

# An Enigma in the Palm of Your Hand

Kiyoshi Akima

<http://kiyoshiakima.tripod.com/funprogs>

2006.04.27

# Contents

<b>1</b>	<b>Rotor-Based Cipher Machines</b>	<b>1</b>
<b>2</b>	<b>The Enigma</b>	<b>2</b>
2.1	Machine Components . . . . .	2
2.1.1	Rotors . . . . .	2
2.1.2	Stepping Motion . . . . .	3
2.1.3	Reflector . . . . .	4
2.1.4	Plugboard . . . . .	4
2.2	Basic Operational Procedures . . . . .	4
2.2.1	Indicators . . . . .	5
2.2.2	Abbreviations and Guidelines . . . . .	6
2.3	Variants . . . . .	7
2.4	Breaking the Enigma . . . . .	7
2.4.1	Security Properties . . . . .	8
2.4.2	Solution Before World War II . . . . .	8
2.4.3	World War II . . . . .	9
<b>3</b>	<b>Emulating the Enigma on the HP 48</b>	<b>10</b>
<b>4</b>	<b>Key Entry</b>	<b>12</b>
4.1	Prompting for Input . . . . .	12
4.2	Rotor Order . . . . .	12
4.3	Ring Settings . . . . .	13
4.4	Plugboard Connections . . . . .	13
4.5	Starting Rotor Positions . . . . .	13
<b>5</b>	<b>Configuring the Virtual Machine</b>	<b>14</b>
5.1	Preparing . . . . .	14
5.1.1	Converting an ASCII String to Binary . . . . .	14
5.1.2	Converting a List to a String . . . . .	14
5.1.3	Breaking out a String . . . . .	14
5.1.4	Rotor Configuration . . . . .	15
5.1.5	Storing it Away . . . . .	16
5.2	Reflector . . . . .	16
5.3	Plugboard . . . . .	16
5.4	Rotors . . . . .	17
5.4.1	Notches and Wiring . . . . .	17
5.4.2	Ring Settings . . . . .	17
5.4.3	Initial Positions . . . . .	18
5.4.4	Keeping in Range . . . . .	18
5.4.5	Translation . . . . .	18
5.5	Input and Output . . . . .	18
5.6	Loop Exit . . . . .	19
5.7	Finishing . . . . .	19

<b>6</b>	<b>Enciphering Text</b>	<b>20</b>
6.1	Controlling the Display . . . . .	20
6.2	Getting a Keystroke . . . . .	20
6.3	Checking Special Keys . . . . .	21
6.3.1	Clearing the Screen . . . . .	21
6.3.2	Entering New Rotor Positions . . . . .	21
6.3.3	Terminating the Loop . . . . .	21
6.3.4	Ignoring Other Keys . . . . .	22
6.4	Enciphering One Letter . . . . .	22
6.4.1	Plaintext . . . . .	22
6.4.2	Rotor Step . . . . .	22
6.4.3	Plugboard . . . . .	23
6.4.4	Rotors . . . . .	23
6.4.5	Reflector . . . . .	23
6.4.6	Rotors . . . . .	23
6.4.7	Plugboard . . . . .	24
6.4.8	Ciphertext . . . . .	24
6.5	Finishing Up . . . . .	24
<b>7</b>	<b>Running the Program</b>	<b>25</b>
7.1	Basic Operation . . . . .	25
7.2	An Example . . . . .	26
7.2.1	Entering the Key . . . . .	26
7.2.2	Deciphering the Message . . . . .	27
7.2.3	Doing it With Four Rotors . . . . .	27
<b>A</b>	<b>Literate Programming</b>	<b>28</b>
<b>B</b>	<b>Building the Program</b>	<b>29</b>
B.1	Tangling . . . . .	29
B.2	Compiling . . . . .	29
B.3	Assembling . . . . .	29
B.4	Linking . . . . .	29
<b>A</b>	<b>Index of Code Fragments</b>	<b>30</b>

## 1 Rotor-Based Cipher Machines

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 skillful efforts of French, British, and American cryptanalysts. By the time the war ended in 1918, Allied code and cipher experts had 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 new 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 cipher 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. Internally, 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 enciphered. 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) and 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 different 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 Emulating the Enigma on the HP 48

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?

In this document I’m going to go one step further and, instead of emulating the Enigma on a computer, I’m going to emulate it on the HP 48 pocket calculator. In addition, I’m going to do it in System RPL (SysRPL).

```

10  <enigma.s 10>≡
    ( Enigma Emulator )
    ( Kiyoshi Akima   )
    ( 2006.04.24      )

    ASSEMBLE
      NIBASC  /HHP48-E/
    RPL
11a>
```

Programming in SysRPL has advantages over programming in UserRPL, as well as disadvantages. On the plus side, a SysRPL program can often be faster and smaller than an equivalent UserRPL program. In addition, SysRPL lets the programmer do some things that simply cannot be done in UserRPL.

On the minus side, much of the time and memory savings come at the expense of safety: a program bug can result in crashes and even completely memory loss instead of merely incorrect results. In addition, SysRPL requires building the program on a computer and subsequently transferring it to the calculator, instead of merely keying it in.

```
11a  <enigma.s 10>+≡                                     <10
      ::
      <main program 11b>
      ;
```

The program is interactive, taking no arguments on the stack.

```
11b  <main program 11b>≡                                     (11a) 11c>
      AtUserStack
```

The program quite naturally decomposes into three main modules. These modules are:

- Get the key settings from the user.
- Configure the virtual machine.
- Encipher/decipher text.

```
11c  <main program 11b>+≡                                     (11a) <11b
      <get key settings 12a>
      <configure virtual machine 14a>
      <encipher text 20a>
```

Each of these modules is discussed in detail in a subsequent section of this document. Readers not interested in the inner workings of the program may skip ahead to Section 7, “Running the Program” on page 25.

## 4 Key Entry

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

```
12a  <get key settings 12a>≡ (11c)
      <define prompting routine 12b>
      <get rotor order 12c>
      <get ring settings 13d>
      <get plugboard connections 13e>
      <get rotor starting positions 13f>
```

### 4.1 Prompting for Input

This program prompts for all components of the key setting. However, it does *no* error checking. The closest thing to a check is seeing whether the user specified four rotors in the order; the program takes this to mean it is emulating the four-rotor Navy Enigma.

Just to make life a little easier (and the program a little smaller) the program defines a routine to do the actual prompting. The parameters specify cursor at the beginning, replace mode, program/algebraic entry, alpha locked, no menu, menu row one, **ON** clears, and no post action.

```
12b  <define prompting routine 12b>≡ (12a)
      ' ::
          ONE TWO ONEONE NULL{} ONE FALSE ZERO InputLine
      ;
```

### 4.2 Rotor Order

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 Roman letters B and G for the Greek letters Β and Γ. They were usually specified from left to right, and this program maintains that tradition.

```
12c  <get rotor order 12c>≡ (12a) 13a▷
      "Rotor Order:" NULL$
      3PICK EVAL
      NOT_IT :: DROP 2RDROP ;
```

User flag 64 is set if the four-rotor Enigma is being emulated, cleared otherwise.

```
13a  <get rotor order 12c>+≡ (12a) <12c 13b>
      DUPLen$ FOUR #=ITE
      :: SIXTYFOUR SetUserFlag ;
      :: SIXTYFOUR ClrUserFlag ;
```

If the four-rotor Enigma is being emulated, I move the “thin” rotor specifier from the beginning to the end. This is so that the positions of the three “normal” rotors are at the same place regardless of the presence or absence of the fourth rotor.

```
13b  <get rotor order 12c>+≡ (12a) <13a
      <move thin to end 13c>
```

Moving a letter from the head of a string to the tail is a simple exercise in string manipulation.

```
13c  <move thin to end 13c>≡ (13)
      SIXTYFOUR TestUserFlag IT :: DUP CDR$ SWAP CAR$ >T$ ;
```

### 4.3 Ring Settings

The ring settings were usually specified by three or four letters, from left to right.

```
13d  <get ring settings 13d>≡ (12a)
      "Ring Settings:" NULL$
      4PICK EVAL
      NOT_IT :: 2DROP 2RDROP ;
      <move thin to end 13c>
```

### 4.4 Plugboard Connections

The plugboard connections were usually specified by pairs of letters; this program uses a single string without breaks or other punctuation.

```
13e  <get plugboard connections 13e>≡ (12a)
      "Plugboard Connections:" NULL$
      5PICK EVAL
      NOT_IT :: 3DROP 2RDROP ;
```

### 4.5 Starting Rotor Positions

Like the ring settings, the rotor starting positions were specified by three or four letters, again from left to right.

```
13f  <get rotor starting positions 13f>≡ (12a)
      "Rotor Positions:" NULL$
      6PICK EVAL
      NOT_IT :: 4DROP 2RDROP ;
      <move thin to end 13c>
```



## 5 Configuring the Virtual Machine

Once the program has acquired the key settings from the user, it can proceed to configure the virtual machine.

```
14a  <configure virtual machine 14a>≡ (11c)
      <prepare to configure 14b>
      <configure components 16c>
      <end configure 19c>
```

### 5.1 Preparing

There're a few more things the program needs in order to configure the virtual machine. These things will be placed in a temporary null-named environment for use during the configuration.

#### 5.1.1 Converting an ASCII String to Binary

There are times when the program needs to convert a string of 26 letters to a binary string. This can be done by simply ANDing the string with an appropriate length string of question marks (hex 3F).

```
14b  <prepare to configure 14b>≡ (14a) 14c>
      "?????????????????????????"
```

#### 5.1.2 Converting a List to a String

There are also times when the program needs to convert a list to a string. This conversion simply proceeds one item at a time.

```
14c  <prepare to configure 14b>+≡ (14a) <14b 14d>
      ' ::
          NULL$
          TWENTYSIX ZERO_DO (DO)
              OVER CARCOMP #>CHR >T$ SWAP CDRCOMP SWAP
          LOOP
          SWAPDROP
      ;
```

#### 5.1.3 Breaking out a String

There are also times when the program needs to break out the three or four individual settings from a string. This is complicated by the fact that there may be either three or four.

```
14d  <prepare to configure 14b>+≡ (14a) <14c 15a>
      ' ::
          THREE #1+_ONE_DO (DO)
              DUP CDR$ SWAPONE SUB$1# SIXTYFOUR #- SWAP
          LOOP
          SIXTYFOUR TestUserFlag ITE
              :: ONE SUB$1# SIXTYFOUR #- ;
          DROPZERO
      ;
```

### 5.1.4 Rotor Configuration

Obviously, the program needs the wiring and notch positions for the individual rotors.

```

15a  <prepare to configure 14b>+≡ (14a) <14d 16b>
      ' ::
      SUB$1#
      ::
      <select rotor configuration 15b>
      ;
      <compile rotor 15f>
      ;

```

#### Rotor Wiring and Notches

Here are the notch position(s) and wiring for all the rotors this program uses. Note that any invalid rotor specifier gets mapped to rotor I.

Here are the Army/Air Force/Navy rotors II-V.

```

15b  <select rotor configuration 15b>≡ (15a) 15c>
      FIFTY      #=<casedrop :: "AJDKSIRUXBLHWTMCQGZNPYFVOE" "E" ;
      FIFTYONE   #=<casedrop :: "BDFHJLCPRTXVZNYEIWGAKMUSQO" "V" ;
      FIFTYTWO   #=<casedrop :: "ESOVZPJAYQUIRXLNFTGKDCMWB" "J" ;
      FIFTYTHREE #=<casedrop :: "VZBRGITYUPSDNHLXAWMJQOFECK" "Z" ;

```

And the three additional Navy rotors VI-VIII.

```

15c  <select rotor configuration 15b>+≡ (15a) <15b 15d>
      FIFTYFOUR  #=<casedrop :: "JPGVOUMFYQBENHZRDKASXLICTW" "MZ" ;
      FIFTYFIVE  #=<casedrop :: "NZJHGRCXMYSWBOUFAIVLPEKQDT" "MZ" ;
      FIFTYSIX   #=<casedrop :: "FKQHTLXOCBJSPDZRAMWNIUYPV" "MZ" ;

```

And the fourth, “thin” rotors B and Γ.

```

15d  <select rotor configuration 15b>+≡ (15a) <15c 15e>
      66         #=<casedrop :: "LEYJVCNIXWPBQMDRTAKZGFUHS" NULL$ ;
      71         #=<casedrop :: "FSOKANUERHMBTIYCWLPZXVGD" NULL$ ;

```

And finally rotor I, which will also be used if an invalid specifier is given.

```

15e  <select rotor configuration 15b>+≡ (15a) <15d
      DROP      "EKMFLGDQVZNTOWYHXUSPAIBRCJ" "Q"

```

#### Compiling the Rotor

For ease of use, this program deals with binary instead of ASCII strings. They're still strings, but instead of containing ASCII letters they contain bytes in the range 1-26.

First the program converts the notch positions.

```

15f  <compile rotor 15f>≡ (15a) 16a>
      DUPL$ DUP #0=ITE
      DROP
      :: 4GETLAM ONE ROT SUB$ AND$ ;
      SWAP

```

The program also determines the reverse mapping for the return journey through the rotors. These could have been specified at compile time instead of being computed at run time, but then it gets harder to add new rotors or to modify the existing ones. And it's not as if either time or memory are in that short a supply.

16a  $\langle \text{compile rotor 15f} \rangle + \equiv$  (15a)  $\triangleleft 15f$

```

4GETLAM AND$
ZERO TWENTYSIX NDUPN {}N
TWENTYSIX #1+_ONE_DO (DO)
    OVER INDEX@ SUB$1# INDEX@ SWAPROT PUTLIST
LOOP
3GETLAM EVAL &$

```

### 5.1.5 Storing it Away

Okay, the program now has everything it need to configure the virtual machine. It takes everything off the stack and put it into a temporary null-named environment so I don't have to keep juggling the stack.

For reference, here's what's in the temporary:

Pos	Item	Type	Pos	Item	Type
1	rotor configuration	secondary	5	rotor positions	string
2	break out a string	secondary	6	plugboard	string
3	list $\rightarrow$ string	secondary	7	ring settings	string
4	string $\rightarrow$ binary	string	8	rotor order	string

16b  $\langle \text{prepare to configure 14b} \rangle + \equiv$  (14a)  $\triangleleft 15a$

```

NULLLAM EIGHT NDUPN DOBIND

```

## 5.2 Reflector

Now for the configuration itself. The reflector is the simplest component, at least in terms of the emulation, so the program starts with that. The only trick is determining which reflector; the three- and four-rotor Enigmas had different reflectors. In addition, different reflectors were issued at different times: this program uses the 'B' reflector for both the three-letter and four-letter modes.

16c  $\langle \text{configure components 16c} \rangle + \equiv$  (14a) 16d  $\triangleright$

```

SIXTYFOUR TestUserFlag ITE
    "ENKQAUYWJICOPBLMDXZVFTHRGs"
    "YRUHQSLDPXNGOKMIEBFZCWVJAT"
4GETLAM AND$

```

## 5.3 Plugboard

The plugboard is a little more complicated. First the program builds a list containing the numbers 1, ... 26 representing the plugboard with no plugs inserted.

16d  $\langle \text{configure components 16c} \rangle + \equiv$  (14a)  $\triangleleft 16c$  17a  $\triangleright$

```

TWENTYSIX #1+_ONE_DO (DO) INDEX@ LOOP TWENTYSIX {}N

```

Then if the user specified at least one pair of plugboard connections, the program writes the pairs into the list. If the user specified an odd number of letters, then the last (unpaired) letter is discarded. However, the program does not check to see whether any letter is specified more than once.

```

17a  <configure components 16c>+≡ (14a) <16d 17b>
      6GETLAM DUPLN$ ONE#> IT ::
      DUPLN$ ONE_DO (DO)
      SWAPOVER ONE SUB$1# SIXTYFOUR #-
      3PICK TWO SUB$1# SIXTYFOUR #-
      2DUP5ROLL PUTLIST UNROT SWAPROT PUTLIST
      SWAP CDR$ CDR$
      TWO +LOOP
      ;
      DROP

```

Then the list is converted to a binary string.

```

17b  <configure components 16c>+≡ (14a) <17a 17c>
      3GETLAM EVAL

```

## 5.4 Rotors

Now for the rotors.

### 5.4.1 Notches and Wiring

The program starts with the three basic rotors, working from left to right.

```

17c  <configure components 16c>+≡ (14a) <17b 17d>
      THREE #1+_ONE_DO (DO)
      8GETLAM INDEX@ 1GETLAM EVAL
      LOOP

```

The program starts with the leftmost, the fourth, “thin” rotor. Obviously, this rotor isn’t present if the program is only emulating the three-rotor machine. If this fourth rotor is present, the program doesn’t care about its notch position.

```

17d  <configure components 16c>+≡ (14a) <17c 17e>
      SIXTYFOUR TestUserFlag ITE
      :: 8GETLAM FOUR 1GETLAM EVAL SWAPDROP ;
      NULL$

```

### 5.4.2 Ring Settings

Now the program breaks out the ring settings, converting each one to a binary integer. Again the program has to deal with the presence or absence of the fourth rotor.

```

17e  <configure components 16c>+≡ (14a) <17d 18a>
      7GETLAM 2GETLAM EVAL

```

### 5.4.3 Initial Positions

The program breaks out the initial positions in the same way.

```
18a  <configure components 16c>+≡ (14a) <17e 18b>
      5GETLAM 2GETLAM EVAL
```

The program also needs to be able to build a string out of the current rotor positions, both for display and for prompting.

```
18b  <configure components 16c>+≡ (14a) <18a 18c>
      ' ::
          SIXTYFOUR TestUserFlag IT :: 7GETLAM SIXTYFOUR #+ #>CHR >T$ ;
          10GETLAM SIXTYFOUR #+ #>CHR >T$
          9GETLAM SIXTYFOUR #+ #>CHR >T$
          8GETLAM SIXTYFOUR #+ #>CHR >T$
      ;
```

### 5.4.4 Keeping in Range

At various times the program needs to make sure it don't fall off the end of the alphabet. This code is also used to keep the program from falling off the end of a circular rotor.

```
18c  <configure components 16c>+≡ (14a) <18b 18d>
      ' ::
          TWENTYSIX OVER#< IT :: TWENTYSIX #- ;
      ;
```

### 5.4.5 Translation

The process of translating a letter through a rotor requires compensating for the rotor position and the ring setting, taking care to keep the value within the range 1-26. With the reverse mapping generated earlier, the same code can be used for both the forward and reverse translations.

```
18d  <configure components 16c>+≡ (14a) <18c 18e>
      ' ::
          #- TWENTYSIX #+          5GETLAM EVAL
          ROT TWENTYSIX #+ OVER#- 5GETLAM EVAL
          ROTSWAP SUB$1# #+        5GETLAM EVAL
      ;
```

## 5.5 Input and Output

The program keeps track of the recent results, both the entered plaintext and the resultant ciphertext.

```
18e  <configure components 16c>+≡ (14a) <18d 19a>
      TWENTYTWO Blank$ DUP
```

## 5.6 Loop Exit

Finally, the program needs a flag to tell it when to exit the encipherment loop.

19a     $\langle \text{configure components 16c} \rangle + \equiv$  (14a)  $\langle 18e \ 19b \rangle$   
       FALSE

## 5.7 Finishing

Okay, that's everything the program needs to encipher/decipher text. It can now get rid of the temporary environment.

19b     $\langle \text{configure components 16c} \rangle + \equiv$  (14a)  $\langle 19a$   
       ABND

And now the program puts all this into a new temporary environment so it again doesn't have to juggle the stack.

For reference, here's what's in the temporary:

Pos	Item	Type	Pos	Item	Type
1	loop exit	flag	13	ring 2	bint
2	ciphertext	string	14	ring 1	bint
3	plaintext	string	15	wiring 4	string
4	rotor translation	secondary	16	wiring 3	string
5	ensure letter	secondary	17	notches 3	string
6	rotor string	secondary	18	wiring 2	string
7	position 4	bint	19	notches 2	string
8	position 3	bint	20	wiring 1	string
9	position 2	bint	21	notches 1	string
10	position 1	bint	22	plugboard	string
11	ring 4	bint	23	reflector	string
12	ring 3	bint	24	prompt	secondary

19c     $\langle \text{end configure 19c} \rangle \equiv$  (14a)  
       NULLAM TWENTYFOUR NDUPN DOBIND

## 6 Enciphering Text

The encipherment occurs inside a loop, terminated when the user presses the **ON** key. If the user presses the **ENTER** key, the program prompts for the new rotor positions.

```

20a  <encipher text 20a>≡ (11c)
      <initialize display 20b>
      BEGIN
        <show rotor positions 20d>
        <get keystroke 20e>
        ::
          <check special keys 21a>
          <encipher one letter 22b>
        ;
      1GETLAM UNTIL
      <terminate display 20c>
      <terminate program 24c>

```

### 6.1 Controlling the Display

Before the program starts the enciphering loop, it turns off the ticking clock (whether or not it's actually displayed) and turns off the menu.

```

20b  <initialize display 20b>≡ (20a 21b)
      RECLAIMDISP C1rDA1IsStat TURNMENUOFF

```

When the program is finished, it turns the menu back on and relinquishes the display, telling the system to update it.

```

20c  <terminate display 20c>≡ (20a 21b)
      TURNMENUOFF RECLAIMDISP C1rDA1sOK

```

At the top of the loop the program displays the current rotor positions, similar to the way they appeared through windows on a real Enigma machine.

```

20d  <show rotor positions 20d>≡ (20a)
      "          ["
      6GETLAM EVAL
      CHR_] >T$ BIGDISPROW4

```

### 6.2 Getting a Keystroke

There's no sense in having the program run down the batteries simply waiting for the user to press a key. The program uses `WaitForKey`, which places the calculator into a low-power state to conserve batteries. When a key is pressed, it returns a fully formed keystroke specifying the keycode and shift plane. The program doesn't care about the plane so it drops it from the stack.




```

20e  <get keystroke 20e>≡ (20a)
      WaitForKey DROP


```

### 6.3 Checking Special Keys

In addition to the letter keys, several keys have special meaning to the program.

Key	Action
	Clear the screen
	Enter new rotor positions
	Break out of the loop and terminate the program

#### 6.3.1 Clearing the Screen



The  key clears the plain- and ciphertext strings and erases them from the display. The rotor positions, however, remain.

```

21a  <check special keys 21a>≡ (20a) 21b>
      TWENTYNINE #=casedrop ::
      TWENTYTWO Blank$ DUP 2DUP
      2PUTLAM BIGDISPROW2
      3PUTLAM BIGDISPROW3
      ;

```

#### 6.3.2 Entering New Rotor Positions


The  key prompts for the new rotor positions. If the user presses  at the prompt to provide new rotor positions then the program stores them, otherwise nothing happens.

```

21b  <check special keys 21a>+≡ (20a) <21a 21c>
      TWENTYFIVE #=casedrop ::
      <terminate display 20c>
      "Rotor Positions:" NULL$ 6GETLAM EVAL
      TWENTYFOUR GETLAM EVAL
      IT ::
      SIXTYFOUR TestUserFlag IT
      :: DUPONE SUB$1# SIXTYFOUR #- 7PUTLAM CDR$ ;
      DUPONE SUB$1# SIXTYFOUR #- 10PUTLAM CDR$
      DUPONE SUB$1# SIXTYFOUR #- 9PUTLAM CDR$
      ONE SUB$1# SIXTYFOUR #- 8PUTLAM
      ;
      <initialize display 20b>
      ;

```

#### 6.3.3 Terminating the Loop

The  key sets the flag to break out of the loop and terminate the program.

```

21c  <check special keys 21a>+≡ (20a) <21b 22a>
      FORTYFIVE #=casedrop :: TRUE 1PUTLAM ;

```



### 6.3.4 Ignoring Other Keys

The program ignores other non-letter keys, dropping them from the stack.

```
22a  <check special keys 21a>+≡ (20a) <21c
      DUP TWENTYSEVEN #> case DROP
```

## 6.4 Enciphering One Letter

The user pressed a letter key. Unfortunately the sequence of letter keys is broken by the **ENTER** key which separates the Y and Z from the rest of the alphabet. Fortunately the rest of the letter keys falls nicely into the A = 1, B = 2 pattern so not much adjustment is necessary.

```
22b  <encipher one letter 22b>≡ (20a) 22c>
      DUP TWENTYFIVE #> IT #1-
```

### 6.4.1 Plaintext

Before enciphering the letter, the program drops the first letter from the head of the plaintext string and appends the new letter to the tail. This string is displayed in lowercase.

```
22c  <encipher one letter 22b>+≡ (20a) <22b 22d>
      3GETLAM CDR$ OVER 96 #+ #>CHR >T$ DUP 3PUTLAM BIGDISPROW3
```

### 6.4.2 Rotor Step

Before the letter is enciphered, 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 so it can be ignored for the duration of this discussion.

```
22d  <encipher one letter 22b>+≡ (20a) <22c 23b>
      <step right rotor 22e>
      19GETLAM 9GETLAM #>CHR CHR>$ ONE POS$ #0=ITE
      :: <step middle rotor? 23a> ;
      :: <step left rotor? 22f> ;
```

The right rotor steps on every keystroke.

```
22e  <step right rotor 22e>≡ (22d)
      8GETLAM DUP#1+ 5GETLAM EVAL 8PUTLAM
```

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

```
22f  <step left rotor? 22f>≡ (22d)
      10GETLAM #1+ 5GETLAM EVAL 10PUTLAM
      9GETLAM #1+ 5GETLAM EVAL 9PUTLAM
      DROP
```

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

23a  $\langle \text{step middle rotor? 23a} \rangle \equiv$  (22d)  
 $\# > \text{CHR CHR} > \$ 17 \text{GETLAM SWAPONE POS} \$ \# 0 < > \text{IT}$   
 $:: 9 \text{GETLAM} \# 1 + 5 \text{GETLAM EVAL } 9 \text{PUTLAM} ;$

### 6.4.3 Plugboard

Now for the encipherment itself. First the current passes through the plug-board...

23b  $\langle \text{encipher one letter 22b} \rangle + \equiv$  (20a)  $\langle 22d \ 23c \rangle$   
 $22 \text{GETLAM SWAP SUB} \$ 1 \#$

### 6.4.4 Rotors

... then through the right rotor...

23c  $\langle \text{encipher one letter 22b} \rangle + \equiv$  (20a)  $\langle 23b \ 23d \rangle$   
 $16 \text{GETLAM } 12 \text{GETLAM } 8 \text{GETLAM } 4 \text{GETLAM EVAL}$

... and similarly through the next two rotors...

23d  $\langle \text{encipher one letter 22b} \rangle + \equiv$  (20a)  $\langle 23c \ 23e \rangle$   
 $18 \text{GETLAM } 13 \text{GETLAM } 9 \text{GETLAM } 4 \text{GETLAM EVAL}$   
 $20 \text{GETLAM } 14 \text{GETLAM } 10 \text{GETLAM } 4 \text{GETLAM EVAL}$

... and through the fourth rotor if present...

23e  $\langle \text{encipher one letter 22b} \rangle + \equiv$  (20a)  $\langle 23d \ 23f \rangle$   
 $\text{SIXTYFOUR TestUserFlag IT}$   
 $:: 15 \text{GETLAM } 11 \text{GETLAM } 7 \text{GETLAM } 4 \text{GETLAM EVAL} ;$

### 6.4.5 Reflector

... then bounces through the reflector...

23f  $\langle \text{encipher one letter 22b} \rangle + \equiv$  (20a)  $\langle 23e \ 23g \rangle$   
 $\text{TWENTYTHREE GETLAM SWAP SUB} \$ 1 \#$

### 6.4.6 Rotors

... which sends it back through the fourth rotor if present...

23g  $\langle \text{encipher one letter 22b} \rangle + \equiv$  (20a)  $\langle 23f \ 23h \rangle$   
 $\text{SIXTYFOUR TestUserFlag IT}$   
 $:: 15 \text{GETLAM TWENTYSEVEN FIFTYTWO SUB} \$$   
 $11 \text{GETLAM } 7 \text{GETLAM } 4 \text{GETLAM EVAL} ;$

... and back through the other three rotors...

23h  $\langle \text{encipher one letter 22b} \rangle + \equiv$  (20a)  $\langle 23g \ 24a \rangle$   
 $20 \text{GETLAM TWENTYSEVEN FIFTYTWO SUB} \$ 14 \text{GETLAM } 10 \text{GETLAM } 4 \text{GETLAM EVAL}$   
 $18 \text{GETLAM TWENTYSEVEN FIFTYTWO SUB} \$ 13 \text{GETLAM } 9 \text{GETLAM } 4 \text{GETLAM EVAL}$   
 $16 \text{GETLAM TWENTYSEVEN FIFTYTWO SUB} \$ 12 \text{GETLAM } 8 \text{GETLAM } 4 \text{GETLAM EVAL}$

### 6.4.7 Plugboard

... and finally back through the plugboard again.

24a     $\langle \textit{encipher one letter 22b} \rangle + \equiv$  (20a)     $\langle 23h \ 24b \rangle$   
       22GETLAM SWAP SUB\$1#

### 6.4.8 Ciphertext

On a real Enigma machine the output was signaled by a glowing lamp. This program does it by adding the letter to the end of the ciphertext string at the top of the display. The procedure is virtually identical to that for the plaintext display, the primary difference being that this time it is done in uppercase.

24b     $\langle \textit{encipher one letter 22b} \rangle + \equiv$  (20a)     $\langle 24a$   
       2GETLAM CDR\$ SWAP SIXTYFOUR #+ #>CHR >T\$ DUP 2PUTLAM BIGDISPROW2

## 6.5 Finishing Up

Before the program terminates, it frees the temporary environment.

24c     $\langle \textit{terminate program 24c} \rangle \equiv$  (20a)  
       ABND

As it currently stands the program is 1988 bytes checksum 6781h.

## 7 Running the Program

### 7.1 Basic Operation

As mentioned earlier the program requires no arguments on the stack: all input is done interactively. You may run it by any of the usual methods: the easiest probably is by pressing the key assigned to the variable containing the program.

All of the prompts have the alpha lock engaged so you don't have to press the  key (unless you need to use the cursor keys).

For all prompts, the  key accepts the current input string.

The  key will erase the current input string if it is not empty.

If the input string is empty, the  key will abort the program.

The program first 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 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 specified 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. This program expects three or four uppercase letters.

The third prompt is for the plugboard connections (*Steckerverbindungen*). This was specified as zero or more (commonly ten) pairs of letters. This program expects a string of 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. Like the ring settings, this is a string of three or four letters.

After the final prompt there is a four-second pause while the program configures the virtual machine. Then the screen is cleared, the current rotor positions are displayed as they are on a real Enigma, and the virtual machine is ready to begin enciphering/deciphering text.

Pressing any of the letter keys 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 the  key will erase the plain- and ciphertext strings from the screen.

Pressing the  key will prompt for new rotor positions.

Pressing the  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
```

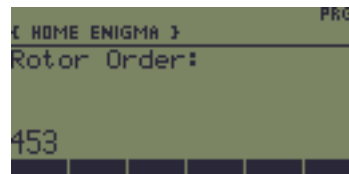
An intercepted message is given as:

```
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
KKXYI 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

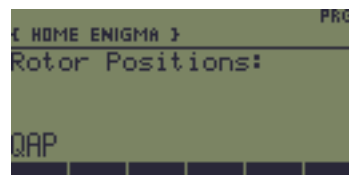
Upon starting up, the first thing the program asks for is the rotor order. Looking in the row corresponding to the message date and translating from Roman numerals, this is determined to be 453. Press    followed by  to accept the setting.



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    followed by .

The next thing the program asks for are the plugboard connections. These are the letter pairs in the key. Press                     followed by .

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    followed by .



There is a brief delay while the program configures the virtual machine. 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 C W U. Do *not* press  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  now and the program prompts for the rotor positions. Enter the message key, followed by .

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 .

### 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 this screen. The message starts out: *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 top row of the display. Remember that you can clear the screen at any time by pressing the  key.

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.

When you're finished, press  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 Programming

This document not only describes the implementation of the Enigma Emulator for the HP 48, it *is* the implementation. The `noweb` system for “literate programming” generates both the document and the program 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 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. 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:

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.

Most fragments also show a list of pages on which the fragment is used. These unadorned use lists are omitted for root fragments, which define modules.

Of course, the simple fact that the documentation can be placed right next to the code doesn’t necessary mean that the documentation and the code match, nor that either is any good. Still, it’s a good start...

For more information about the `noweb` system of literate programming, please refer to <http://www.eecs.harvard.edu/~nr/noweb>.

## B Building the Program

### B.1 Tangling

The following command line will extract the program source from the `noweb` document:

```
notangle -Renigma.s -t4 enigma48.nw >enigma.s
```

This will create the file `enigma.s`.

### B.2 Compiling

Once the program source has been extracted, it can be compiled with the following command line.

```
rplcomp enigma.s enigma.a
```

This will produce an assembly file `enigma.a`.

### B.3 Assembling

The assembly file can then be assembled with the following command line:

```
sasm enigma.a
```

This will produce an object file `enigma.o` and a listing file `enigma.l`.

### B.4 Linking

The linker requires a command file:

```
29 <enigma.m 29>≡
    LL enigma.lr
    OU enigma
    RE enigma.o
    SE c:\hp48\lib\entries.o
    SU XR
```

This linker command file can be extracted from the `noweb` document with the following command line:

```
notangle -Renigma.m enigma48.nw >enigma.m
```

This will create the file `enigma.m`. (You will probably have to edit the location of `entries.o` to match your directory structure.) Then the linker can be run with this command line:

```
sload -H enigma.m
```

If all goes well, this will produce an HP 48 binary `enigma` and a listing file `enigma.lr`.

The binary can then be transferred to the HP 48, stored in a variable, and executed like any other program.

For further details on the building process, please consult the appropriate HP documentation.



## A 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.

<*check special keys* 21a> 20a, 21a, 21b, 21c, 22a  
 <*compile rotor* 15f> 15a, 15f, 16a  
 <*configure components* 16c> 14a, 16c, 16d, 17a, 17b, 17c, 17d, 17e, 18a, 18b,  
     18c, 18d, 18e, 19a, 19b  
 <*configure virtual machine* 14a> 11c, 14a  
 <*define prompting routine* 12b> 12a, 12b  
 <*encipher one letter* 22b> 20a, 22b, 22c, 22d, 23b, 23c, 23d, 23e, 23f, 23g, 23h,  
     24a, 24b  
 <*encipher text* 20a> 11c, 20a  
 <*end configure* 19c> 14a, 19c  
 <*enigma.m* 29> 29  
 <*enigma.s* 10> 10, 11a  
 <*get key settings* 12a> 11c, 12a  
 <*get keystroke* 20e> 20a, 20e  
 <*get plugboard connections* 13e> 12a, 13e  
 <*get ring settings* 13d> 12a, 13d  
 <*get rotor order* 12c> 12a, 12c, 13a, 13b  
 <*get rotor starting positions* 13f> 12a, 13f  
 <*initialize display* 20b> 20a, 20b, 21b  
 <*main program* 11b> 11a, 11b, 11c  
 <*move thin to end* 13c> 13b, 13c, 13d, 13f  
 <*prepare to configure* 14b> 14a, 14b, 14c, 14d, 15a, 16b  
 <*select rotor configuration* 15b> 15a, 15b, 15c, 15d, 15e  
 <*show rotor positions* 20d> 20a, 20d  
 <*step left rotor?* 22f> 22d, 22f  
 <*step middle rotor?* 23a> 22d, 23a  
 <*step right rotor?* 22e> 22d, 22e  
 <*terminate display* 20c> 20a, 20c, 21b  
 <*terminate program* 24c> 20a, 24c