Nicholas Vlahos-Sten

Jeff Ondich

Computer Security

6 November 2025

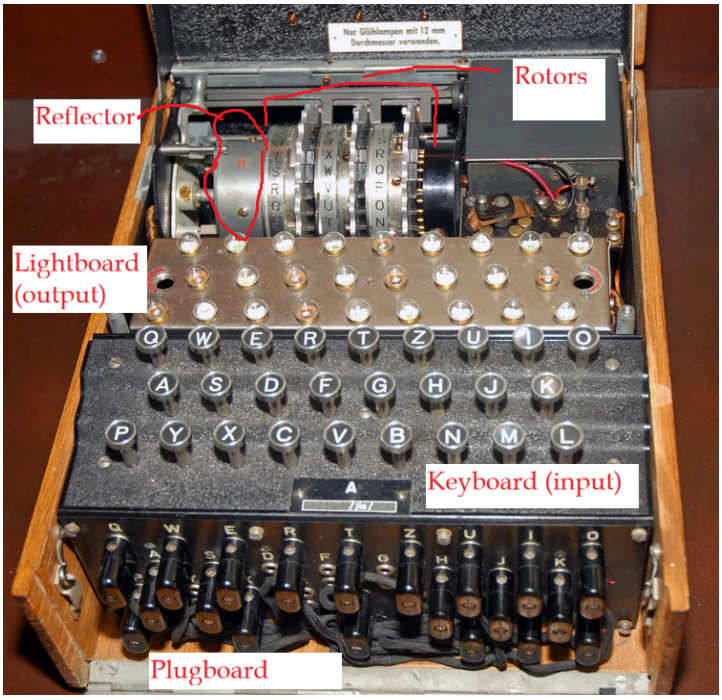
Enigma Machine Cryptanalysis
by Nicholas Vlahos-Sten

Imagine your classic cryptography scenario: Alice is trying to send a message to Bob, but Eve is listening. How can Alice send a secret message to Bob without Eve understanding it?

Now imagine it's 1940. Alice is a Nazi commander. She fires up the radio, and uses Morse code to send an encrypted message to Bob, a u-boat commander. You are Eve, a British cryptanalysist who has intercepted a Morse code message between Alice and Bob, only to find an unintelligible string of seemingly random letters. The message is encrypted, and if you could decrypt it, you could reveal orders to attack or maneuver. With those, you could intercept Bob and foil their plans. The fate of the free world lies on you. How do you decrypt it?

Part 1: How the Enigma Machine Worked

First, we have to understand how the Nazis encrypted their radio messages. Each radio station and submarine was equipped with an encrypting device called an **Enigma machine**:



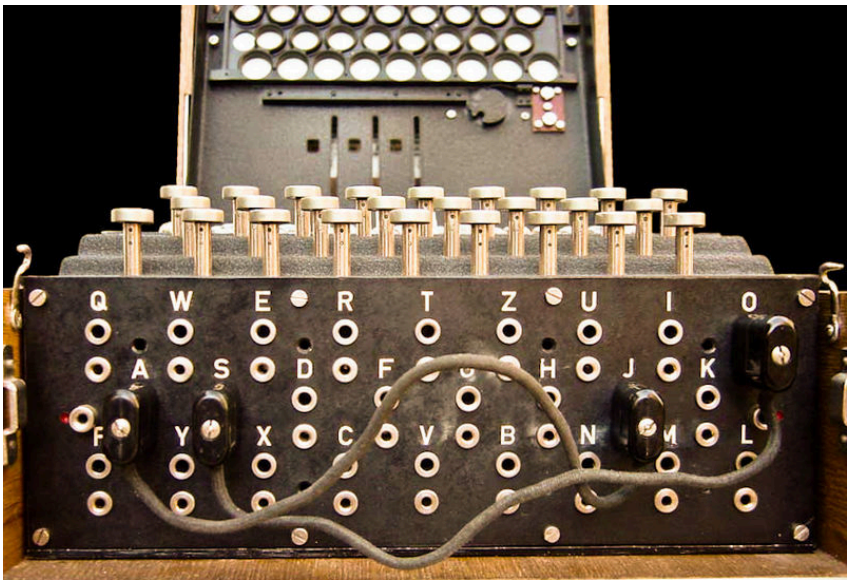
Enigma encrypts letters one at a time. To start encrypting your plaintext message, press its first letter on the **keyboard**. The **lightboard** would then light up, showing you what the first letter of the ciphertext message should be. Jot that down, then press your message's second letter on the keyboard, note its corresponding ciphertext on the lightboard, so on and so forth.

Once you have an encrypted message, send it as Morse code. The Enigma machine serves as both an encrypting and decrypting device. Any receiving party that has an Enigma machine set to the **same initial settings** (the "**key**") as you can type the encrypted message onto the keyboard and get the decrypted message on the lightboard.

The Nazis used the same key for every Enigma machine, and they would change this key every single day. To agree upon the key, a codebook was distributed to each station with an Enigma machine every month. The codebook contained the keys to be used every day of the specified month.

Again, the Enigma machine inputs a plaintext letter and outputs a different ciphertext letter. Let's take a look inside, at how this happens, by tracing the journey of the plaintext letter. What does the machine do to it to turn it into a different ciphertext letter?

Step 1: Plugboard:



After you typed a letter on the keyboard, the inputted letter would first go through a **plugboard**, where it might be transformed into another letter. As part of the **key**, the operator would insert cables into the plugboard connecting pairs of letters. In the above example, the operator has connected **A and J** and **O and S**.

If the inputted letter is in a pair, the plugboard outputs the other member of the pair. So if you press **A,** the plugboard will output **J.** It's reciprocal, so if you press **J,** the plugboard will output **A.**

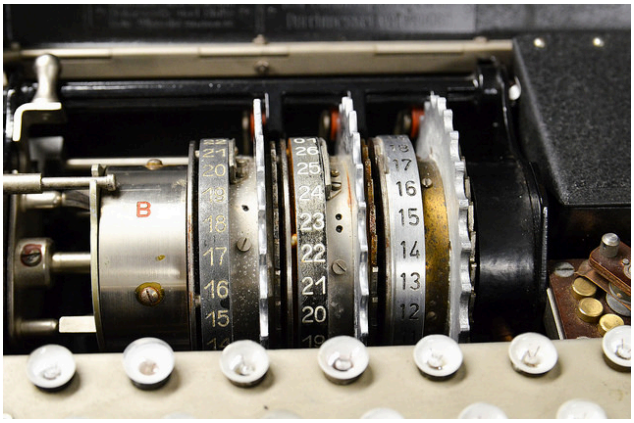
If the letter is not in a pair (IE there's not an external cable inserted into it on the plugboard), it outputs itself. So if you press **E,** the plugboard will output **E.**

The German word for plugboard was steckerbrett. If the plugboard connected A and J, then British cryptanalysts would say "A is **steckered** to J."

This is the first part of the key, a list of which letters are steckered to which other letters. It was standard Nazi practice to use 10 cables for every key, creating 10 letter pairs and leaving 6 letters steckered to themselves. When choosing 10 cables, the amount of possible ways to arrange the plugboard is:

$$\frac{26!}{(6! \cdot 10! \cdot 2^{10})} = 150{,}738{,}274{,}937{,}250$$

Step 2: Rotors



Rotors in an Enigma machine. Note the letter will go through them **right to left**.

After going through the plugboard, the (possibly) transformed letter would then pass through three successive **rotors.** The Enigma machine came with **five possible rotors** to insert, and each rotor hosted a unique substitution cipher. Let's take a look at a rotor.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	A	I	B	R	C	J

This table shows the substitution cipher for Rotor I (the rotors were simply labeled I through V). The top row is the inputted letter and the bottom is the outputted letter. So if you put an **A** into rotor 1, you'd output an **E.** Input **B,** output **K.**

The five rotors were distinguished by unique letter orderings in the bottom row. For comparison's sake, here's Rotor II:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	J	D	K	S	I	R	U	X	B	L	H	W	T	M	C	Q	G	Z	N	P	Y	F	V	O	E

As you can see, Rotor II features a different ordering of letters in the second row.

Of course, when you're encrypting a letter in the Enigma machine, you pass through multiple rotors. So if an **A** is passing through Rotor I and Rotor II, it first turns into an **E** at Rotor I, then that **E** turns into an **S** at Rotor II.

Complexity Update!

A **key** now consists of the **10 plugboard pairs**, plus the **choice of rotors**. The latter is a probability case of "5 options, pick 3, order matters" which can be thought of as "you have 5 choices for the first rotor, 4 for the second, and 3 for the third." Thus you have $5 \cdot 4 \cdot 3 = 60$ choices for rotor order. Adding on to our plugboard possibilities gives you the following key space

$$(5 \cdot 4 \cdot 3) \cdot \frac{26!}{(6! \cdot 10! \cdot 2^{10})}$$ = 9.0442965+e15 possible keys

Now here's the tricky part. The rotors could, well, **rotate**. This is the reason why these simple ciphers are encased within cog wheels. **Rotating** a rotor made it so that every letter in its bottom row shifted right one place. For example, take Rotor I:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	A	I	B	R	C	J

→

Rotating Rotor I once shifts the bottom row one spot over to create this:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
J	E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	A	I	B	R	C

→

As you can see, A, B, C and D originally coded to E, K, M and F. Now, after rotation, they code to J, E, K and M.

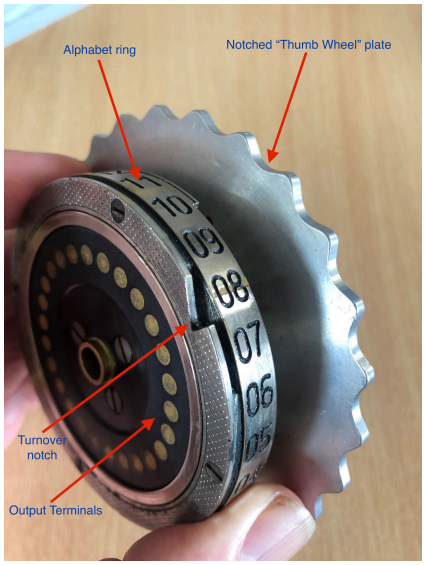
With rotations, each rotor now has **26 starting positions** based on what the bottom sequence's starting letter E codes to. In position 1, it codes to A, in position 2 it codes to B, position 3 C, so on and so forth. You can see the possibilities this adds:

Complexity Update!
(Spoiler: this is final complexity update!)

A **key** now consists of the **10 plugboard pairs**, plus the **choice of rotors** plus **the starting positions for each rotor**. You have 26 different options for each of 3 independent rotors, and $26 \cdot 26 \cdot 26 = 17,576$ choices for rotor starting positions. Adding on to our plugboard and rotor choice possibilities gives you the following key space:

$$(26^{3}) \cdot (5 \cdot 4 \cdot 3) \cdot \frac{26!}{(6! \cdot 10! \cdot 2^{10})} = 1.5896256\text{+e20 possible keys}$$

Now here's a cool feature of the Enigma machine. Once you start encrypting a series of letters, the **rotors start rotating on their own**. After you type one letter of plaintext, a mechanical mechanism in the machine will **rotate the first rotor the letters go through (the rightmost one) one position**.



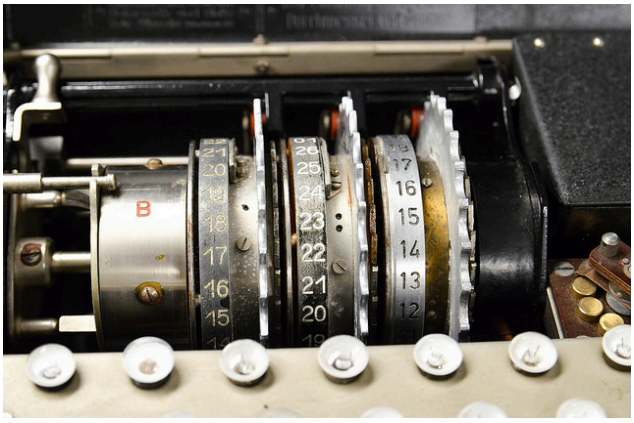
The second and third rotors rotate too, but far less frequently. Each rotor is equipped with a **notch** next to one of its outputted letters. You can see an example of this notch on the image on the left. When that specific letter, say Y, hits the first position (IE when an inputted A outputs Y), the notch will hit the adjacent rotor and force it to move one position.

What this means is that the first rotor will rotate one position with every letter pressed. Then the second rotor will rotate one position once during the first 26 inputted letters (when exactly is based on where the first rotor's notch is in the rotor's chosen starting position) and then rotate once after every 26 subsequent letter presses.

Similarly, the third rotor will rotate one position once during the first 676 inputted letters (again based on where the notches in the first two rotors start off) and then rotate once after every 676 additional letter presses. So it will take 17,576 typed letters for the rotors to loop back to their starting positions (making finding this position pretty difficult).

What all of this auto-rotation nonsense actually means is that **typing the same letter of plaintext twice will output different letters of ciphertext.** That is the main security reason for this automatic rotation.

Part 3: the reflector.



When you saw this image originally, you might have wondered what that metal circle marked “B” was. Or not. In either way, I’ll explain. That’s the **reflector**.

The **reflector** can be thought of as **yet another substitution cipher**, albeit one that doesn’t rotate. I believe in later versions of the Enigma machine you could reconfigure the reflector, but in earlier versions the reflector was prebuilt and locked in. Thus, I am **not** going to consider reflector configuration to be an element of an **enigma key**.

Here is the substitution cipher for the basic UKW-A reflector.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
E	J	M	Z	A	L	Y	X	V	B	W	F	C	R	Q	U	O	N	T	S	P	I	K	H	G	D

Notice how, unlike the rotors, **no letter maps to itself.** This is **REALLY REALLY IMPORTANT.**

So this next part is complicated. After going through the three rotors, the inputted letter would go through the reflector. Then whatever letter the reflector spits out would go through the **three rotors AGAIN,** except it goes through them **backwards** and in **reverse.**

What does that mean? Take Rotor I. When I go through it regularly, I use the top row for inputs and the bottom row for outputs. So if I input an **A** I output **E.**

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	**U**	V	W	X	Y	Z
E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	**A**	I	B	R	C	J

But when I go through the rotor **backwards,** I use the **bottom row for inputs and the top row for outputs.** So if I input an **A**, I output **U** (relevant columns bolded).

By **reverse,** I mean it goes through the rotors in the order of 3, 2, 1 (left to right) instead of 1, 2, 3 (right to left).

This reversal is why the reflector maps no letter to itself. If it mapped a letter to itself, all the Enigma machine would be doing is going through three rotors, then immediately reversing the process, and the resulting ciphertext would just be the plaintext. And knowing what letters the reflector encrypts to itself would make it much easier to figure out the key. Oops!

Anyway, why do this? Well, this is best showcased again by my upcoming example encryption (as why this works is not immediately obvious), but doing this reflector nonsense means that **the Enigma machine is both an encryptor and decryptor.** If you have two Enigma machines with the same starting key, every letter of ciphertext will “encrypt” to the corresponding letter of plaintext. So radio operators could take the encrypted message, type it into the machine as if they were themselves encrypting a message, and come out with the decrypted message.

After this process, the letter goes through the plugboard one last time, and then we are done! In total the route a plaintext letter takes through the Enigma machine is this (with R standing for Rotor and * specifying going through a Rotor backwards):

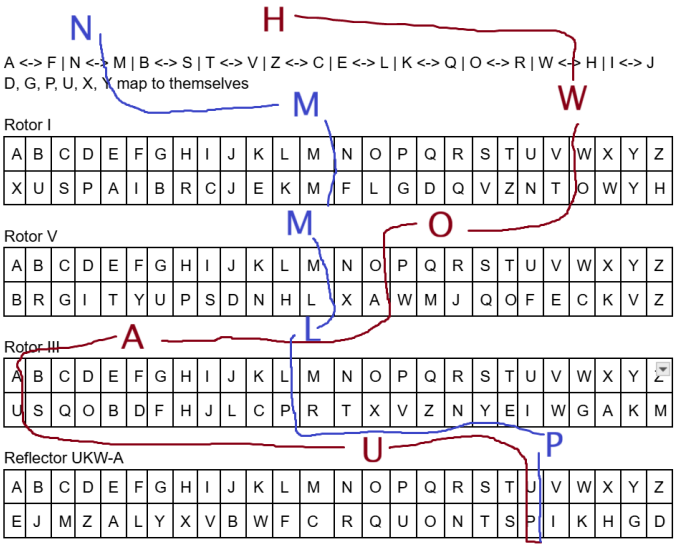
Input -> Plugboard -> R1 -> R2 -> R3 -> Reflector -> R3* -> R2* -> R1* -> Plugboard -> Output

Example Encrypting!

Okay, I am now going to walk through encrypting the message “HIJEFF” letter by letter. I am using the following key:

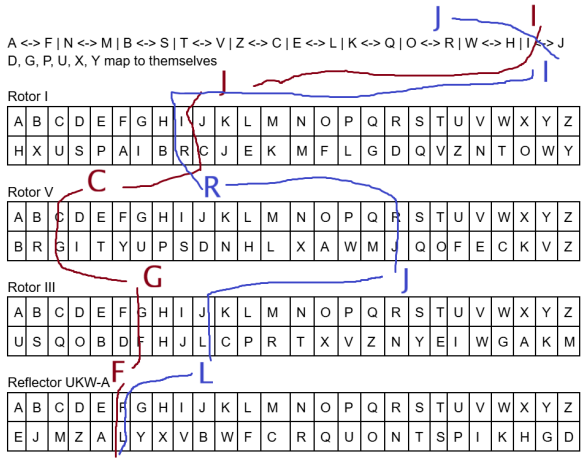
Rotors, in order: I, V, III
(importantly, these Rotors have notches on the letters Y, H and D respectively)
Rotor initial positions: 11, 24, 5
Plugboard:
A <-> F | N <-> M | B <-> S | T <-> V | Z <-> C | E <-> L | K <-> Q | O <-> R | W <-> H | I <-> J
D, G, P, U, X, Y map to themselves

Here we go. To follow the path the letter H takes, start at H and **follow the red lines down and the blue lines up.**

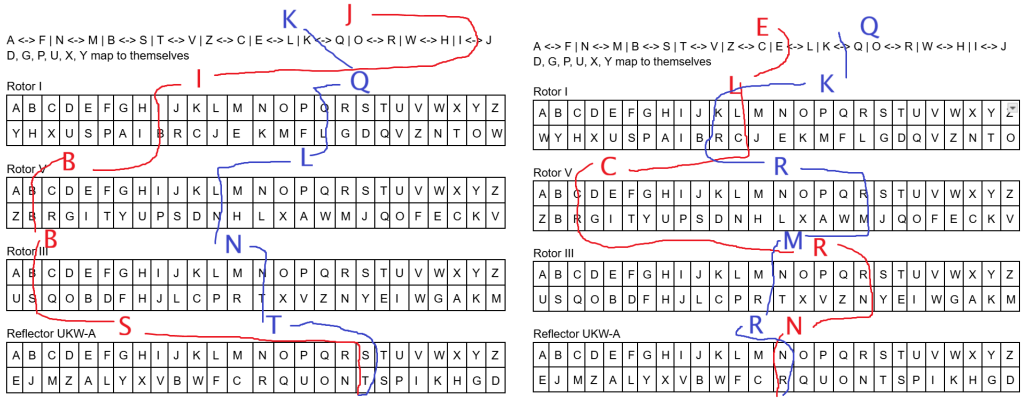


Now this image is the best possible showcase for why encrypting the ciphertext again gets you the plaintext. Say you receive my key and the letter N. What would happen if you put N into the machine? Well look at the image again, but this time start at N and **follow the blue lines down and the red lines up.** Spoiler, you get H.

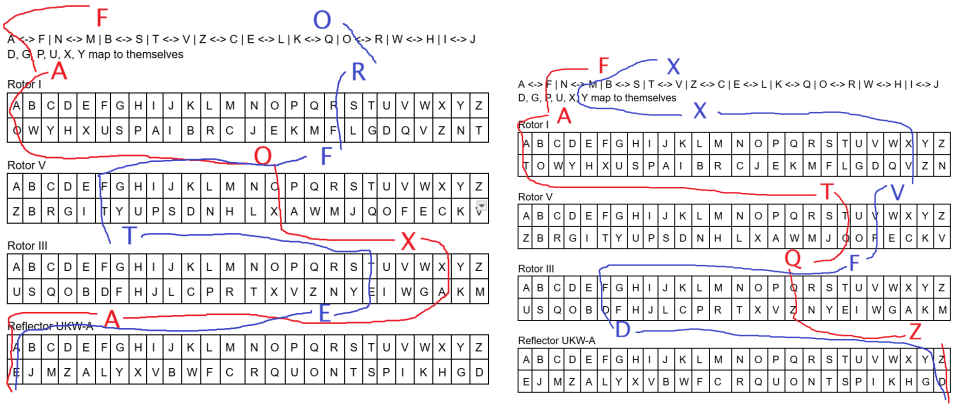
Anyway, for our next letter, we rotate the first rotor 1 position and do the process again.



Now we rotate once more. But when we rotate the next time, on Rotor I, A will encode to Y. The notch on Rotor I happens to be next to Y, which means that **adjacent Rotor V will also rotate once**. Anyway, let's get through a few of these.



Finally, for the last two letters, notice how the rotation of Rotor I makes it so that two consecutive Fs code to different letters.



So there we go. The plaintext HIJEFF encodes to the ciphertext NJKQOX

Part 2: how to crack the enigma code

The Enigma Machine's Achilles Heel

Remember the really really important "feature" of the Enigma machine? That no letter can encrypt to itself? It turns out that this impossibility is a major vulnerability, and is really really bad for Alice and Bob. It forms the basis for Eve's primary attack. British cryptanalysts, led by Alan Turing, intercepted encrypted messages at the secret code-breaking center Bletchley Park and employed a **probable phrase attack** to figure out what the key was.

Here's how a probable phrase attack works.

Step 1: Find a Crib

First, pick an encrypted message, and choose a probable phrase, or a **sequence of plaintext you think the plaintext message contains**. This phrase is called a **crib**. The main crib pop culture likes to flaunt is "Heil Hitler." The story goes that German radio operators had to end

every message with “Heil Hitler,” so Eve always knew the last 10 letters were ciphertext for HEILHITLER. No reliable source I researched mentions this phrase as a crib, however, which leads me to believe the story is probably apocryphal (though I see why people want it to be true, it’s quite ironic). Rather, the cribs Eve actually used were more boring. Here’s some popular ones:

- WETTER VORHESAGE – translates to “WEATHER REPORT” There were many weather forecasts sent over radio, and unfortunately for the Germans, they were standardized
- KEINE BESONDEREN EREIGNEISSE – “NOTHING TO REPORT”
- Many messages started with ANX (TO and a space; many operators pressed X on the Enigma machine to represent a space) followed by the name of whoever they were sending the message to. So if you knew who Alice and Bob were, this is a great crib.
- If Britain badly needed to figure out a daily key (like if they were planning an offensive), they would sometimes plant mines in a random waterway and wait for a German minesweeping ship to report via radio that they were mines in the location (so your crib would be something likes “mines at location”). This actually sometimes worked.

Now it sounds ridiculously hard to guess that a specific phrase is in a specific message, and it was, but it was made slightly easier by the fact that all messages used the same key each day. So you could, for example, specifically pick a boring report from a base far away from any action, and make KEINE BESONDEREN EREIGNEISSE your crib. Then, if you could solve the key on that message, you could decrypt every other message sent that day.

Anyway, it’s not enough to know that KEINE BESONDEREN EREIGNEISSE is in a message– you need to know **exactly where** in the message it is. This step was actually way easier than you’d think because of the Enigma machine’s Achilles’ Heel– no letter encrypted to itself.

Let me give an example: Let’s say I had the ciphertext DLGAEFGURXRWS and I thought part of it coded to A L L C L E A R. Well, let's just try lining ALL CLEAR up to the cipher text to see what letters the plaintext encrypted to. it could logically fit there. I will guess that A codes to the first D…

A	**L**	L	C	L	E	A	R					
D	**L**	G	A	E	F	G	U	R	X	R	W	S

Well, there we go! ALLCLEAR is not there! Because if the first 8 letters of the message was ALLCLEAR, the first L would encrypt to itself! That’s impossible! So let’s shift it over and try a different place ALLCLEAR could be

	A	L	L	C	L	E	A	**R**				
D	L	G	A	E	F	G	U	**R**	X	R	W	S

Nope! The R encodes to itself! That's not possible. Let's try again.

		A	L	L	C	L	E	A	R			
D	L	G	A	E	F	G	U	R	X	R	W	S

There we go! ALLCLEAR could possibly translate to GAEFGURX.

Obviously, you can have false positives. The longer the crib the less false positives there would be.

Step 2: Crib Menus and Stecker Hypothesises

Once you have a crib and a corresponding piece of ciphertext, you need to figure out what key turned the crib into the ciphertext. The way you do this is unfortunately brute force: you have to try a ridiculous number of keys to test not only whether the crib turns into the ciphertext, but also if the rest of the message produces readable German.

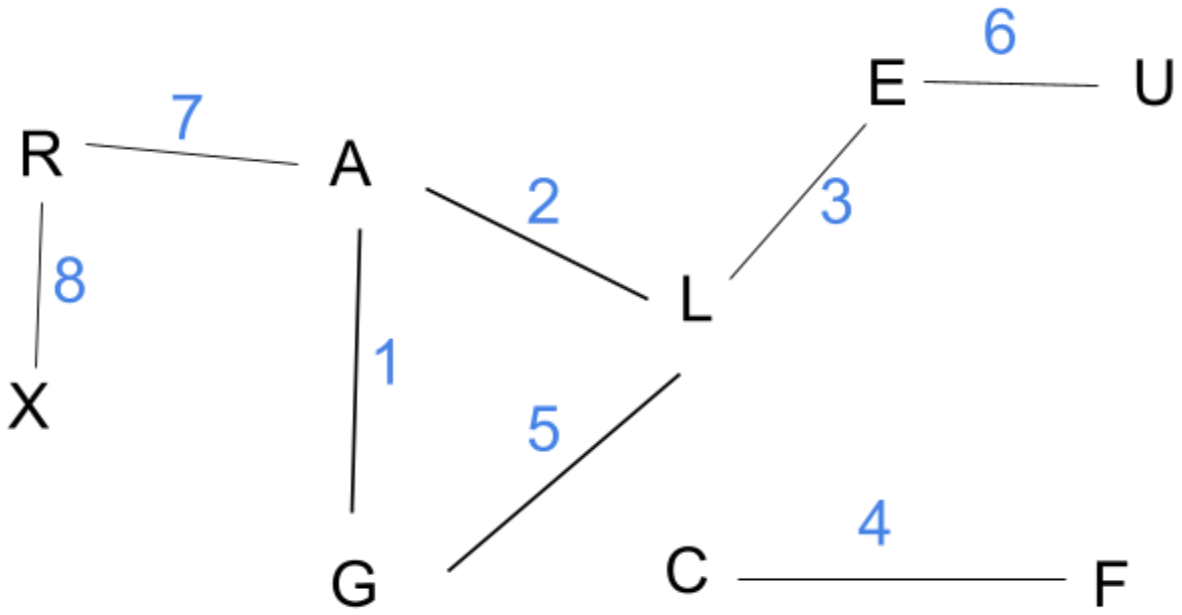
Now, it's the 1940s. There is no way for a machine to tell you if a message is readable German. You had to check that by hand. With the amount of possible keys, it was not possible to do this all by hand.

But there was a more efficient way to reduce the number of messages you had to check by hand, by running each possible key through only a few specific letters in the crib to see if they logically could produce the ciphertext. You exploited the fact that the plugboard never changed.

First align your crib and corresponding ciphertext and label each position:

1	2	3	4	5	6	7	8
A	L	L	C	L	E	A	R
G	A	E	F	G	U	R	X

Remember that the Enigma machine is self-reflexive. At position 1, not only does A encode to G, but G encodes to A. With this fact in mind, we are going to draw a diagram, showing at what position each letter maps to each other letter. This diagram is called a **menu.**



The A –1–G here indicates that at position 1, A and G code to each other.

Notice there is a **loop** between A L and G. At one point A codes to G, G codes to L and L codes to A.

Now the brute force begins. We are going to guess a key. This loop lets us figure out if our guess is correct **in such a way that a machine can do**. To do this, we are actually going to try and prove our guessed key is not correct. We will use a proof via contradiction.

We know that A is connected to one of 26 letters on the plugboard. We are going to pick one at random. Let's guess that A is steckered to D. We now know the following:

- A is steckered to D
- L is steckered to some letter LET_1
- G is steckered to some letter LET_2

Now, let's ignore the plugboards for a minute. We know that, at position 2, D goes through the rotors and gets transformed into LET_1. Since, at position 2, A and L code to each other we can make the following diagram of what the Engima machine does at position 2.

A -> plugboard -> D -> rotors -> LET_1 -> plugboard -> L

It inputs A, which is steckered to D, which goes through the rotors to produce LET_1, which is steckered to L.

Now note the last bit. **L is steckered to LET_1.** Now let's look at what happens at position 5…

L -> plugboard -> LET_1 -> rotors -> LET_2 -> plugboard -> G

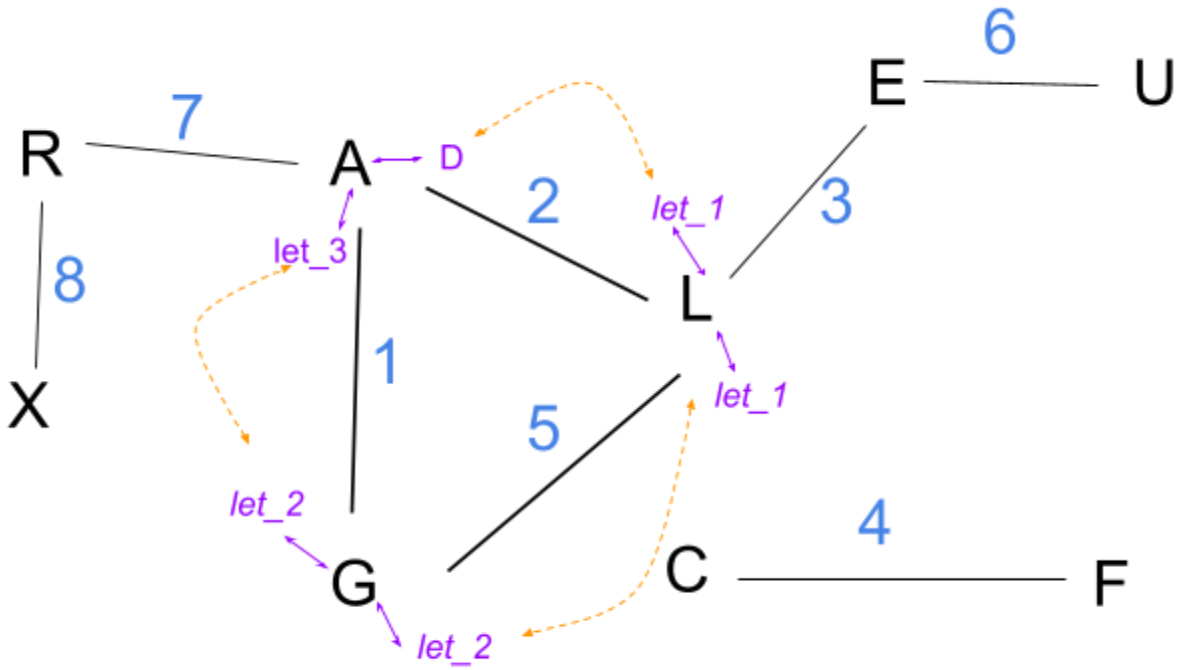
…and finally position 1…

G -> plugboard -> LET_2 -> rotors -> LET_3 -> plugboard -> A

Now, if our guess was correct, LET_3 has to be D, because A can only be steckered to one letter. We of course have a way to test this. Using your guessed key:

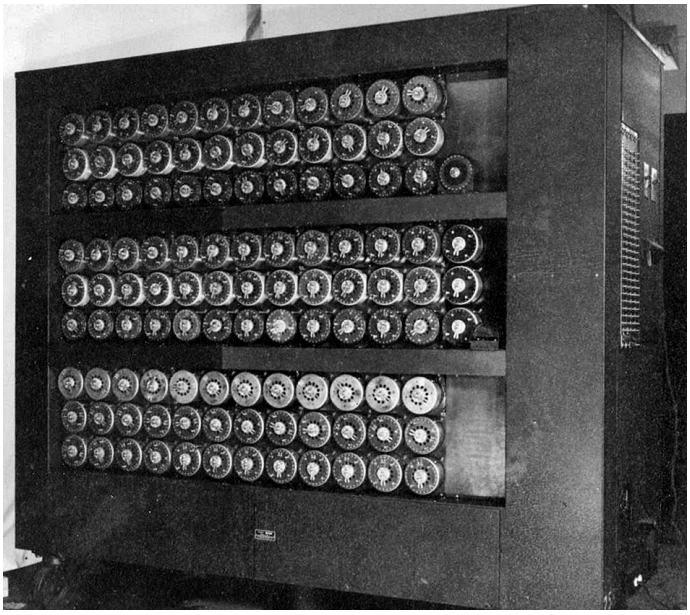
- Make the letter D go through only the rotors/reflector in position 2 to find LET_1
- Make LET_1 go through the rotors in position 5 to find LET_2
- Make LET_2 go through the rotors in position 1 to find LET_3
- If LET_3 is not D, then proof via contradiction: **your guessed key is wrong**
- But if LET_3 is D, then **your guessed key could be correct** (could also be a false stop)

Here's a diagram of this, the purple arrows representing stecker pairs, and the orange arrows representing trips through the rotors (which are also self-reflexive).



Let_3 must equal D. If it doesn't, our key is wrong!

Now this is a completely logical operation that does not require recognizing German words. A machine can do this. While it couldn't read German, it could take a menu, a guessed key, and tell you if, based on the key, if the menu was logically consistent.



A bombe. Each column of three wheels mimics the three rotors in an Enigma machine.

Alan Turing and his team built a machine called a **bombe**, which inputted these menus and outputted a list of key settings that could logically work. A human would then use an Enigma machine to check these **stops**, inputting the possibly correct settings, typing in the encrypted message, and manually checking if it resembled actual German. The more loops in the menu, the longer the crib, the fewer false positives the bombe would spit out.

Here ends my exploration of the Enigma code. I was originally planning to further explore how the bombe worked (as well as Gordon Welchman's **diagonal board**, an improvement to the bombe that reduced the number of false positives it spat out) but I could not quite make heads and tails of most of the bombe explanations I found, which focused on how it worked electrically and mechanically. Further, while I understand how the bombe could logically ascertain if a guessed key was a possible solution, I am still confused as to how it was able to try so many guessed keys in a reasonable time.

Sources Cited

https://enigma.virtualcolossus.co.uk/technical.html
https://www.youtube.com/watch?v=ybkkiGtJmkM
https://www.youtube.com/watch?v=9u7bjZfgEvo
http://www.ellsbury.com/enigmabombe.htm
https://www.tnmoc.org/bh-16-menus-and-cribs
https://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma
https://hackaday.com/2017/08/22/the-enigma-enigma-how-the-enigma-machine-worked/
https://www.cryptomuseum.com/crypto/enigma/working.htm
https://cryptii.com/pipes/enigma-machine
https://www.theguardian.com/technology/2014/nov/14/how-did-enigma-machine-work-imitation-game

Images Used

https://jgandrews.com/posts/the-enigma-machine/
https://www.mub.eps.manchester.ac.uk/science-engineering/2018/11/28/cracking-stuff-how-turing-beat-the-enigma/
https://hackaday.com/2017/08/22/the-enigma-enigma-how-the-enigma-machine-worked/
https://www.tnmoc.org/bh-2-the-enigma-machine
https://en.wikipedia.org/wiki/Bombe