Enigma

Kiyoshi Akima
http://kiyoshiakima.tripod.com/funprogs

2006.04.27

Contents

1 Rotor-Based Cipher Machine

World War I brought a new dimension to cryptography and cryptanalysis. The traditional paper-and-pencil cryptographic systems, the classical cipher systems and simple code systems, had become targets of opportunity for Allied cryptanalysts. Even the ADFGVX Cipher, then believed by its users to offer the ultimate in security, by war's end offered only token resistance to the shillful efforts of French, British, and American cryptanalysts. By the time the war ended in 1918, Allied code and cipher experts ahd become confident that they could handle almost any type of system they might encounter.

After the war, however, there began a new era for cryptographic ideas. Inventors began thinking about new encryption methods. The U.S. Patent Office began processing patent applications for anew cipher devices and machines. The most important invention was a new electro-mechanical enciphering and deciphering machine. With this invention emerged the concept of the electrical *rotor* or "transfer wheel," which until the late 1960s has held a prominent and important, if not guarded, place in the cryptographic community. (Incidentally, it is not certain where or when the word "rotor" first came into usage; and it is curious, too, that the word is a palindrome.)

Rotor machines appear to have been independently, and almost simultaneously, invented in four different countries, in the U.S. by E. H. Hebern, in Sweden by A. Damm, in Holland by H. Koch, and in Germany by A. Scheribius. Koch later worked with Scheribius in Berlin where together they produced the now infamous Enigma ciper machine of World War II.

A rotor is simply a flattened drum made of an insulating material. On each of the rotor's two faces, 26 electrical contacts protrude. Intermally, the contacts from one face are connected randomly to those on the other face. In the typical rotor machine, a number of rotors, usually three to five, are placed side-by-side, so that electrically the rotors have contact with each other. When a key is pressed on a keyboard, an electrical current leaves the key, passes through each of the rotors, in turn, and finally exits, causing a bulb (lamp) to light, or, by means of some form of printing mechanism, to print a letter. Before the next letter is enciphered, "stepping gears" usually cause one or more of the rotors (which contain teeth on their peripheries) to rotate or step. Cryptographic security in a rotor-type cipher machine is thus due to the maze of electrical connections between the keyboard and the indicating device, which changes as letters are enchiphered. While previous machines often used 26 different enciphering alphabets in some pseudorandom manner, the new rotor-type machines were capable of producing as many as 26^n different enciphering alphabets with n rotors.

So important was the concept of the rotor that during World War II most of the major powers used machines incorporating it. For example, Germany had the Enigma, Britain the Typex, and U.S. the SIGABA (M-134).

2 The Enigma

Like other rotor machines, the Enigma machine is a combination of mechanical and electrical systems. The mechanical mechanism consists of a keyboard; a set of rotating disks called *rotors* arranged adjacently along a spindle; and a stepping mechanism to turn one or more of the rotors with each key press. The exact mechanism varies, but the most common form is for the right-hand rotor to step once with every keystroke, and occasionally the motion of neighboring rotors is triggered. The continual movement of the rotors results in a different cryptographic transformation after each key press.

2.1 Machine Components

The mechanical parts act in such a way as to form a varying electrical circuit—the actual encipherment of a letter is performed electrically. When a key is pressed, the circuit is completed; current flows through the various components and ultimately lights one of many lamps, indicating the output letter. For example, when encrypting a message starting `ANX...`, the operator would first press the `A` key, and the `Z` lamp might light; `Z` would be the first letter of the ciphertext. The operator would then proceed to encipher `N` in the same fashion, and so on.

2.1.1 Rotors

The rotors (alternatively, *wheels* or *drums*—*Walzen* in German) form the heart of the Enigma machine. Approximately four inches in diameter, each rotor is a disk made of hard rubber or bakelite with a series of brass spring-loaded pins on one face arranged in a circle; on the other side are a corresponding number of circular electrical contacts. The pins and contacts represent the alphabet—typically the 26 letters `A-Z` (this will be assumed for the rest of the document). When placed side by side, the pins of one rotor rest against the contacts of the neighboring rotor, forming an electrical connection. Inside the body of the rotor, a set of 26 wires connects each pin on one side to a contact on the other in a complex pattern. The wiring differs for every rotor.

By itself, a rotor performs only a very simple type of encryption—a simple substitution cipher. For example, the pin corresponding to the letter `E` might be wired to the contact for letter `T` on the opposite face. The complexity comes from the use of several rotors in series—usually three or four—and the regular movement of the rotors; this provides a much stronger type of encryption.

When placed in the machine, a rotor can be set to one of 26 positions. It can be turned by hand using a grooved finger-wheel which protrudes from the internal cover when closed. So that the operator knows the position, each rotor has a *alphabet ring* attached around the outside of the disk, with 26 letters or numbers; one of these can be seen through a window, indicating the position of the rotor to the operator. The position of the ring is known as the *Ringstellung* ("ring settings"). In the military versions, the ring contains a notch used to control the stepping of the rotors.

The Army and Air Force Enigmas came equipped with several rotors; when first issued there were a total of three. In 1938 this changed to five, from which three were chosen for insertion in the machine. These were marked with Roman numerals to distinguish them: `I`, `II`, `III`, `IV`, and `V`, all with single notches. The Navy version had always been issued with more rotors than the other services: at first, five, then seven and finally eight. The additional rotors were named `VI`, `VII`, and `VIII`, all with different wiring, and had two notches cut into them, resulting in a more frequent turnover.

The four-rotor Navy Enigma (M4) accommodated an extra rotor in the same space as the three-rotor version. This was accomplished by replacing the original reflector with a thinner reflector and adding a special fourth rotor. The fourth rotor can be one of two types: Beta or Gamma. This fourth rotor never steps, but can be manually placed in any of the 26 positions.

2.1.2 Stepping Motion

To avoid merely implementing a simple substitution cipher, some rotors turn with consecutive presses of a key. This ensures that the cryptographic transformation is different at each position, producing a formidable *polyalphabetic substitution* cipher.

The most common arrangement utilizes a ratchet and pawl mechanism. Each rotor is affixed with a ratched with 26 teeth; a group of pawls engage engage the teeth of the ratchet. The pawls are pushed forward in unison with each keypress on the machine. If a pawl engages the teeth of a ratchet, that rotor advances by one step.

In the Wehrmacht Enigma, each rotor is affixed with an adjustable notched ring. At a certain point, a rotor's notch will align with the pawl, allowing it to engage the ratchet of the next rotor with the subsequent keypress. When a pawl is not aligned with the notch, it will simply slide over the surface of the ring without engaging the ratchet. In a single-notch rotor system, the second rotor is advanced one position every 26 advances of the first rotor. Similarly, the third rotor is advanced one position for every 26 advances of the second rotor. The second rotor also advances at the same time as the third rotor, meaning the second rotor can step twice on subsequent key presses—"double-stepping"—resulting in a reduced period.

A double step occurs as follows: the first rotor steps, and takes the second rotor one step further. If the second rotor has moved by this step into its own notch position, the third pawl can drop down. On the next step this pawl pushes the ratchet of the third rotor and advances it, but will also push into the second rotor's notch, advancing the second rotor a second time in row.

With three wheels and one notch on each wheel, the machine has a period of $26 \times 25 \times 26 = 16\,900$. Historically, messages were limited to a couple of hundred letters, and so there was no risk of repeating any position within a single message.

When pressing a key, the rotors step before the electrical circuit is connected.

2.1.3 Reflector

The last rotor is followed by a *reflector* (*Umkehrwalze* in German), a patented feature distinctive of the Enigma family among the various rotor machines designed in the period. The reflector connects outputs of the last rotor in pairs, redirecting current back through the rotors by a different route. The reflector ensures that Enigma is *self-reciprocal:* conveniently, encryption is the same as decryption. However, the reflector also gives Enigma the property that no letter can encrypt to itself. This was a severe conceptual flaw and a cryptological mistake subsequently exploited by codebreakers.

In most models of the Enigma, the reflector is fixed and does not rotate.

2.1.4 Plugboard

The *plugboard* (*Steckerbrett in German*) is a variable wiring that could be reconfigured by the operator. It was introduced on German Army versions in 1930 and was soon adopted by the Navy as well. The plugboard contributes a great deal to the strength of the machine's encryption, more than an extra rotor would. Enigma without a plugboard—"unsteckered" Enigma—can be solved relatively straightforwardly using hand methods; these techniques are generally defeated by the addition of a plugboard, and codebreakers resorted to special machines to solve it.

A cable placed onto the plugboard connects letters in pairs, for example, `E` and `Q` might be a steckered pair. The effect is to swap those letters before and after the main rotor scrambling unit. For example, when an operator presses `E`, the signal is diverted to `Q` before entering the rotors. Several such steckered pairs, up to 13, might be used at one time.

Current flows from the keyboard through the plugboard, and proceeds to the entry stator or *Eintrittswalze.* Each letter on the plugboard has two jacks. Inserting a plug will disconnect the upper jack (from the keyboard) adn the lower jack (to the entry stator) of that letter. The plug at the other end of the crosswired cable is inserted into another letter's jacks, switching the connections of the two letters.

2.2 Basic Operational Procedures

In German military usage, communications were divided up into a number of different networks, all using different settings for their Enigma machines. These communications nets were termed *keys* at Bletchley Park and were assigned codenames such as *Red*, *Chaffinch*, and *Shark*. Each unit operating on a network was assigned a settings list specifying the Enigma for a period of time. For a message to be correctly encrypted and decrypted, both sender and receiver have to set up their Enigmas in the same way; the rotor selection and order, the starting position plugboard connections need to be identical; these settings have to be agreed on beforehand and were distributed in codebooks.

An Enigma machine's initial state, the *cryptographic key*, has several aspects:

- **Wheel order** ***(Walzenlage)***—the choice of rotors and the order in which they are used.
- **Initial position of the rotors**—chosen by the operator, different for each message.
- **Ring settings** ***(Ringstellung)***—the position of the alphabet ring relative to the rotor wiring.
- **Plug settings** ***(Steckerverbindungen)***—the connections of the plugs in the plugboard.

Enigma was designed to be secure even if the rotor wiring was known to an eavesdropper, although in practice the wiring was kept secret. With secret wiring, the total number of possible configurations has been calculated to be around 10^{114} (approximately 380 bits); with known wiring and other operational constraints, this is reduced to around 10^{23} (76 bits). Users of Enigma were assured of its security by the large number of possibilities; it was not feasible for an adversary to even begin to try every possible combination in a brute force attack.

2.2.1 Indicators

Most of the key were kept constant for a set time period, typically a day. However, a diffrent initial rotor position was chosen for each message, because if a number of messages are sent encrypted with identical or near identical settings, a cryptanalyst has several messages "in depth," and might be able to attack the messages using frequency analysis. To counter this, a different starting position for the rotors was chosen for each message; a concept similar to an initialization vector in modern cryptography. The starting position was transmitted along with the ciphertext. The exact method used is termed the "indicator procedure"—weak indicator procedures allowed the initial breaks into Enigma.

One of the earliest indicator procedures was exploited to make the initial break into the Enigma by Polish cryptanalysts. The procedure was for the operator to set up his machine in accordance with his settings list, which included a global initial position for the rotors (*Grundstellung*—"ground setting"), `AOH`, say. The operator would turn his rotors until `AOH` was visible through the rotor windows. At this point, the operator would choose his own, arbitrary starting position for that particular message. An operator might select `EIN`, and this became the *message settings* for that encryption session. The operator would type `EIN` into the machine, twice, to allow for detecting transmission errors. The results would be an encrypted indicator—the `EIN` typed twice might turn into `XHTLOA`, which would be transmitted along with the message. Finally, the operator would then spin the rotors to his message settings, `EIN` in this example, and the text of the actual message was typed in.

At the receiving end the operation was reversed. The operator set the machine to the initial settings and typed in the first six letters of the message (`XHTLOA`). In this example `EINEIN` would be produced. By moving his rotors to `EIN`, the receiving operator would then type in the rest of the ciphertext, deciphering the message.

The weakness came from two factors: the use of a global ground setting—this was later changed so that the operator selected his initial position to encrypt the indicator, and sent the initial position in the clear. The second problem was the repetition of the indicator, which was actually a security flaw. The message key was encoded twice, resulting in a relation between first and fourth, second and fifth, and third and sixth characters. This security problem enabled the Polish Cipher Bureau to break the pre-war Enigma messages. However, from 1939 on, the Germans changed the procedure to increase the security, transmitting the encrypted indicator only once.

During the Second World War, German operators used the codebooks only to set up the rotors and ring settings and to make the plugboard connections. For each message, he selected a random start position, let's say `WZA`, and random message key, let's say `SXT`. He moved the rotors in the `WZA` start position, and encoded the message key `SXT`. Let us assume that the result was `UHL`. He sets up the message key `SXL` as start position, and encode the message. Next, he transmits the start position `WZA`, the encoded message key `UHL` together with the message. The receiver sets up the start position according to the first trigram, `WZA`, and decodes the second trigram, `UHL`, to obtain the `SXT` message key. Next, he uses this `SXT` message key as start position to decode the message. This way, each ground setting was different and the new procedure avoided the security flaw of double encoded message keys.

This procedure was used by Army and Air Force only. The Navy procedures on sending messages with the Enigma were far more complex and elaborate.

2.2.2 Abbreviations and Guidelines

The Army Enigma machine only used the 26 alphabet characters. Signs were replaced by rare character combinations. A space was omitted or replaced by an `X`. The `X` was generally used as point or full stop. Some signs were different in other parts of the armed forces. The Army replaced a comma by `ZZ` and the question mark by `FRAGE` or `FRAQ`. The Navy however, replaced the comma by `Y` and the question mark by `UD`. The combination `CH`, as in *Acht* (eight) or *Richtung* (direction) was replaced by `Q` (`AQT`, `RIQTUNG`). Two, three, or four zeros were replaced by `CENTA`, `MILLE`, and `MYRIA`.

The Army and Air Force transmitted the messages in groups of five characters. The Navy, using the four rotor Enigma, applied four letter groups. Frequently used names or words were to be varied as much as possible. Words like *Minensuchboot* (minesweeper) could be written as `MINENSUCHBOOT`, `MINBOOT`, `MMMBOOT`, or `MMM354`. To make cryptanalysis harder, more than 250 characters in one message were forbidden. Longer messages were divided in several parts, each using its own message key.

2.3 Variants

Far from being a single design, there are numerous models and variants of the Enigma family. The earliest Enigma machines were commercial models dating from the early 1920s. Starting in the mid-20s, the various branches of the German military began to use Enigma, making a number of changes in order to increase its security. In addition, a number of other nations either adopted or adapted the Enigma design for their own cipher machines.

The Enigma model A was exhibited at the Congress of the International Postal Union in 1924 and 1924. The machine was heavy and bulky, incorporating a typewriter. A model B was introduced, and was of a similar construction. While bearing the Enigma name, both models A and B were quite unlike later versions; they differed in physical size and shape, but also cryptographically, in that they lacked the reflector.

The reflector was first introduced in the Enigma C (1926) model. The reflector is a key feature of all subsequent Enigma machines.

The German Army introduced their own version of the Enigma in 1928. The major difference from the commercial Enigma models was the addition of a plugboard to swap pairs of letters, greatly increasing the cryptographic strength of the machine. Other differences included the use of a fixed reflector, and the relocation of the stepping notches from the rotor body to the movable alphabet rings.

A four rotor Enigma was introduced by the Navy for U-boat traffic in 1942. The extra rotor was fitted in the same space by splitting the reflector into a combination of a thin reflector and a thin fourth rotor. This thin rotor did not rotate with the other rotors, but it could be set in any of 26 positions. In one of these positions, the four-rotor Enigma enciphered exactly the same way as the three-rotor Enigma (wow, *déjà vu*, emulating a three-rotor machine on something else).

2.4 Breaking the Enigma

Enigma was designed to defeat basic cryptanalysis techniques by continually changing the substitution alphabet. Like other rotor machines, it implemented a *polyalphabetic substitution* cipher with a long period. With single-notched rotors, the period of the machine was $16,900$ ($26 \times 25 \times 26$). This long period helped protect against overlapping alphabets.

The Enigma machines added other possibilities. The sequence of alphabets used was different if the rotors were started in position `ABC`, as opposed to `ACB`; each rotor had a rotatable ring which could be set in different positions, and the starting position of each rotor was also variable. Most of the military Enigmas also featured a plugboard (German: *Steckerbrett*) which exchanged letters. Even so, this complex combination key could be easily communicated to another user, comprising as it did only a few simple items: rotors to be used and their order, ring positions, starting positions, and plugboard connections. Potentially this made the Enigma an excellent system.

2.4.1 Security Properties

The various Enigma models provided different levels of security. The presence of a plugboard substantially increased the complexity of the machine. In general, unsteckered Enigma could be attacked using hand methods, while breaking versions with a plugboard was more involved, and often required the use of machines.

The Enigma machine had a number of properties that proved helpful to cryptanalysts. First, a letter could never be encrypted to itself (with the exception of the early models which lacked a reflector). This was of great help in finding *cribs*—short sections of plaintext that are known (or suspected) to be somewhere in a ciphertext. This property can be used to help deduce where the crib occurs. For a possible location, if any letter in the crib matches a letter in the ciphertext at the same position, the location can be ruled out; at Bletchley Park, this was termed a "crash."

Another property of the Enigma was that it was *self-reciprocal:* encryption is performed identically to decryption. This imposed constraints on the type of scrambling that Enigma could provide at each position, and this property was used in a number of codebreaking methods.

A weakness of many Enigma models was that the rightmost rotor turned a constant number of places before the next rotor turned.

Apart from the less-than-ideal inherent characteristics of the machine, the way Enigma was used proved its greatest weakness in practice. Mistakes by operators were common, and a number of the officially-specified procedures for using Enigma provided avenues for attack. It has been suggested by some of those working on its cryptanalysis at Bletchley Park that the Enigma would have been unbreakable in practice had its operators not been so error-prone, and had its operating procedures been better thought out.

2.4.2 Solution Before World War II

In December 1932, a 27-year-old Polish mathematician, Marian Rejewski, who had joined the Polish Cipher Bureau in September that year, made one of the most important breakthroughs in cryptologic history by using algebraic mathematical techniques to solve the Enigma wiring.

At the time, the indicator procedure was to encrypt an operator-selected message setting twice, with the machine at its "ground setting," and to place the twice-encrypted message setting at the opening of the message. For instance, if an operator picked `QRS` as his 'message setting,' he would set the machine to the day's ground settings, and then type `QRSQRS`. This might be encrypted as `JXDRFT`. The feature of Enigma that Rejewski exploited was that the disk moved three positions between the two sets of `QRS`—knowing that `J` and `R` were originally the same letter, as were `XF` and `DT`, was vital information. Although the original letters were unknown, it was known that, while there were a huge number of rotor settings, there were only a small number of rotor wirings that would change a letter from `J` to `R`, `X` to `F` and `D` to `T`, and so on. Rejewski called these patterns *chains*.

However, in 1939 the German Army increased the complexity of its Enigma operating procedures. Initially only three rotors had been in use, and their sequence in the slots was changed periodically. Now two additional rotors were introduced; three of the five would be in use at any given time. The Germans also stopped transmitting a twice-enciphered individual three-letter message setting at the beginning of a message, thus putting an end to one of the Poles' original methods of cryptological attack.

2.4.3 World War II

British codebreakers at Bletchley Park had adopted the Polish Enigma-breaking techniques, but had to remain alert to German cryptographic advances. The German Army had changed its practices (more rotors, a more secure indicator system, etc.). The German Navy—some of whose Enigma ciphers the Poles had broken—had always used more secure procedures.

German Army and Air Force Enigma-machine operators also gave the decrypters immense help on a number of occasions. In one instance an operator was asked to send a test message, and simply hit the `T` key repeatedly and sent the resulting letters. A British analyst received from the intercept stations a long message without a single `T` in it, and immediately realized what had happened. In other cases, Enigma operators would constantly use the same settings as message keys, often their own initials or those of girlfriends (called "cillies," after an operator with the apparent initials "C.I.L."). Analysts were set to finding these messages in the sea of intercepts every day, allowing Bletchley Park to use the original Polish techniques to find the initial settings for the day. Other German operators used "form letters" for daily reports, notably weather reports, in which case the same crib might be used every day.

Later in the war, British codebreakers learned to fully exploit a crucial security flaw associated with German weather reports: they were broadcast from weatherships to Germany in lower-level ciphers, easy to decrypt, then retransmitted to U-boats at sea in Enigma, thus giving Bletchley Park regular cribs. This was crucial in attacking the special four-rotor U-boat Enigma machine introduced in 1942.

Cipher material was captured at sea. The first capture of Enigma material occurred in February 1940, when rotors VI and VII, the wiring of which was at that time unknown, were captured from the crew of U-33. On May 7, 1941, the Royal Navy captured a German weather ship, together with cipher equipment and codes. They did it again shortly afterwards. And two days later U-boat U-110 was captured, complete with Enigma machine, codebook, operating manual and other information.

And then there was the *bombe*, a precursor of the modern computer. Space precludes a discussion of this electro-mechanical marvel: Books have been written on the subject so I won't go into it here.

3 The Emulator

Real Enigma machines are rare and thus command premium prices. But we can emulate one on a computer. For many cryptographic systems, emulating one can be better than having the real thing:

- **Availability.** Many devices are not available, due either to low production numbers or government restrictions.
- **Cost.** Except for the initial cost of the computer, which in most likelihood was purchased for other purposes, no additional expenses are necessary, whereas purchase of individual cryptographic devices (assuming that one could even find them for sale) would continually add up. Prices for computers are going down, while the cost for cryptographic devices, if they can be found, are already high, and climbing.
- **Ease of Operation.** Entering or changing keys in a program is relatively simple when compared to some cryptographic devices. Though some devices can be simulated by using sliding strips, their use often requires careful attention to the relative motion of the alphabets (sliding strips), thus introducing a large margin of error.
- **Ease of Modification.** When using a computer, simple program changes often may be done in minutes, whereas changes in complex wiring or gearing systems might take days and even weeks to accomplish. Thus, using a computer allows the "inventor" to see immediately the results arising from modifications. In addition, new or radically revised systems can often be designed in a very short time, and the resulting cryptographic security can be tested quickly. Without a computer, how long would it take to rewire a rotor? Or to implement an Enigma with five rotors instead of three or four?

3.1 Program Structure

This BCPL program has a very simple structure. It GETs the library header, defines some static variables and a terminal handling module (Section 6), and defines the function `start`.

10 ⟨* 10⟩≡

```
// Enigma

GET "libhdr"

STATIC {
    ⟨static variables 12c⟩
}

⟨terminal handling module 23a⟩

⟨function start 11a⟩
```

3.2 The `start` Function

As usual, execution begins with a call to `start`. For the emulator, execution naturally falls into four main phases:

- Initialize the program.
- Get the key settings from the user.
- Encipher/decipher text using the key.
- Clean up.

11a ⟨*function* `start` 11a⟩≡ (10)

```
LET start() = VALOF {
      ⟨subroutines in start 12b⟩
      ⟨variables in start 16b⟩

      ⟨initialize the program 11b⟩
      ⟨get key settings 12a⟩
      ⟨encipher text 18a⟩
      ⟨clean up 11c⟩

      RESULTIS 0
}
```

3.2.1 Initialization

11b ⟨*initialize the program* 11b⟩≡ (11a)

```
rdch()
```

3.2.2 Termination

When the program is finished, it clears the screen.

11c ⟨*clean up* 11c⟩≡ (11a) 13a ▷

```
TermClearScreen()
```

4 Getting the Key

Before the Enigma can be used, it must be set up with the desired encoding key. The key consists of:

- Rotor order *(Walzenlage)*
- Ring settings *(Ringstellung)*
- Plugboard connections *(Steckerverbindungen)*
- Starting rotor positions

The program clears the screen before and after the key entry phase.

12a ⟨*get key settings* 12a⟩≡ (11a)
```
  TermClearScreen()
  TermMoveCursor(10, 0)
  ⟨get key components 12d⟩
  TermClearScreen()
```

4.1 Prompting for Input

12b ⟨*subroutines in* `start` 12b⟩≡ (11a) 13f▷
```
  LET Prompt(prompt) = VALOF {
      LET c, i = ?, 1
      sawritef("%s: ", prompt)
      {
          c := rdch()
          SWITCHON c INTO {
          CASE 9:
              IF 1 < i i := i = 1
              ENDCASE
          CASE '*n':
              BREAK
          DEFAULT:
              inbuff!i, i := c, i + 1
              ENDCASE
          }
      } REPEAT
      RESULTIS i - 1
  }
```

This routine uses an input buffer…

12c ⟨*static variables* 12c⟩≡ (10) 13b▷
```
  inbuff
```

…which needs to be allocated before use…

12d ⟨*get key components* 12d⟩≡ (12a) 13c▷
```
  inbuff := getvec(30)
```

. . . and deallocated. The program doesn't deallocate it immediately after the key entry phase since it will be needed again to reposition the rotors.

13a ⟨*clean up* 11c⟩+≡ (11a) ◁11c 14d▷
```
  freevec(inbuff)
```

4.2 Rotor Order

The first component of the key is the rotor order *(Walzenlage)*. Historically the rotors were identified by Roman numerals (and Greek letters for the fourth "thin" rotors). This program uses the Arabic digits 1-8 for the Roman numerals `I-VIII` and the lowercase Roman letters `b` and `g` for the Greek letters `B` and Γ. They were usually specified from left to right, and this program adheres to that tradition.

4.2.1 Rotor Configuration

The program emulates the four-rotor Enigma if the user specifies four rotors, otherwise it emulates the three-rotor Enigma.

13b ⟨*static variables* 12c⟩+≡ (10) ◁12c 13d▷
```
  fourth = 0
```

13c ⟨*get key components* 12d⟩+≡ (12a) ◁12d 13e▷
```
  IF 4 = Prompt("Rotor Order") fourth := 1
```

The program stores the rotor configurations in a vector.

13d ⟨*static variables* 12c⟩+≡ (10) ◁13b 15e▷
```
  rotors
```

13e ⟨*get key components* 12d⟩+≡ (12a) ◁13c 15d▷
```
  rotors := getvec(3 + fourth)
  FOR i = 1 TO 3 rotors!i := ConfigRotor(inbuff!(i+fourth))
  IF fourth      rotors!4 := ConfigRotor(inbuff!1)
```

13f ⟨*subroutines in* `start` 12b⟩+≡ (11a) ◁12b 14e▷
```
  LET ConfigRotor(r) = VALOF {
      LET v = getvec(3)
      ⟨configure rotor 13g⟩
      RESULTIS v
  }
```

Rotor configuration consists of three components:

- The rotor forward translation table.
- The rotor reverse translation table.
- The notch position(s).

The rotor information is packaged in a pair of functions.

13g ⟨*configure rotor* 13g⟩≡ (13f) 14a▷
```
  LET w, n = RotorWiring(r), RotorNotches(r)
```

The forward translation table is used to map the current flowing through the rotor from right to left.

14a ⟨*configure rotor* 13g⟩+≡ (13f) ◁13g 14b▷

```
v!1 := getvec(26)
FOR i = 1 TO 26 v!1!i := w%i - 'A'
```

The reverse translation table is used to map the current flowing through the rotor from left to right.

14b ⟨*configure rotor* 13g⟩+≡ (13f) ◁14a 14c▷

```
v!2 := getvec(26)
FOR i = 1 TO 26 v!2!(w%i - 'A' + 1) := i - 1
```

The notch position(s) affect the motion of the rotors.

14c ⟨*configure rotor* 13g⟩+≡ (13f) ◁14b

```
v!3, v!3!1 := getvec(3), n%0
FOR i = 1 TO n%0 v!3!(i+1) := n%i - 'A'
```

All of the memory allocated for the rotors need to be deallocated at program termination.

14d ⟨*clean up* 11c⟩+≡ (11a) ◁13a 15f▷

```
FOR i = 1 TO 3 + fourth {
    freevec(rotors!i!1)
    freevec(rotors!i!2)
    freevec(rotors!i!3)
    freevec(rotors!i)
}
freevec(rotors)
```

4.2.2 Rotor Information

These two functions store the information about the rotors. First the internal wirings.

14e ⟨*subroutines in* `start` 12b⟩+≡ (11a) ◁13f 15c▷

```
AND RotorWiring(r) = VALOF {
    SWITCHON r INTO {
        ⟨rotor wirings 14f⟩
    }
    RESULTIS ""
}
```

Here are the wirings of the five standard rotors used by all branches of the *Wehrmacht*.

14f ⟨*rotor wirings* 14f⟩≡ (14e) 15a▷

```
DEFAULT:    RESULTIS "EKMFLGDQVZNTOWYHXUSPAIBRCJ"
CASE '2':   RESULTIS "AJDKSIRUXBLHWTMCQGZNPYFVOE"
CASE '3':   RESULTIS "BDFHJLCPRTXVZNYEIWGAKMUSQO"
CASE '4':   RESULTIS "ESOVPZJAYQUIRHXLNFTGKDCMWB"
CASE '5':   RESULTIS "VZBRGITYUPSDNHLXAWMJQOFECK"
```

Here are the wirings of the three additional rotors used by the *Kriegsmarine*.

15a ⟨*rotor wirings* 14f⟩+≡ (14e) ◁14f 15b▷

```
CASE '6':   RESULTIS "JPGVOUMFYQBENHZRDKASXLICTW"
CASE '7':   RESULTIS "NZJHGRCXMYSWBOUFAIVLPEKQDT"
CASE '8':   RESULTIS "FKQHTLXOCBJSPDZRAMEWNIUYGV"
```

And finally here are the wirings of the two "thin" rotors used by the *Kriegsmarine.*

15b ⟨*rotor wirings* 14f⟩+≡ (14e) ◁15a

```
CASE 'b':   RESULTIS "LEYJVCNIXWPBQMDRTAKZGFUHOS"
CASE 'g':   RESULTIS "FSOKANUERHMBTIYCWLQPZXVGJD"
```

The rotor notches are provided the same way.

15c ⟨*subroutines in* `start` 12b⟩+≡ (11a) ◁14e 20c▷

```
AND RotorNotches(r) = VALOF {
    SWITCHON r INTO {
    DEFAULT:    RESULTIS "Q"
    CASE '2':   RESULTIS "E"
    CASE '3':   RESULTIS "V"
    CASE '4':   RESULTIS "J"
    CASE '5':   RESULTIS "Z"
    CASE '6':
    CASE '7':
    CASE '8':   RESULTIS "MZ"
    CASE 'b':
    CASE 'g':   RESULTIS ""
    }
    RESULTIS ""
}
```

4.3 Ring Settings

The second component is the ring settings *(Ringstellung)*.

15d ⟨*get key components* 12d⟩+≡ (12a) ◁13e 16a▷

```
Prompt("Ring Settings")
ring := getvec(4)
IF fourth ring!4 := inbuff!1 - 'a'
FOR i = 1 TO 3 ring!i := inbuff!(i + fourth) - 'a'
```

The program stores the ring settings in a vector. . .

15e ⟨*static variables* 12c⟩+≡ (10) ◁13d 16c▷

```
ring
```

. . . which needs to be deallocated at clean up.

15f ⟨*clean up* 11c⟩+≡ (11a) ◁14d 16e▷

```
freevec(ring)
```

4.4 Plugboard Connections

The third component is the plugboard connections *(Steckerverbindungen)*.

16a ⟨*get key components* 12d⟩+≡ (12a) ◁15d 16d▷
```
  n := Prompt("Plugboard Connections")
```

16b ⟨*variables in* `start` 16b⟩≡ (11a) 18d▷
```
  LET n = ?
```

The program stores the plugboard connections as a vector of integers.

16c ⟨*static variables* 12c⟩+≡ (10) ◁15e 16j▷
```
  plugs
```

This vector needs to be allocated…

16d ⟨*get key components* 12d⟩+≡ (12a) ◁16a 16f▷
```
  plugs := getvec(26)
```

…and deallocated at clean up.

16e ⟨*clean up* 11c⟩+≡ (11a) ◁15f 16k▷
```
  freevec(plugs)
```

First the plugboard is initialized to represent the state with no connections.

16f ⟨*get key components* 12d⟩+≡ (12a) ◁16d 16g▷
```
  FOR i = 1 TO 26 plugs!i := i - 1
```

Then, if at least one connection is specifed, pairs of letters are connected to each other.

16g ⟨*get key components* 12d⟩+≡ (12a) ◁16f 16h▷
```
  UNLESS n < 2 n := n - n MOD 2
  FOR i = 1 TO n BY 2 {
      LET c1, c2 = inbuff!i - 'a', inbuff!(i+1) - 'a'
      plugs!(c1+1), plugs!(c2+1) := c2, c1
  }
```

4.5 Rotor Positions

The final component of the key is the starting positions of the rotors.

16h ⟨*get key components* 12d⟩+≡ (12a) ◁16g 17b▷
```
  posn := getvec(4)
  ⟨get rotor positions 16i⟩
```

16i ⟨*get rotor positions* 16i⟩≡ (16h 19b)
```
  Prompt("Rotor Positions")
  IF fourth posn!4 := inbuff!1 - 'a'
  FOR i = 1 TO 3 posn!i := inbuff!(i + fourth) - 'a'
```

Like the ring settings, the current positions are stored in a vector…

16j ⟨*static variables* 12c⟩+≡ (10) ◁16c 17a▷
```
  posn
```

…which needs to be deallocated at clean up.

16k ⟨*clean up* 11c⟩+≡ (11a) ◁16e 17c▷
```
  freevec(posn)
```

4.6 Reflector

Strictly speaking, the reflector is not part of the key settings. However, the three- and four-rotor Enigmas had different reflectors, and the program must take this into account. With the appropriate "thin" rotor in the correct position, a four-rotor Enigma could be made compatible with the three-rotor machine.

The program stores the reflector mapping in a vector. . .

17a ⟨*static variables* 12c⟩+≡ (10) ◁16j 19e▷

```
  reflector
```

. . . which must be allocated. . .

17b ⟨*get key components* 12d⟩+≡ (12a) ◁16h 17d▷

```
  reflector := getvec(26)
```

. . . and deallocated.

17c ⟨*clean up* 11c⟩+≡ (11a) ◁16k 20b▷

```
  freevec(reflector)
```

Now the program must determine the reflector in use. For both the three- and four-rotor emulations, the program uses the B variant of the reflector.

17d ⟨*get key components* 12d⟩+≡ (12a) ◁17b 17e▷

```
  TEST fourth n := "ENKQAUYWJICOPBLMDXZVFTHRGS"
  ELSE        n := "YRUHQSLDPXNGOKMIEBFZCWVJAT"
```

Then the mappings are converted to integers.

17e ⟨*get key components* 12d⟩+≡ (12a) ◁17d 20a▷

```
  FOR i = 1 TO 26 reflector!i := n%i - 'A'
```

5 Enciphering/Deciphering Text

Now we get to the heart of the program, the enciphering/deciphering loop. This is an infinite loop, broken when the user presses $\boxed{\text{X}}$ to terminate the program.

18a ⟨*encipher text* 18a⟩≡ (11a)

```
{
    ⟨show rotor positions 18b⟩
    ⟨get keystroke 18c⟩
    ⟨process special keys 19a⟩
    IF 'a' <= ch <= 'z' {
        ⟨encipher one letter 19d⟩
    }
} REPEAT
```

5.1 Rotor Positions

On a real Enigma machine, the current rotor positions were visible through windows on the top of the machine. This program emulates that by displaying the rotor positions on the top row of the screen.

18b ⟨*show rotor positions* 18b⟩≡ (18a)

```
TermMoveCursor(0, 27)
sawrch('[')
IF fourth sawrch('A' + posn!4)
FOR i = 1 TO 3 sawrch('A' + posn!i)
sawrch(']')
```

5.2 Input

Reading input is a simple matter of homing the cursor and getting one keystroke.

18c ⟨*get keystroke* 18c⟩≡ (18a)

```
TermMoveCursor(0, 0)
ch := sardch()
```

18d ⟨*variables in* `start` 16b⟩+≡ (11a) ◁16b

```
LET ch = ?
```

5.3 Special Keys

Some keys have special meaning to the program. Note that these are all upper-case letters.

Key	**Meaning**
$\boxed{\text{C}}$	Clear the screen
$\boxed{\text{R}}$	Change rotor positions
$\boxed{\text{X}}$	Exit the program

5.3.1 Clearing the Screen

The $\boxed{\text{C}}$ key clears the plain- and ciphertext strings from the screen. The strings also have to be cleared from memory.

19a ⟨*process special keys* 19a⟩≡ (18a) 19b ▷
```
IF 'C' = ch {
    TermMoveCursor(2, 0);   TermClearEoL()
    TermMoveCursor(3, 0);   TermClearEoL()
    FOR i = 1 TO 61 ptext%i, ctext%i := ' ', ' '
    LOOP
}
```

5.3.2 Changing the Rotor Positions

The $\boxed{\text{R}}$ key allows the user to change the rotor positions. The process is identical to that used for the initial entry of the positions.

19b ⟨*process special keys* 19a⟩+≡ (18a) ◁ 19a 19c ▷
```
IF 'R' = ch {
    TermMoveCursor(10, 0)
    ⟨get rotor positions 16i⟩
    TermMoveCursor(10, 0);  TermClearEoL()
    LOOP
}
```

5.3.3 Exiting the Program

The $\boxed{\text{X}}$ key exits the program.

19c ⟨*process special keys* 19a⟩+≡ (18a) ◁ 19b
```
IF 'X' = ch BREAK
```

5.4 Enciphering One Letter

If the user entered a lowercase letter, we get to the meat of the program; the encipherment/decipherment. First the program adds the letter to the end of the plaintext string and the string is displayed.

19d ⟨*encipher one letter* 19d⟩≡ (18a) 20d ▷
```
AppendLetter(ptext, ch)
TermMoveCursor(2, 0)
sawritef("%s", ptext)
```

The plaintext string, as well as the ciphertext string used for output must be declared.

19e ⟨*static variables* 12c⟩+≡ (10) ◁ 17a
```
ptext; ctext
```

And initialized.

20a ⟨*get key components* 12d⟩+≡ (12a) ◁17e

```
  ptext, ptext%0 := getvec(16), 60
  ctext, ctext%0 := getvec(16), 60
  FOR i = 1 TO 61 ptext%i, ctext%i := ' ', ' '
```

And deallocated at clean up.

20b ⟨*clean up* 11c⟩+≡ (11a) ◁17c

```
  freevec(ptext)
  freevec(ctext)
```

The program displays up to 60 letters in each of the plain- and ciphertext strings. The first letter is dropped from the head of the string and the string shifted to the left before the new letter is added to the tail.

20c ⟨*subroutines in* `start` 12b⟩+≡ (11a) ◁15c 21a▷

```
  LET AppendLetter(s, l) BE {
      FOR i = 1 TO 59 s%i := s%(i+1)
      s%60 := l
  }
```

Next the program converts the letter to an integer.

20d ⟨*encipher one letter* 19d⟩+≡ (18a) ◁19d 20e▷

```
  ch := ch - 'a'
```

5.4.1 Rotor Step

Before any actual encipherment takes place, the rotors step. This alters the substitution so that a letter is not enciphered the same way twice in succession.

The fourth rotor on the four-rotor machine never steps to it can be ignored for the duration of this discussion.

20e ⟨*encipher one letter* 19d⟩+≡ (18a) ◁20d 21b▷

```
  TEST AtNotch(2) ⟨step left rotor 20f⟩
  ELSE            ⟨step middle rotor? 20g⟩
  ⟨step right rotor 20h⟩
```

If the middle rotor was already at its notch position(s), both the left and middle rotors step (the “double-step”).

20f ⟨*step left rotor* 20f⟩≡ (20e)

```
  posn!1, posn!2 := (posn!1 + 1) MOD 26, (posn!2 + 1) MOD 26
```

Otherwise, if the right rotor was at its notch position, the middle rotor steps.

20g ⟨*step middle rotor?* 20g⟩≡ (20e)

```
  IF AtNotch(3)   posn!2 := (posn!2 + 1) MOD 26
```

The right rotor steps on every keystroke.

20h ⟨*step right rotor* 20h⟩≡ (20e)

```
  posn!3 := (posn!3 + 1) MOD 26
```

This function determines whether a rotor is at its notch position(s).

21a ⟨*subroutines in* `start` 12b⟩+≡ (11a) ◃20c 21f▹
```
LET AtNotch(r) = VALOF {
    FOR i = 1 TO rotors!r!3!1 IF posn!r = rotors!r!3!(i+1) RESULTIS 1
    RESULTIS 0
}
```

5.4.2 Plugboard

Now for the encipherment itself. First the letter is passed through the plugboard.

21b ⟨*encipher one letter* 19d⟩+≡ (18a) ◃20e 21d▹
⟨*plugboard translation* 21c⟩

The plugboard translation is a simple table look-up. One has to be added to the index to account for the BCPL vectors beginning with 1.

21c ⟨*plugboard translation* 21c⟩≡ (21b 22c)
```
ch := plugs!(ch+1)
```

5.4.3 Rotors

Then through the standard three rotors, from right to left.

21d ⟨*encipher one letter* 19d⟩+≡ (18a) ◃21b 21e▹
```
FOR i = 3 TO 1 BY -1 ch := Translate(i, 1, ch)
```

And through the fourth rotor, if present, also from right to left.

21e ⟨*encipher one letter* 19d⟩+≡ (18a) ◃21d 21g▹
```
IF fourth ch := Translate(4, 1, ch)
```

This function translates a letter through a rotor. With the reverse mapping generated earlier, this function can handle both the forward and reverse translations. The only difference is the translation table used (or rather, the index of the translation table).

21f ⟨*subroutines in* `start` 12b⟩+≡ (11a) ◃21a
```
LET Translate(r, tt, c) = VALOF {
    LET o = (ring!r - posn!r + 26) MOD 26
    c := (c - o + 26) MOD 26
    c := rotors!r!tt!(c+1)
    c := (c + o) MOD 26
    RESULTIS c
}
```

5.4.4 Reflector

Then through the reflector. This is also a simple table look-up, just like the plugboard.

21g ⟨*encipher one letter* 19d⟩+≡ (18a) ◃21e 22a▹
```
ch := reflector!(ch+1)
```

5.4.5 Rotors

Then back through the fourth rotor, if present, from left to right this time.

22a ⟨*encipher one letter* 19d⟩+≡ (18a) ◁21g 22b▷
```
  IF fourth ch := Translate(4, 2, ch)
```

And back through the standard three rotors, from left to right.

22b ⟨*encipher one letter* 19d⟩+≡ (18a) ◁22a 22c▷
```
  FOR i = 1 TO 3 ch := Translate(i, 2, ch)
```

5.4.6 Plugboard

Finally the letter is passed through the plugboard again.

22c ⟨*encipher one letter* 19d⟩+≡ (18a) ◁22b 22d▷
  ⟨*plugboard translation* 21c⟩

5.4.7 Output

An a real Enigma machine the output is indicated by a glowing lamp. This program does it by adding the letter to the end of the ciphertext string displayed just below the plaintext string.

22d ⟨*encipher one letter* 19d⟩+≡ (18a) ◁22c
```
  AppendLetter(ctext, ch + 'A')
  TermMoveCursor(3, 0)
  sawritef("%s", ctext)
```

6 Handling the Terminal

Due to its interactive nature, this program needs the ability to control the terminal, or at least some aspects of it, namely the ability to position output. A more ambitious program might handle the terminal by determining its type and figuring out how to do various operations appropriately, perhaps by reading `/etc/termcap` and delving into its depths. An even more ambitious program might port the Curses library. This program is not that ambitious. Instead, it has a small set of core functionality for ANSI terminals. They work for such things as VT-100s, console windows under Windows, and xterms.

6.1 Clear the Screen

This procedure clears the screen and homes the cursor. (Actually, the procedure homes the cursor and clears to the end of the display.)

23a ⟨*terminal handling module* 23a⟩≡ (10) 23b ▷
```
  LET TermClearScreen() BE
      sawritef("*e[1;1H*e[2J")
```

6.2 Clear to End-of-Line

This procedure clears the screen from the current cursor position to the end of the line.

23b ⟨*terminal handling module* 23a⟩+≡ (10) ◁ 23a 23c ▷
```
  LET TermClearEoL() BE
      sawritef("*e[K")
```

6.3 Move the Cursor

This procedure moves the cursor to the specified row and column. Rows and columns both start with 0 at the upper-left corner.

23c ⟨*terminal handling module* 23a⟩+≡ (10) ◁ 23b
```
  LET TermMoveCursor(row, col) BE
      sawritef("*e[%i1;%i1H", row + 1, col + 1)
```

7 Running the Emulator

You've seen the program. Now it's time to run it.

7.1 Basic Operation

The program is completely interactive, requiring no arguments on the command line.

The program first clears the screen and then prompts for the rotor order *(Walzenlage)*. Historically the standard rotors were identified by Roman numerals and the "thin" rotors by Greek letters; this program expects three Arabic digits in the range 1-8 and the lowercase Roman letters `b` and `g`. No range checks are performed: an invalid specifier will result in rotor `I` being used. The program also does not ensure that the thin rotors are used only in the leftmost position, nor that only the thin rotors are used in the leftmost position. Nor does the program check to see whether a rotor was requested more than once; real Enigmas were issued with one of each rotor but this program will allow the same rotor to be used more than once. The rotors were traditionally specifed left to right, and this program does not break with that tradition.

The program emulates the four-rotor Enigma if four rotors are specified and emulates the three-rotor Enigma if only three rotors are specified. (Kinda obvious, isn't it?)

The second prompt is for the ring settings *(Ringstellung)*. Depending on the markings on the rings, this was specified either by numbers in the range 1-26 or by letters. On a real Enigma they each were set by lifting a lever and turning the ring. This program expects three or four lowercase letters.

The third prompt is for the plugboard connections *(Steckerverbindungen)*. This was specified as zero or more (commonly ten) pairs of letters. On a real machine the connections were made by inserting double-ended banana plugs into lettered sockets. This program expects a string of lowercase letters without any spacing or punctuation. In the real world one letter cannot be plugged to two others; this program does not check.

The fourth prompt is for the initial position of the rotors. On a real machine they were set by turning thumbwheels on the top of the box. This program expects a string of three or four lowercase letters, like the ring settings

Then the screen is cleared, the current rotor positions are displayed as they are on a real Enigma, and the program is ready to begin enciphering/deciphering text.

Pressing any of the letter keys (with CAPS LOCK off) results in that letter being enciphered. As each letter is enciphered, the display is updated to show the new rotor positions, the entered plaintext, and the resulting ciphertext.

Pressing $\boxed{\text{C}}$ key will erase the plain- and ciphertext strings from the screen.

Pressing $\boxed{\text{R}}$ key will prompt for new rotor positions.

Pressing $\boxed{\text{X}}$ key will terminate the program.

7.2 An Example

Now it's time to work through an example. This is taken from the book *Enigma* by Robert Harris and the subsequent movie starring Dougray Scott and Kate Winslet. (Need I say that the book is better than the movie?)

The keys for March 1943 are given in part as:

```
27 III II. V.. LZC DV LF NQ GE OS FK EW MR IT HK
28 IV. V.. III XRV SY EK NZ OR CG JM QU PV BI LW
29 V.. II. IV. TPK JT NW DU EO KV BY FS HQ IM LX
```

An intercepted message is given as:

```
STNX
B28/03/43 1930 5886 SF282 A236
OKH DE ADU (1830) 174= QAP CWU=
UFJZS NKIRA CGTPF UONXD GQMPU QXUGF OWEZS TCBJD
JLFME AZQRM NZZYI CGSSR YOFQX ADSPU QIMXM MELYR
XKXYI MDEEW ISKDP RSTFR TCOKB GGQTQ KPKMP NCCGH
YUVJO TIVMA IVIGK WQKWJ FOYMR VFBVY RKEZF SYCBY
QQSOQ CIZUU SUTB
```

Obviously, we're dealing here with the three-rotor variant.

7.2.1 Entering the Key

Upon starting up, the first thing the program asks for is the rotor order. Looking in the row corresponding to the message date and translating from Roman numerals, this is determined to be 453. Press $\boxed{4}$ $\boxed{5}$ $\boxed{3}$ followed by $\boxed{\text{ENTER}}$ to accept the setting.

The next thing the program asks for are the ring settings. In the key, this is given immediately following the rotor order. In this case these are `XRV`. Press $\boxed{X}$ $\boxed{R}$ $\boxed{V}$ followed by $\boxed{\text{ENTER}}$.

The next thing the program asks for are the plugboard connections. These are the letter pairs in the key. Enter the string `SYEKNZORCGJMQUPVBILW` followed by $\boxed{\text{ENTER}}$.

Now the program asks for the rotor positions. The first trio of letters enclosed between the equals signs (=), sent in the clear, gives the rotor positions used to encode the message key. For this message, this is `QAP`. Press $\boxed{Q}$ $\boxed{A}$ $\boxed{P}$ followed by $\boxed{\text{ENTER}}$.

When the screen clears and then displays the current rotor positions (`[QAP]`), we can decipher the message key, the second trio of letters enclosed between the equals signs. Press $\boxed{C}$ $\boxed{W}$ $\boxed{U}$. You do not need to press $\boxed{\text{ENTER}}$ here. Instead, take note of the three letters at the top of the display, the `[MPY]`. This is the message key—the starting rotor positions used to encrypt this particular message.

I'm going to assume you can remember these three letters, at least long enough to enter them in the next step. Press $\boxed{\text{R}}$ now and the program prompts for the new rotor positions. Enter the message key, followed by $\boxed{\text{ENTER}}$.

Even though the text will scroll across the screen, it might be helpful to clear the screen before deciphering the message itself. Do this by pressing $\boxed{\text{C}}$.

7.2.2 Deciphering the Message

Now we can start deciphering the message proper. Begin entering the ciphertext, beginning with the first five-letter group `UFJZS`.

After the first three five-letter groups, you should see the text *An OKH. Dringend*. [To Army High Command. Urgent.] (Recall from the Operational Procedures that `X` was generally used as a full stop.)

Type in the remainder of the ciphertext, reading the plaintext off the bottom row of the display.

If you can read German, you should be able to understand the entire message. Even if you can't read German, you should be able to puzzle out that the message is referring to something west of Smolensk. Remember that you can clear the screen at any time by pressing the $\boxed{\text{C}}$ key.

When you're finished, press $\boxed{\text{X}}$ to terminate the program.

7.2.3 Doing it With Four Rotors

The four-rotor Enigma was capable of emulating the three-rotor Enigma. This was necessary to allow U-boats to communicate with other units which did not have the four-rotor machine. This was done by using the `B` "thin" rotor with ring setting `A` and position `A`.

To decipher the above message in four-rotor mode, specify rotor order `B453`, ring settings `AXRV`, the same plugboard connections, and starting rotor positions `AQAP`. After obtaining the message key `MPY`, set the rotor positions to `AMPY` and proceed to decipher the message.

A Literate Programs

This document not only describes the implementation of the Enigma Emulator, it *is* the implementation. The noweb system for "literate programming" generates both the document and the code from a single source. This source consists of interleaved prose and labelled code *fragments*. The fragments are written in the order that best suits describing the program, namely the order you see in this document, not the order dictated by the BCPL/ programming language. The program noweave accepts the source and produces the document's typescript, which includes all of the code and all of the text. The program notangle extracts all of the code, in the proper order for compilation.

Fragments contain source code and references to other fragments. Fragment definitions are preceded by their labels in angle brackets. For example, the code

27a ⟨*a fragment label* 27a⟩≡ 27c ▷
```
sum := 0
FOR i = 1 TO 10 DO ⟨increment sum 27b⟩
```

27b ⟨*increment sum* 27b⟩≡ (27a)
```
 sum := sum + x!i
```

sums the elements of `x`. Several fragments may have the same name; notangle concatenates their definitions to produce a single fragment. noweave identifies this concatenation by using + ≡ instead of ≡ in continued definitions:

27c ⟨*a fragment label* 27a⟩+≡ ◁ 27a
```
sawritef("%i*n", sum)
```

Fragment definitions are like macro definitions; notangle extracts a program by expanding one fragment. If its definition refers to other fragments, they themselves are expanded, and so on.

Fragment definitions include aids to help readers navigate among them. Each fragment name ends with the number of the page on which the fragment's definition begins and a letter giving its sequence within that page. If there is only one fragment on a page then there is no letter. This is also shown in the left margin. Each continued definition also shows the previous definition, and the next continued definition, if there is one. ◁ 7b is an example of a previous definition that appears on page 7, and 11 ▷ says the definition is continued on page 11. These annotations form a double linked list of definitions; the left arrow points to the previous definition in the list and the right arrow points to the next one. The previous link on the first definition is omitted, and the next link on the last definition is omitted. These lists are complete: If some of a fragment's definition appears on the same page with each other, the links refer to the page on which they appear.

Fragments also show a list of pages on which the fragment is used, as illustrated by the (27a) to the right of the definition for ⟨*increment sum*⟩, above.

B Index of Code Fragments

Underlined entries are to the definition of the Code Fragment. In many cases, the definition of a fragment can be continued from one piece to another.

C Index of Identifiers

Underlined entries are their definitions. Standard library definitions are not listed here. Nor are `FOR` control variables and most other variables local to a procedure.