. There are 26 different mappings that can be used. The three rotors are chosen from a set of available rotors and each one has a unique ordering of letters on both its rings. The order in which the rotors are input into the machine is, therefore, very important.

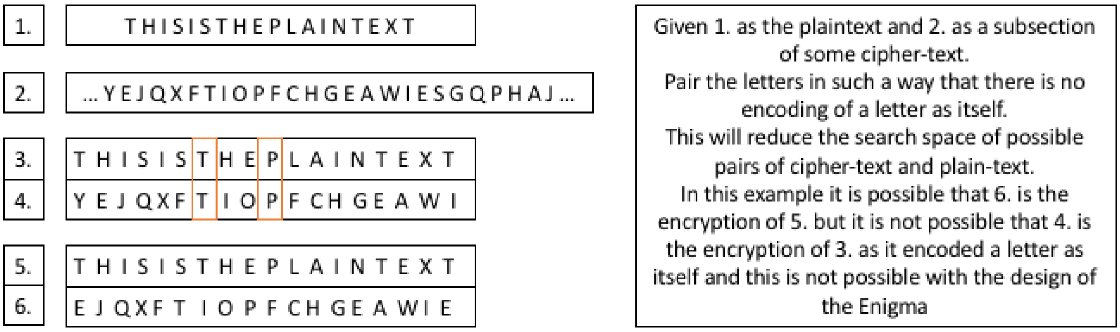
The Enigma machine takes the first letter of the message that is to be encrypted from a keyboard and passes it through the plug-board. It will then go to a converter which allows the signal to be passed to the first rotor, another mapping, then the second, and the third. In each of these cases the inner ring of the previous rotor is connected to the outer ring of the next. When the reflector is then reached, the mapped letter is then passed to the inner ring of the third rotor and this is a reverse mapping. The outer ring of the third rotor is connected to the inner ring of the second rotor and thus the signal will pass back through the rotors in this fashion. Once the plug-board is reached, the reverse mapping is done and a light will trigger on the lamp-board showing the letter to which the input letter has been mapped. After this the rotors are then stepped; this means that the first rotor will rotate one position for each letter that is encrypted. The second rotor will only step when specific notches are reached on the first rotor; the third is the same but for notches on the second rotor. This method means the first rotor will rotate more than the second and the second will rotate more than the third.

Figure One


Reflector:
This is a one to one mapping of input alphabet to output alphabet. This is programmable.
Rotor:
There are three rotors, right, centre, and left. This is a one to one stepping mapping, as an input is taken; the first rotor will step, this will change the mapping by a single character. There are only 26 possible configurations per rotor as the wiring of the rotor cannot be changed.
Plug-board:
This is a one to one mapping of one alphabet to another, this is programmable.
INPUT
OUTPUT

The BOMBE, developed at Bletchley Park, was very similar to the Enigma but its antithesis. The BOMBE was based on a probable phrase attack; this means that for the BOMBE to work there needs to exist a known or guessed portion of ciphertext and its corresponding plaintext. An example of this used regularly was the German equivalent of the radio Weather Report; this was repeated daily and could easily be guessed. Once confirmed, then the exact letter-to-letter transformation must be established; this is done through the matching of letter-to-letter in the phrase such that no letter is encoded as itself (impossible for the Enigma). Examples of these matchings are shown in Figure Two.

Figure Two



1. THISISTHEPLAINTEXT

2. ...YEJQXFTIOPFCHGEAWIESGQPHAJ...

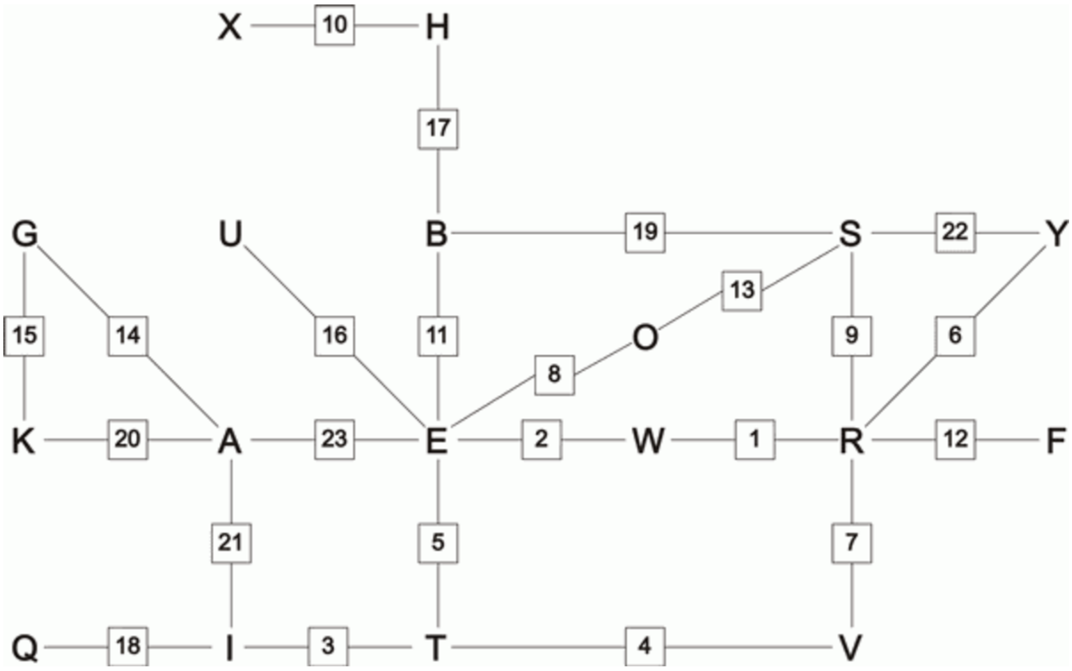
3. THISISTHEPLAINTEXT
4. YEJQXFTIOPFCHGEAWI

5. THISISTHEPLAINTEXT
6. EJQXFTIOPFCHGEAWIE

Given 1. as the plaintext and 2. as a subsection of some cipher-text.
Pair the letters in such a way that there is no encoding of a letter as itself.
This will reduce the search space of possible pairs of cipher-text and plain-text.
In this example it is possible that 6. is the encryption of 5. but it is not possible that 4. is the encryption of 3. as it encoded a letter as itself and this is not possible with the design of the Enigma

The next stage of pre-processing is the creation of crib-ciphertext pairings; this is the mapping of a specific letter at a specific set up of the Enigma. We can see in the figure above that T maps to E at the first set up of the Enigma. At the second set up of the Enigma, because of the rotor stepping, we are now in a different set up, H maps to J, and so on. These pairs can then be joined to create a graph of pairings with the numbered setting as a label for the edge, a random example of which is shown in Figure Three.

Figure Three



There are significant implications that can be drawn from the connection network which can be seen in the graph of pairings in Figure Three. Using the connection network, the search space can be reduced by using the implications of some transformations on others. We first must recall that each transformation consists of two 'stecker swappings' and a 'scrambler encryption' - these are the set of rotor transformations and reflector respectively. A 'stecker swapping' is the transformation of a letter through use of the three rotors present in the Enigma, a 'scrambler encryption' is the transformation via the reflector; a further 'stecker swapping' is the use of the reverse direction of the rotors. A relative position is the numbered set up that the Enigma is currently in, so the first letter input into the Enigma machine is using relative position one. This is simply a set up of the Enigma. Relative position two is the set up of the Enigma given that a letter has already been input, and that the set up has changed due to the way the rotors behave. Using the connection network above, here is an example of the search space reduction that can be done using these details.

For example, let us test the hypothesis that E is mapped to K through the first three rotors, also known as E is ”steckered” to K. Now consider how E is transformed into A at relative position 23, this is shown on the connection network above.

If E is ”steckered” to K, K will be input to the reflector or ”scrambler” at relative position 23 which will output some other letter L1. Since we know that E transforms into A at relative position 23, we know that A must be ”steckered” to L1.

But if A is ”steckered” to L1, L1 will be input to the ”scrambler” at relative position 21 which will output some other letter L2. Since we know that A transforms into I at relative position 21, we know that I must be ”steckered” to L2.

But if I is ”steckered” to L2, L2 will be input to the ”scrambler” at relative position 3 which will output some other letter L3. Since we know that I transforms into T at relative position 3, we know that T must be ”steckered” to L3.

But if T is ”steckered” to L3, L3 will be input to the ”scrambler” at relative position 5.

Now consider the output of the ”scrambler” at relative position 5 when it receives L3 as its input. Since we know that T transforms into E at relative position 5, we know that the output letter of the ”scrambler” at relative position 5 must be ”steckered” to E.

Suppose, when given L3 as its input, the ”scrambler” at relative position 5 outputs J. By our original hypothesis E is ”steckered” to K, but by our chain of implications E is ”steckered” to J, so in this case E is ”steckered” to both K and J. But the construction of the Enigma does not permit one letter to be ”steckered” to two letters, so E cannot be ”steckered” to both K and J, so our original hypothesis has resulted in an impossible consequence and so must be false.

But now suppose, when given L3 as its input, the ”scrambler” at relative position 5 outputs K. Then by our original hypothesis E is ”steckered” to K, and by our chain of implications E is ”steckered” to K, so in this case E is ”steckered” only to K, thus our original hypothesis has resulted in a consistent consequence and so may be true.

The altered connection network below shows the hypothesis that E is ”steckered” to K via the steps of implication that we have been through.

The preprocessing that was done helped reduce the search space such that it would have to be iterated through by the BOMBE, thus reducing the time taken for a result to be found.

The way the electro-mechanical device was created is very similar to an exploded view of the Enigma; it was created so that there would be no return journey through the rotors, instead another three rotors were added after the reflector. They were added in such a way that one pass through these 7 components is the equivalent of a pass through to the reflector and back through the rotors of the Enigma. The BOMBE was made up of three batteries of 12 sets of these 7 components. With the use of a ”menu”, also known as the connection network, this monstrous machine could be wired up so as to eliminate as many combinations as possible.

The next stage of the BOMBE set up is the plugging-up of the BOMBE. This is done with the help of what is known as a diagonal board, created by Gordon Welchman, as it encapsulated the reciprocal property of the Enigma, meaning that if R was steckered to Q then Q was steckered to R and so on for every pair in a specific Enigma set up. This could be used to eliminate parts of the search space. The diagonal board was used in conjunction with the menu or connection network to plug-up the BOMBE. The diagonal board consisted of 26 cables each containing 26 wires leading into and out of it, with a total of 325 connections between them. For example, the a wire in the B cable was permanently connected to the b wire in the A cable, and the a wire in the C cable was permanently connected to the c wire in the A cable, and so on. The effect of the diagonal board, when attached to the BOMBE, was to increase the feedback into the double-ended scramblers, and thus reduce the necessity of obtaining loops in the crib-ciphertext pairing. This, in turn, made it possible to use shorter menus which were less likely to include a turnover of the Enigma's middle rotor during the encipherment of the plaintext. This is highly advantageous for reasons that will be outlined later. Once a menu and diagonal board are in place it will be plugged up such that each rotor bridges two of the diagonal board's cables. When all the rotors have been connected in this way they are rotated to the positions in accordance to the relative positions in the menu. The drums are mounted onto their shafts next. We first select a particular rotor order to try. In this example we try rotors one, two, and three of eight. We mount nine rotors, each of which corresponds to rotor number one on the first nine top shafts in a battery, then we mount nine rotors each of which corresponds to rotor number two on the first nine middle shafts in the same battery. We then mount nine rotors, each of which corresponds to rotor number three on the first nine lower shafts in the same battery. The other two batteries are now plugged up in exactly the same way except that different drum orders are selected. Note that the loops in the menu are represented by loops in the plugged-up BOMBE.

We will go through a simplified example of this using an eight letter alphabet, as using twenty six would be overwhelming. We will use the first eight letters of the alphabet, and the principles will be identical to that of the twenty six alphabet version of the BOMBE.

1	2	3	4	5	6	7	8	9
B	E	A	C	H	H	E	A	D
E	D	B	G	E	A	H	D	B

Table 2: Mapping Example

Figure Four: From this table we can derive the menu or connection network.

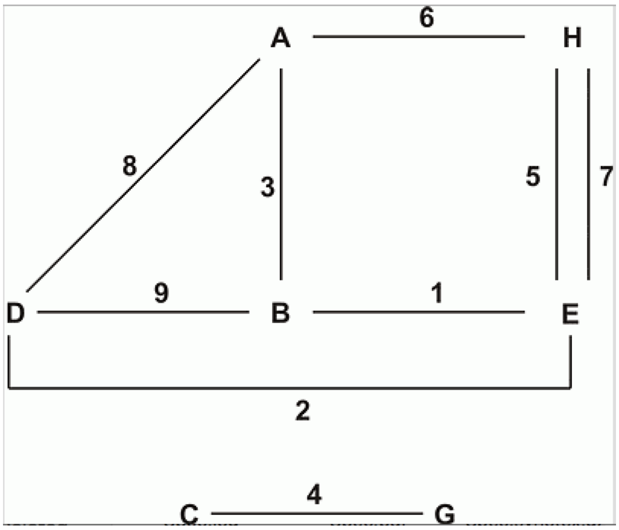


Figure Five: This is the equivalent diagonal board for an 8 letter alphabet.

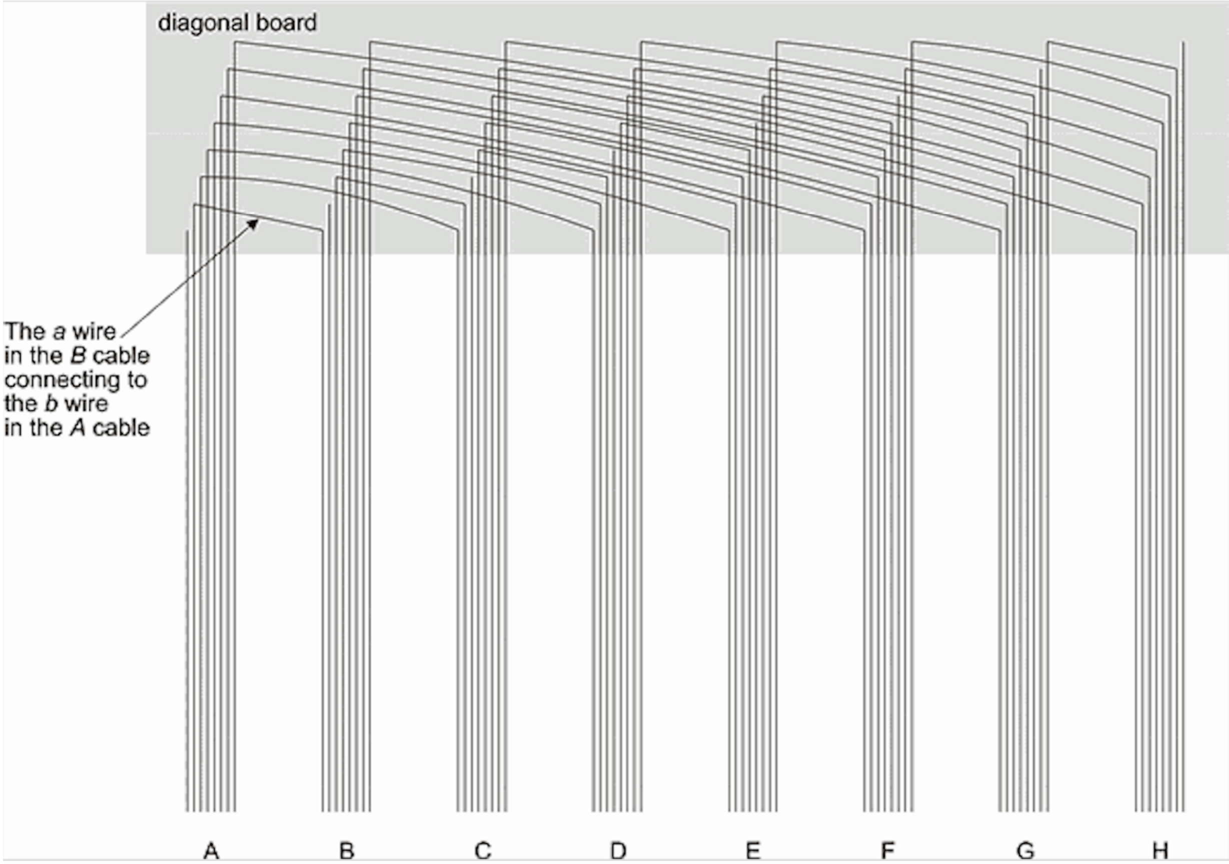
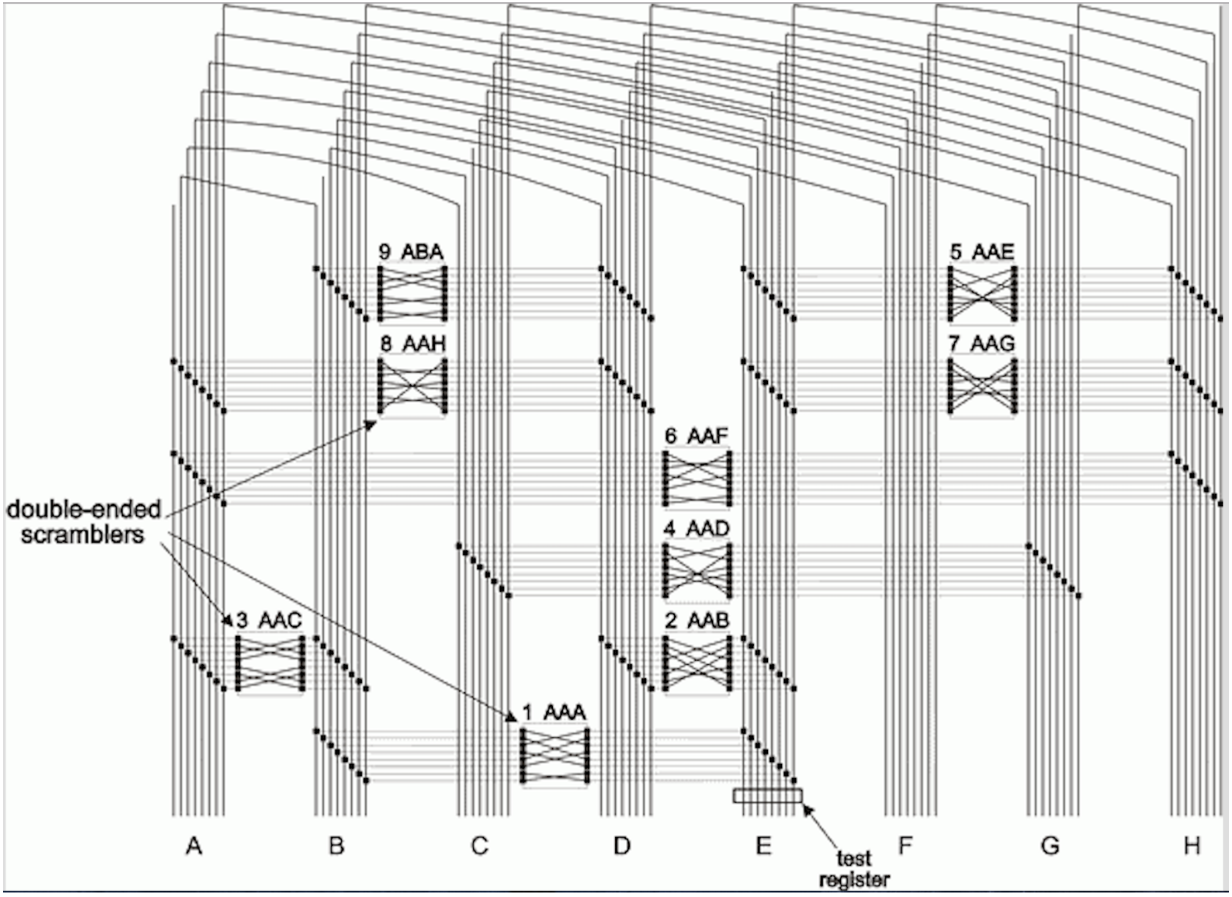

diagonal board
The *a* wire in the *B* cable connecting to the *b* wire in the *A* cable
A
B
C
D
E
F
G
H

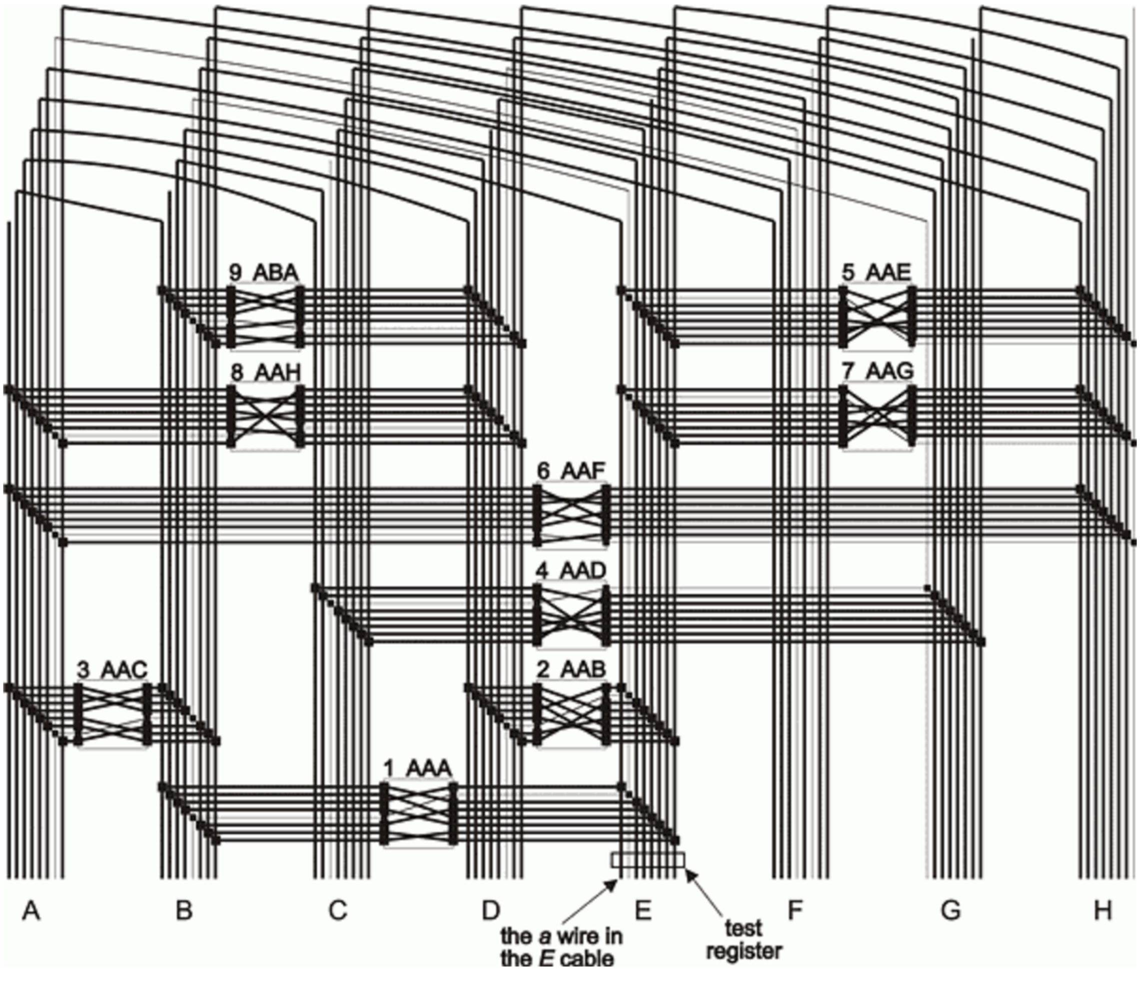
Figure Six: This is the fully plugged up diagram of an eight letter BOMBE.



Once the BOMBE has been plugged up, it will be set in motion. This consists of the top row of rotors stepping at a rate of 120rpm, with the second row of rotors at 1/26th of that and the third row at 1/26th of the second row. They will continue to step until a ”Stop” is reached or the rotors return to their original position. A ”Stop” is defined as the transformations produced by the settings used yield the ciphertext from the crib. This was known by the BOMBE due to the test register. This was attached to one of the cables of the diagonal board; the 26 wires in a cable pass through the test register, with it being able to tell how many of its wires were ”live” at any given time. When the BOMBE was started it applied a voltage to an arbitrary wire in the test register's cable, say the a wire in the E cable, to represent the hypothesis that A is steckered to E and E is steckered to A. Every wire connected to the a wire in the E cable and the e wire in the A cable would immediately become live, but suppose that another wire in the E cable, for example the h wire, becomes live. This represents the hypothesis that H is steckered to E and E is steckered to H, but E cannot be steckered to both A and H so we have arrived at a contradiction and we know that this rotor order, with this rotor position, and the hypothesis that A is steckered to E, is false. Having applied the current to the a wire in the E cable, and therefore the e wire in the A cable, and having had the current return to the h wire in the E cable and therefore the e wire in the H cable, we note that the current will pass through any scrambler connected to the A, E and H cables to wires in other cables. These will not generally be the wires already energised in the A, E and H cables. Thus, as the current flashes through the cryptoanalytical circuitry of the BOMBE feeding back into cables and scramblers, more and more wires become live until, with a good menu, the feedback will almost always make all the wires in the test register live. There are two notable exceptions.

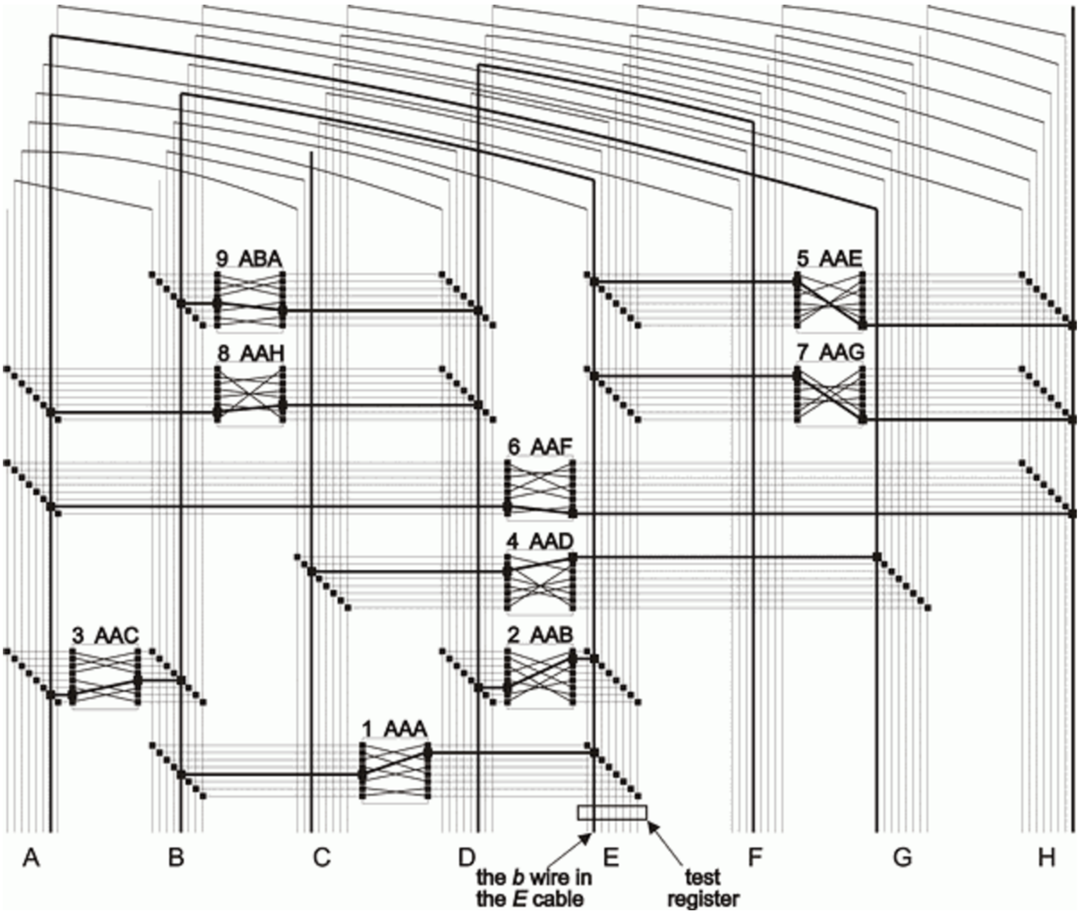
Suppose that the rotors are in the correct position but that the hypothesis being tested is false. For example, suppose that the rotor order and position is correct, and that the false hypothesis that E is steckered to A is being tested. This situation is shown in Figure Eight in which the thick black lines indicate those wires which are live after a voltage has been applied to the a wire in the E cable. In this case the wires representing the true hypotheses are not connected to the wires representing the false hypotheses. However, for the reason discussed above, the wires representing the false hypotheses are connected together, so when any one of them has a voltage applied to it the voltage appears on all of them. So, in this case, if we look in the test register we will find exactly 25 live wires and 1 dead wire. The dead wire, and all the wires connected to it, indicate stecker swapping consistent with the crib-ciphertext pairing. See Figure Six.

Figure Seven: The bold wires indicate a ”live” wire. The dead wires in the cables indicate that A is steckered to G, B is steckered to E, C is self-steckered, D is steckered to F, and H is self-steckered.



Now suppose that the rotors are in the correct position and that the hypothesis being tested is true. For example, suppose that the rotor order and position is correct, and that the true hypothesis that E is steckered to B is being tested. This situation is shown in Figure Seven, which has exactly the same scrambler positions as Figure Six, and in which the thick black lines again indicate those wires which are live, this time after a voltage has been applied to the b wire in the E cable. In this case the wires representing the true hypotheses are not connected to the wires representing the false hypotheses, so when a voltage is applied to any one of the wires representing a true hypothesis, the voltage appears on all the other wires which also represent true hypotheses, and on no other wires. So, in this case, if we look in the test register we will find exactly 1 live wire and 25 dead wires. The live wire, and all the wires connected to it, indicate stecker swapping consistent with the crib-ciphertext pairing. In the example shown in Figure Seven, the live wires in the cables indicate that A is steckered to G, B is steckered to E, C is self-steckered, D is steckered to F, and H is self-steckered; in other words, the steckering that the BOMBE has found is identical to that of Figure Seven. The wires, which are live in the test register in these two situations, are the logical complement of each other. So there are two ”Stop” conditions which the BOMBE looks for - either exactly one live wire or exactly 25 live wires in the test register.

Figure Eight



Once a stop had been established, it would then be tested. This was a manual process, usually done by one of the operators of the machines known as wrens. The question may now be asked, ”What about the plug-board?”. This was still done manually and only done after a stop had been found. The breaking of the plug-board is straightforward enough, due to the fact it does not change as the letters are input, so, if the settings detected by the stop are correct then this can be broken by frequency analysis. Frequency analysis is the process of matching frequency of letter in the plaintext language and the letter frequency of the ciphertext language. For example, if in the ciphertext every other letter is an E (meaning that half of all letters are an E) and in the corresponding ciphertext half of all the letters are Q, then we can have an educated guess that E maps to Q. This is an extreme example, but the principle holds and can be applied to every letter in the alphabet used with the same process. This is how the plug-board is broken using the settings detected by the ”Stop”.

III SOLUTION

A Enigma

C++ was chosen as the implementation language for the Enigma mainly due to its rich library. It has support for a wide range of external libraries, the main one we will be focused on in this case is the Intel tool set. This may not be much use when it comes to the Enigma but it is a very strong and well established tool that allows C or C++ code to be parallelised which is the aim of the BOMBE implementation that will be outlined later. The reason that C++ was chosen specifically for the Enigma implementation is that it is a middle level language and thus enables the benefits of low level and high level languages as well as being highly portable and non operating system specific. It is also very well established in the teaching at Durham University.

The recreation of the Enigma machine was to be done in C++ and this was chosen as it has significant parallel tools available for it, in the form of the Intel C++ compiler - a tool that has shown great capabilities in parallelism. This would be very helpful in the recreation of the advanced BOMBE as developing in two separate languages didn’t make sense.

The recreation revolves around a settings file; this is a text file that contains all the settings that will be used by the Enigma machine to encode the message it is given through the command line interface. These settings include the plug-board mapping, the reflector mapping, the rotors that will be used, and their corresponding initial displacement. This file is then updated automatically to reflect the changes that have occurred to the settings as the message is encrypted. Each letter will have a different set of settings as the displacement of the rotors changes and this will be reflected in the settings file. The C++ file that makes up the core of the code that has realised Enigma, the main file, will manage all the changes that occur. In addition it will manage user interaction with the device. It has a central location with which to manage the other C++ files that realise different parts of the Enigma machine.

The plug-board, reflector, rotors, as well as a converter file, have all been created in their own separate files so as to improve the modularity of the system, as well as to provide a more focused development route; each file can be developed without removing functionality for the others.

The main file contains the instructions on how to use the system, it contains all file dependency instructions, as well as containing the set up function and main function. The set up function will be run before the system is usable, it will find the set up file, using the information that it contains, it will then make the system usable through the main function of the main file. As the set up function is run, it will call the set up functions of all the other parts of the Enigma in other files, the plug-board file will store its mapping, the reflector will store its mapping and the rotors file will store the rotors that are currently in use, and make sure that the displacement is correct for each one. Once this has been done, the main function is used. This will take in user input through the command line, the only thing that will be input is the plaintext. This is the message to be encrypted. The main file will then iterate through each of the letters in the message and encrypt them individually, using unique settings for each letter. The first letter will use the settings that have been read in by the set up file. The first stage is to convert the letter to its corresponding number; for example, A is equal to 1 and Z is equal to 26. This makes use of the converter file. This file contains functions to convert between letters and numbers as well as another function that does the reverse. This was done because numbers are much more manageable than letters in terms of computation. Once this has been done the number will then be sent to the plug-board. The plug-board stores the mapping from one set of 26 numbers to another set of 26 numbers. This is a computational realisation of the alphabet-to-alphabet mapping that is done by the electromechanical Enigma device. Once this has been done in the required direction, the plug-board has two directions mapping forwards and mapping back. We will use the mapping forward function now, and the mapping back function at the end to receive the encrypted message. We have now finished the mapping through the plug-board and will move onto the first rotor. The first rotor will have been chosen by the user in the set up file as well as its initial displacement from its standard position. This is another alphabet to alphabet mapping, but with a twist. The twist is that each letter does not have the same mapping. Once a letter has been encrypted the Enigma machine will increment the displacement of the rotor, and once the first rotor has reached a specific displacement, it will increment the displacement of the second rotor. This in turn may reach a specified displacement and cause the third rotor to increment its displacement. This is only done once the current letter has been fully encrypted. It has passed forwards through every rotor as well as back through them. Once the letter has been mapped by each of the rotors, it reaches the reflector. This is very similar to the plug-board in it can be re-programmed in a very similar way. This mapping is then done, with the caveat that a letter cannot be encrypted as itself at this point, and the return journey begins. The letter will travel back through each of the rotors, this time being mapped in the opposite direction, then to the plug-board, which will also map in the backwards direction as mentioned before. This concludes the encryption of this letter. In the electromechanical Enigma this would cause a letter to light up on the device to show which letter the input letter has been encrypted as. In this device the converter file will be used. This is because, throughout this process the 'letter' has actually been the number corresponding to that letter. In order to simplify the process, we will map from number to letter and output this encrypted letter. Once we have a fully encrypted letter, the rotors will change their displacement as the process will start again for the next letter in the message. This will continue for all letters in the message until we reach the end. At this point, the new setting that the system is using, will be output to the settings file. This is done so that we are not reusing the same setting each time we encrypt something, as the set up function will be reading in the same values again and again from an unchanged settings file.

Once the Enigma was formed in its component parts, each was tested for expected behaviour. The plug-board was set up, and then each letter of the alphabet was input, and the output validated. This was done multiple times with random settings as doing 26! tests was not feasible. The reflector was the next thing to be tested. This was done in the same way - each of the hard-coded reflectors are tested, then an identical test to the one performed on the plug-board is done, the reflector is reconfigured, and then tested. This is repeated until it is deemed that enough of the settings space has been covered. The rotors were also tested. There are eight rotors that can occupy any one of three positions, with no repetitions, and there are 17576 unique paths through any three rotors. This results in 984,256 possible unique routes through the rotors, with any three rotors chosen. Once all the individual components had been tested, and passed their respective tests, the machine was set up in its entirety, meaning that we have a working Enigma. A few preliminary tests were done to make sure that the Enigma behaved as expected. These were known paths through the Enigma done outside of the system manually, then shown to behave the same within the system.

The Enigma was built in pieces, firstly to make sure that each functioned as expected, and could be tested independently of the others. Secondly, that the development style would be able to avoid a waterfall model, where one thing is based on another and then another is based on that. This builds up a very significant reliance relationship between the components and thus causes significant issues if one of the first components to be created, is found to have an issue further down the development chain. Instead of this method, an agile method was adopted. Each component was developed individually, and simultaneously, so at a foundation is established for each component. This is then built upon until we reach the point that all components function as expected and they can then be used together in the final system. This circumvents the issue that a broken component results in more development time on a component that was reliant on the broken component, but was otherwise working as expected.

B BOMBE

The main development tool for this project was the Intel toolset. This allowed testing of the hypothesis with the least overhead. This means that it is a mature and well established tool that allows parallel techniques to be applied simply and with confidence. The specific reason that this tool was used in conjunction with C++ was that Fortran, the only other option with this specific tool, is seen by many as an antiquated language it is rather steep with its learning curve and has already a supposed high barrier for entry, this was passed over for the ”relatively easy to start and learn” C++.

A parallel technique is any theory of parallelism that can be applied in practice. A great example of this would be the idea of a master with slave processors. The master will manage everything and the slave or slaves will do all the work. This is implemented through assigning a leader processor, the master, with the rest as slaves. The master processor will manage data access, work loads, and synchronisation across the slaves, along with any other management tasks. The slaves are given work and execute it to the best of their abilities. They then send it back to the master for synchronisation with the other pieces of work that were given to the other slaves.

The first version of the BOMBE that was recreated in C++ code never actually existed. This was made as a benchmark for the others. In this version, we will use a brute force approach with no limitations; that is, we will try every combination of setting possible to attempt to map from the input encoded message and the guessed message that it corresponds to. This is done through a similar method to that which was used to make the Enigma Machine. In fact, a copy of the exact Enigma machine code was used in this implementation. The only difference is the main file and settings file. The main file will rewrite the settings file with each possible setting and then it will be tested. If the results are wrong, then the next setting is used. This is done until the settings that were used in this case are found. They will then be output by the main file so that they might be used by an Enigma clone to break any other messages sent in the same block as the message just broken, and thus have almost identical settings. The brute force approach is slow. The reason that this approach is slow, is that there are a lot of settings that have to be tested. Each of the components have their own complexity, and thus each have a number of settings that need to be iterated through. The plug-board has 26 letters in it. This means that there are $26!$ ways that this can be encoded. This does not change through usage however, and this is a useful feature in the later BOMBE recreations; but for this implementation, it is one of the more challenging components. The rotors of which there are eight rotors that can be chosen to occupy any one of the three positions. This means that there are 56 different set ups that can be used from the eight rotors. Each rotor then has 26 different displacements that can be used, meaning that in total there are 17,576 different settings, after three rotors have been chosen, and 984,256 different set ups if any of the eight rotors can be chosen. The reflector has a very similar complexity as the plug-board, however it cannot encode a letter as itself, meaning that there are $25!$ ways that it can be encoded. The results of this are that the brute force BOMBE has to go through $6.157\,05 \times 10^{57}$ different settings. The exact number of settings that need to be tested is actually; $6\,157\,051\,363\,093\,094\,576\,378\,463\,970\,354\,900\,547\,389\,095\,936\,000\,000\,000\,000$. This was known from the start and is the reason it was done. This is the slowest that a BOMBE could possibly be using modern techniques.

The BOMBE recreation in C++ code was done in a very similar way to the Enigma. It is a main file that contains all the details and management, whilst other files contain the details and behaviours of the components that make up the BOMBE. The rotors and reflector are as they are in the Enigma. The files have not changed. The diagonal board is also an additional file that will be used to create the menu as well as test the BOMBE for stops. Once a stop has been found the details are input into a frequency analysis file, that will then validate the stop is correct, and then pass this information back to the main file. The main file will then output the correct settings. The only input that is needed by the main file is the ciphertext and guessed crib. This will then be passed to the diagonal board for a menu to be created. This is a function inside the diagonal board file. Once a menu has been made, this will be passed to the diagonal board set up function. This will then create a computer model of what we see in Figure Five. This is the testing board for each of the settings output by the model of the Enigma, run from the main file. If a stop is detected, then these settings will be passed to the frequency analysis file, along with the crib and ciphertext. This will then run and output the settings used in the plug-board, then all settings will be output by the main file. At this point, we have the settings for all components that were used by the Enigma to encrypt the ciphertext. Any other messages sent in the same network as this message, which will use the same run of settings, will be easily decrypted as the settings are already known.

The ”parallelising” of the BOMBE was multistage. First using the brute force approach basis to allow for simple, single line commands. Multiple Instruction Multiple Dataset and Single Instruction Multiple Dataset techniques can applied without anything more advance being used and the rest being left to the compiler. This resulted in a very substantial increase in performance. The Multiple Instruction Multiple Dataset technique split the workload across cores and this resulted in an almost 3 times performance increase from the non-parallel version. This is due to the lack of data dependencies, and lack of overhead in setting up the file, a data dependency would mean that multiple processes are attempting to access a specific piece of data at the same time, resulting in a queue forming to access that piece of data, meaning time wasted. The data that each of the cores was using, was stored in the cache associated with each. The only thing that changed per core were the variables that store the settings currently being tested, and the output that would be tested against the known ciphertext value. This was set to be immutable, so that it could be treated as read only memory for each of the cores resulting in none of the processes waiting for the use of the data. This was a very effective technique. The use of Single Instruction Multiple Dataset technique is less effective. This is the use of vectorised loads from memory, meaning that more than a single value can be retrieved from memory at a single time. This process is not significantly data dependent. Most of the process is reading data that cannot be vectorised, and even if it could be, it would not be useful as the data load is specific to one letter from one data source, then another letter from another data source. This continues until the next letter is input. This process, whilst having significant improvements from the original, takes such a long time that this approach is still slower than the time taken by the exact BOMBE recreation. The next stage of ”parallelising” the BOMBE is the application of parallel techniques to the exact BOMBE recreation. This was based on the lesson learnt from the application of parallel techniques to the brute force BOMBE - that MIMD is much more effective than SIMD. This technique is applied to the settings search, by splitting the search space between all cores available. This means that there is an N times speed up, where N is the number of cores available. This means that there is still a fair amount of overhead - the frequency analysis cannot be sped up. The result of this is still however a marked improvement.

The testing methodology is identical to that of the Enigma. Each file was tested individually for consistency with expected behaviour, then joined to make the whole. The rotor and reflector files are identical to the ones used in the Enigma, which were previously tested. The diagonal board was tested through an incremental test and each of the 325 connections was tested for the expected ”live” wires. Once this was done, the creation of the menu was tested. An exemplar ciphertext and crib were input, and the result was tested against a manually made menu. This was done multiple times with different pairings. The behaviour, whilst testing each of the rotors, was also tested, but this required the BOMBE to be run from the main file as a whole. This was done at that stage. The frequency analysis was tested by taking multiple known examples and running them to make sure the result was correct. This is a deterministic process so the result isn't always perfect. This was a problem that was encountered in the original BOMBE also. Once each of the components were tested, the BOMBE was built. This was then tested against random known encodings from the Enigma machine. The results were then compared to the plaintext and, due to the deterministic nature of the frequency analysis, a rough result was that one letter may differ in a 10 letter crib. This was considered enough to make out the word or message that was sent. If the message was of critical nature, such that a single letter being wrong was going to cause an issue, then the frequency analysis could be done manually as it was done originally. This was done once in the testing process to prove it could be done.

Again an agile approach was adopted for this project. Each component is not dependent on another, such that modularity and independent development was possible. This also allowed for independent testing, which was a key asset in the confirmation of the correctness of the BOMBE. Besides just the output being correct, it was also shown to be correct throughout the process.

IV RESULTS

The testing criterion that were followed was to test a twenty letter ciphertext with a 20 letter crib across both the exact recreation and parallel versions. This is then compared to the brute force time taken and the historic BOMBE. The testing basis followed was to use random 20 letter ciphertexts encrypted via the Enigma and the corresponding plaintext as the crib. This would ensure that both are known and controlled. This will minimise issues that occur with the frequency analysis that is done at the end of the decryption. This analysis can introduce issues, such as letters with similar frequency being replaced incorrectly. These were not taken into account in the testing.

The time taken for the historic version of the BOMBE to decrypt a twenty letter ciphertext is not known, and there are very few resources that held any information on the decryption time of the BOMBE so it will have be worked out mathematically. The top rotors of the BOMBE rotated at 120rpm in later versons, and this is the figure we will be using in our calculations. There are 26 positions of a rotor. If there are three rotors, this is 17576 unique positions of a three rotor set. There are 56 different combinations of rotors that can be used. We know that there are three banks of twelve of these sets of three, meaning a total of 36 different combinations. If we remove the time taken for swapping out the rotors as well, then this means that three different settings can be tested simultaneously, given that the ciphertext is 12 letters long. A test of three of the 56 settings will take 17576 divided by 120 seconds. This means roughly 150 seconds will then be multiplied by 19 to allow all combinations to be tested, along with some extra time for swapping over the rotors between tests. This comes to 2850 seconds or 47.5 minutes. This only takes into account the time taken to break the rotors and reflector, there is no mention of the plugging up of the BOMBE, or the time taken to apply the frequency analysis after the settings have been found. These are harder to calculate, given that we do not know how many people worked on any given machine, and that the number of people and techniques used in the frequency analysis is also not known. So, we give a rough estimate of the time taken for an Enigma machine to be fully decrypted as being in a number of hours.

The brute force attack was not feasible to run. This was due simply to the number of settings that would have to be tested. The result was worked out mathematically using the theory explained in the creation of the brute force version of the BOMBE. Given that an estimate for the number of operations that a processor can do per second, now resides in the billions, a figure of 5 billion operations per second was used to give 3.23×10^{38} years to test every setting of the Enigma or 6.12×10^{47} seconds.

Each test was run ten times, the averages of these are shown. The results of each of the tests differed based on when the correct settings were tested. For example, if the correct setting was the first to be tested, then the results would be available almost instantly, whereas if the correct settings were tested last, then this would take a lot longer. This was overcome through the use of random settings, the knowledge of the time taken by each implementation to exhaust all possible settings as well as the time taken if the correct settings were the first to be tested. This is shown in Table 4.

Version	20 letters (Seconds)	40 letters (Seconds)
Historic	14400	14400
Brute Force	6.12×10^{47}	12.24×10^{47}
Recreation	48.22	67.75
Parallel	17.62	26.67

Table 3: Results

Version	20 letters (Seconds)	Correct Setting First (Seconds)	Correct Setting Last (Seconds)
Recreation	48.22	4.06	85.32
Parallel	17.62	4.08	26.81

Table 4: Deviations

The time taken to decrypt double the length of ciphertext using the historic BOMBE, would not take much longer than decrypting half the length. There would just be more on which to validate the result. This means that the time taken has not been altered, as the same calculation could be done on the first 20 characters, then the second 20 would be used to validate the settings that have been output by the breaking of the second 20.

V EVALUATION

The strengths of the project lie mainly in the simplicity of the parallelism applied. The BOMBE is inherently very complex, and this has been shown in the amount of background focus that this paper has gone into on the topic. However, the speed up has been very simple to apply and measure. The completeness of the BOMBE and Enigma can also not be overlooked. The thorough testing that went into each constituent component proved that each behaves as it should, and the testing that was done on both of the final systems verified this. The simplicity of the application of the tools used in the parallelisation of the BOMBE is also a strength of the projects choice of tools. The Intel tool set was used effectively and without hassle. It is simple enough to use with ease, whilst also containing the functional might to apply advance parallel concepts if required. The management of the project was also done very well, the use of an agile development strategy with individual separate components was simple and easy to use, meaning that the project progressed in a well established way. Continuous testing was able to occur without changes having to be made at a later date when more development was done on the system.

To evaluate the results, the deviations that occurred between tests should be analysed. This shows that the order in which the settings are tested produces quite different results. The order in which the settings were tested in both of the more advanced BOMBE recreations was constant, the settings that they were testing however, were not however. This resulted in time deviations between each of the tests run, an average of these were taken, but the range cannot be ignored.

The maximum time taken and the minimum time taken are shown in Table 4. These are the extremes, with the average time taken with random settings being roughly halfway between these two values. The minimum value is above zero as the time taken to apply frequency analysis in both cases must be performed, even if the setting first tested is the correct one, this must be validated.

Parallel programming relies on the fact that multiple processors are available to separate the work. In this instance, a limitation was found with the time available for testing so only N = 4 or four separate cores were tested.

The weaknesses of the system mainly revolve around the organisation of the development, and the lack of time for testing that this resulted in. Poor time management and external factors resulted in an expedited work flow that meant less due diligence could be done than would have been optimal. Given more time, it would have been advantageous to explore more avenues with regards to parallel techniques. There are many more than are available than the Intel tool set, so a significant amount of time would have been required to develop the system to use this other tool set. An example of a possible candidate is OpenMP.

If this project were to be repeated, an area that would be improved upon is the amount of time available to test each component. This was not done to exhaustion and was done through random testing and edge cases that result in a reasonable expectation of performance. This is a standard technique for systems that cannot be tested with every possible scenario, and instead revert to test cases to make sure that the most likely settings are tested, and shown to work as well as the more unlikely settings. This thoroughly verify the extremes of the settings that the system can expect. One other focus point that could be developed upon is the documentation side. This was underestimated and thus resulted in time limited writing of some of the documentation.

VI CONCLUSION

The main findings of this project are that the developments that have been achieved in the last near century are significant. The time taken for a full decryption of the Enigma has dropped from being measured in hours to being measured in single figure seconds. This is then compounded by further development of the BOMBE to incorporate parallel techniques, such that the decryption time is almost halved from the unparalleled version. Improvement can always take place and this is how the project could be extended. There are many developments that could take place on the parallel techniques used, and there are also more developments that could be done on the formulation of the unparalleled BOMBE to allow for more eliminations to be achieved in the search space.

References

Anna, B. & Elzbieta, R. (2014), 'The Cryptanalysis of the Enigma Cipher. The Plugboard and the Cryptologic Bomb', *Computer Science* **15**(4), 365.
URL: *https://journals.agh.edu.pl/csci/article/view/1175*

Enigma – Britannica Academic (n.d.).
URL: *http://academic.eb.com.ezphost.dur.ac.uk/levels/collegiate/article/32677*

Lin, Y. C. & Snyder, L. (2009), *Principles of parallel programming*, Pearson/Addison Wesley.

Navy M3/M4 Enigma Machine Emulator (n.d.).
URL: *http://enigma.louisedade.co.uk/enigma.html*

Pacheco, P. S. (2011), *An introduction to parallel programming*, Morgan Kaufmann.
URL: *http://discover.durham.ac.uk/primo_library/libweb/action/display.do?tabs=detailsTab&ct=displa programming&dum=true&dstmp=1503599723862*

Stoler, M. S. (2007), ‘Re-engineering the Enigma Cipher’.

The Turing Bombe - Cribs and Menus (n.d.*a*).
URL: *http://www.ellsbury.com/%5C/bombe1.htm*

The Turing Bombe - Cribs and Menus (n.d.*b*).
URL: *http://www.ellsbury.com/%5C/bombe1.htm*