Appendix 1
ENIGMA3 listing

```
5 "START"
6 DIM A$(10)*160,W(5,26,2),P(5),IR(5),P$(1)*2048,
  C$(1)*(2048)
10 DISP TAB 25;"******* ENIGMA3 *******":DISP
11 DISP TAB 10;"VERSION 26 JUN 84  (C) GEORGE
   SASSOON":DISP
20 DISP "PROGRAM TO SIMULATE ENCRYPTION AND
   DECRYPTION WITH THE ENIGMA CIPHER MACHINE."
22 DISP "THIS VERSION USES FOUR 26-POSITION ROTORS
   AND A ROTATING REFLECTOR."
30 DISP
46 REM   ENTRY AND EXIT POINTS FOR
47 REM  LETTERS TO AND FROM ROTOR 1:
48 REM   ABCDEFGHIJKLMNOPQRSTUVWXYZ
49 REM   ──────────────────────────
50 DATA "ABCDEFGHIJKLMNOPQRSTUVWXYZ":REM  THESE
   ALPHABETS DEFINE THE
52 DATA "MTREIVUWKPOGNBFAZCYDHJSLQX":REM
   CROSS-CONNECTIONS WITHIN
53 !ROTOR 2                                    THE
   ROTORS.  A GOES TO A,
54 DATA "ABCDEFGHIJKLMNOPQRSTUVWXYZ":REM  B GOES
   TO B, ETC. IN EACH
56 DATA "YLOGSAFMXITJEHNCRWZDVQKPBU":REM  PAIR OF
   DATA STATEMENTS.
57 !ROTOR 3                                   EACH
   LETTER MUST OCCUR ONCE
58 DATA "QWERTYUIOPASDFGHJKLZXCVBNM":REM  ONLY IN
   EACH ALPHABET.
60 DATA "POBEVUZXCKDQFARTINGSMLYJHW":REM  IN THE
   CASE OF THE REFLECTOR,
61 !ROTOR 4                                    THE
   CONNECTIONS MUST BE
62 DATA "ETAOINSHRDLUGPVZKJBXYFCQWM":REM
   SYMMETRICAL, I.E. IF T GOES
64 DATA "ZYXWVUTSQRPONMLKJIHGFEDCBA":REM  A THEN A
   MUST GO TO T ETC.?
71 !REFLECTOR
72 DATA "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
74 DATA "TVMFNDJOSGUYCEHQPWIAKBRZLX"
```

```
80 DATA "Z":REM END OF DATA MARKER
90 REM    GO SUB 20000:REM USE TO CHECK ALPHABETS
   OK
95 GO SUB 11000:REM SET UP ROTOR ARRAY
100 GO TO 1000:REM DO THIS TO TEST
105 DISP :DISP TAB 18;"*** INPUT INITIAL ROTOR
    SETTINGS! ***"
110 DISP :FOR I=1 TO 4
120 DISP "ENTER SETTING OF ROTOR";I;"  0-25  : ";:
    INPUT P(I)
130 IF (P(I)<0) OR (P(I)>25) THEN 120
140 IR(I)=P(I):NEXT I
150 INPUT "ENTER SETTING OF REFLECTOR  0-25 :
    ";P(5)
160 IF (P(5)<0) OR (P(5)>25) THEN 150
170 IR(5)=P(5)
180 REM IR ARRAY SAVES INITIAL ROTOR POSITIONS FOR
    PRINTING AND
181 REM TEST DECRYPTION
199 !
200 REM LOOP BACK TO HERE
290 X$="":MOVE P$(1),X$,1,1:MOVE C$(1),X$,1,1:REM
    NULL LONG STRINGS
300 DISP "ENTER A MESSAGE, FINISH WITH TWO
    <ENTER>'S : "
310 DISP :A$(1)="":INPUT A$(1)
320 IF A$(1)<>"" THEN MOVE *P$(1),A$(1),1,LEN
    A$(1):GO TO 310
350 !HERE WHEN MESSAGE IN P$(1)
360 FOR I=1 TO LEN P$(1):MOVE Z$,P$(1),I,1:GO SUB
    12000
365 DISP Y$;
370 MOVE *C$(1),Y$,1,1:IF F=1 THEN GO SUB 10000
380 NEXT I
400 !NOW GOT TARGET TEXT IN C$(1)
410 PRINT
420 PRINT "*** ENIGMA3 ***  INITIAL ROTOR
    SETTINGS: ";
430 FOR I=1 TO 5:PRINT USING 1250;IR(I);:NEXT I:
    PRINT
440 HP=0
450 FOR I=1 TO LEN C$(1) STEP 5:MOVE Z$,C$(1),I,5:
    PRINT Z$;" ";
460 HP=HP+6:IF HP>70 THEN PRINT :HP=0
470 NEXT I:PRINT :PRINT
490 GO TO 100
999 STOP
1000 !ROUTINE TO TEST ENCRYPTION AND DECRYPTION
     GONE TO AT L. 100
1010 NL=7:A$(3)="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
1020 FOR I=1 TO 5:P(I)=0:NEXT I
```

```
1030 PRINT TAB 20;"*** TESTING ENIGMA3 ***":PRINT
1040 PRINT "ALPHABET:";TAB 30;"TEST DECRYPT";TAB
     60;"ROTOR SETTINGS 1-5"
1050 PRINT :PRINT A$(3):PRINT
1060 !
1100 !LOOP HERE
1140 NL=NL+1:IF NL>60 THEN NL=0:PRINT CHR$
     252;CHR$ 12;CHR$ 31:REM NEW PAGE
1150 A$(1)=A$(3):A$(2)=""
1160 FOR I=1 TO LEN A$(1):Z$=MID$ (A$(1),I,1):GO
     SUB 12000
1170 A$(2)=A$(2)+Y$:NEXT I
1180 PRINT A$(2);TAB 30;:A$(1)=A$(2):A$(2)=""
1190 FOR I=1 TO LEN A$(1):Z$=MID$ (A$(1),I,1):GO
     SUB 12000
1200 A$(2)=A$(2)+Y$:NEXT I
1210 PRINT A$(2);TAB 60;
1220 FOR I=1 TO 5:PRINT USING 1250;P(I);:NEXT I:
     PRINT
1230 GO SUB 10000:GO TO 1100
1250 IMAGE 'X ZZ'
10000 !SUBR TO INCREMENT ROTOR POSITIONS
10010 I9=1
10020 P(I9)=P(I9)+1:IF P(I9)<26 THEN RETURN
10030 P(I9)=0:I9=I9+1:IF I9<6 THEN 10020
10040 RETURN
10999 !
11000 !SUBR TO LOAD ROTOR ARRAY W(5,26,2)
11050 RESTORE :RO=0
11060 READ A$(1):IF A$(1)="Z" THEN 11200
11070 READ A$(2):RO=RO+1
11080 FOR I9=1 TO 26:Z$=CHR$ (I9+64)
11090 SEARCH A$(1),1,Z$,P1:SEARCH A$(2),1,Z$,P2
11091 !SEARCH FUNCTION LOOKS FOR STRING Z$ IN
      STRING A$(1), STARTING
11092 !AT MID$(A$(1),1,1).  IF IT FINDS IT RETURNS
      P1=POSN OF Z$
11093 !IN A$(1), IF NOT RETURNS P1=0
11100 D1=MOD (P1-P2+52,26):D2=MOD (P2-P1+52,26)
11118 IF (W(RO,P1,1)<>0) OR (W(RO,P2,2)<>0) THEN
      DISP "ERROR":STOP
11119 REM THIS HAPPENS IF LETTER DUPLICATED
11120 W(RO,P1,1)=D2:W(RO,P2,2)=D1
11130 NEXT I9
11140 GO TO 11060
11199 !
11200 !HERE WHEN FINISHED
11201 RETURN
11999 !
12000 !SUBR TO ENCRYPT/DECRYPT CHAR Z$, RETURN Y$
```

```
12050 F=0:Z=ASC (Z$)-65:IF (Z<0)+(Z>25) THEN
      Y$="":RETURN
12055 F=1:REM TELLS MAIN ROUTINE TO ADVANCE ROTORS
12059 !FIRST ADVANCE THROUGH ROTORS 1-5 AND
      REFLECTOR
12060 FOR I9=1 TO 5
12070 Z=Z+W(I9,MOD (Z+P(I9),26)+1,1)
12080 NEXT I9
12100 !NOW BACK THROUGH THEM
12110 FOR I9=4 TO 1 STEP -1
12120 Z=Z+W(I9,MOD (Z+P(I9),26)+1,2)
12130 NEXT I9
12300 Y$=CHR$ (MOD (Z,26)+65):REM Y$=OUTPUT
      CHARACTER
12310 RETURN
19999 !
20000 !SUBR TO CHECK ALPHABETS
20010 RESTORE :ALPH=0
20020 READ A$(1):IF A$(1)="Z" THEN 20100
20025 OK=1:ALPH=ALPH+1:DISP "ALPHABET: ";ALPH;
20030 FOR I=65 TO 90:Z$=CHR$ (I):S=1:F=0
20040 SEARCH A$(1),S,Z$,A
20050 IF A<>0 THEN F=F+1:S=A+1:IF A<>LEN A$(1)
      THEN 20040
20055 IF F<>1 THEN OK=0:DISP " ";Z$;F;" ";
20060 NEXT I
20070 IF OK=1 THEN DISP "OK";
20080 DISP :GO TO 20020
20100 RETURN
```

The Enigma machine

Much more complex ciphers of this type have been solved, such as for example the Enigma machine used by the Germans during the Second World War. This produced in effect a polyalphabetic substitution with an enormously long key length. The starting position within the key was changed each day, but none the less the volume of traffic was such that, using an early computer, the cipher was finally broken.

Like all good inventions, the Enigma was basically a simple device, but of most ingenious design. It consisted of a keyboard, wired to a circular array of electrical contacts, and an array of 26 electric torch bulbs which lit up to display the letters of the alphabet.

Several drum-like rotors were provided, which were fitted into the machine side by side, between the circular array of contacts and a so-called reflector plate. On the sides of the drums, and on the reflector plate, there were also arrays of 26 contacts. Within each rotor, wires were fitted connecting the contacts on one side to those on the other more or less at random. The reflector plate had 13 wires connecting the 26 contacts together in pairs, also at random. Each rotor had a different random arrangement of wires, though of course the wiring was identical for the set of rotors and the reflector plates in each machine.

Pressing a key on the keyboard, say the letter A, sent an electric current to one of the fixed array of contacts. The current made its way through rotor no. 1, being moved round the circle a random amount, then through the other rotors in the same way. On reaching the reflector plate, the current was routed back to the last rotor via a different contact, and passed back to the fixed contacts via a different route. This would cause a bulb corresponding to a different letter to A to light up.

Supposing this letter to be M, suppose we then press the M key of the keyboard. This disconnects the M bulb from the M fixed contact, and passes current through it into the rotors. This will follow the reverse path through them, and emerge at the A contact, lighting the A bulb.

Used in this way, the machine produces a simple symmetrical monoalphabetic substitution cipher. The symmetrical feature is a considerable advantage in that encryption and decryption are the same process. If 'abcd' encrypts to 'QZWF', then typing in 'QZWF' will cause A, B, C and D bulbs to light in succession. However, no letter may encrypt to itself; this constitutes a slight weakness of the system, for the ciphertext will not then be completely and utterly random when analysed. The cryptanalyst will find a slight shortage of E's in the ciphertext, E being the commonest letter in German as in English, but this does not really tell him anything if he already knows that the ciphertext comes from an Enigma machine being used by Germans.

This disadvantage is more than offset by the great advantage of the machine, which is that the rotors are moved after encrypting each character. When one rotor has moved around 26 positions, the next is moved on one position and so on. With four rotors, this gives 26^4 or 456,976 different rotor settings before the machine repeats its cycle, that is 456,976 different monoalphabetic substitution ciphers each used only once in each message. Different parts of this cycle can be used by starting the rotors off in different positions.

In the 1940s the Enigma machine was generally regarded as producing an unbreakable cipher; certainly the Germans believed this, to their cost, though towards the end of the war they took measures – such as increasing the number of rotors – which suggest that they had some doubts as to its security. The number of combinations can also be increased by changing the positions of the rotors within the machine, and by adding a plugboard between the keyboard and the fixed contacts, which amounts to superenciphering with a simple monoalphabetic substitution. In spite of these measures, though, the British cryptanalysts at Bletchley Park were able to crack it, using a

combination of genius, teamwork, and the world's first electronic computer. Probably because it was convenient, the Germans continued to use Enigma to the end of the war, a fact which no doubt hastened its end.

The development of the first Enigma machines must have taken thousands of hours of skilled engineers' time, and the production line would have involved hundreds of people filing little bits of metal, screwing things together, dropping springs on the floor and swearing, and all the other activities which go with the manufacture of complex mechanical devices. Today, thanks to the microchip, a moderately skilled programmer can knock one up in a few hours; and as for the production line, all one needs to do is to copy tapes or disks. This is an excellent illustration of the general-purpose nature of modern computers. They are generalised machines, capable of doing anything according to the programming. Early computers were not programmed from a keyboard, punched cards, or anything like that; you went round the back of huge racks of equipment with a soldering iron and soldered wires between terminals. If the 'program' didn't work, you went round the back again, this time with wire-cutters, and started again. The idea of storing a program in memory, as opposed to hard-wiring it into the machine, was a revolutionary concept of Alan Turing's, the principal genius of Bletchley Park. It may seem obvious today, but it was not so in the 1940s, when a single bit of memory required a double-triode valve and associated components costing several shillings altogether, not to mention the power needed to run it. Today we can buy a chip containing thousands of bits of memory for a pound or two.

The other advantage of the stored program is that the programs can be made self-modifying, which gives still greater flexibility; the development of true machine intelligence will only come with the use of self-modifying programs. These will initially be quite simple, but will modify themselves in the course of 'education' so as to act in the required way. Here, of course, we are coming close to the simulation of the human brain.

ENIGMA3: a Basic listing

Philosophical digressions aside, no self-modifying program is required to simulate an Enigma machine. This is a very simple task for the Basic programmer, and a listing for one is given in

Alphabets produced by ENIGMA3

Plaintext alphabet:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
Rotor settings 1–4 and reflector:																											
00 00 00 00 00	M	F	J	X	I	B	U	K	E	C	H	Z	A	T	S	V	Y	Q	O	n	G	P	Y	D	W	L	t
01 00 00 00 00	Q	P	J	W	Z	K	L	u	O	C	F	G	V	T	I	B	A	X	Y	N	H	M	R	D	S	E	h
02 00 00 00 00	P	J	D	C	u	N	K	R	Z	B	G	O	Q	F	L	A	M	H	T	S	E	W	V	Y	X	I	e
03 00 00 00 00	J	V	M	I	N	X	L	K	D	A	H	G	C	E	P	O	s	T	Q	R	Y	B	Z	F	U	W	q
04 00 00 00 00	H	I	O	P	Z	T	S	A	B	M	W	X	J	Q	C	D	N	V	G	F	y	R	K	L	U	E	u
05 00 00 00 00	J	U	Q	L	M	K	I	N	g	A	F	D	E	H	V	X	C	Z	W	Y	B	O	S	T	P	R	i
06 00 00 00 00	U	X	e	O	C	N	Z	W	J	I	V	T	Y	F	D	R	S	P	Q	L	A	K	H	B	M	G	c
07 00 00 00 00	C	E	A	W	B	N	S	Z	O	K	j	P	T	F	I	L	Y	U	G	M	R	X	D	V	Q	H	k
08 00 00 00 00	Q	k	D	C	S	Y	M	Z	N	R	B	O	G	I	L	W	A	J	E	U	T	X	P	V	F	H	b
09 00 00 00 00	F	I	J	Z	S	A	H	G	B	C	Q	O	U	X	L	W	K	y	E	V	M	T	P	N	R	D	r

Appendix 1. To show how it works, the first few alphabets it produces are shown in the table on p.102.

The lower-case letters show how the message 'thequickbr' encrypts to NUUSY GEJKY. To decrypt, of course, the rotors are reset to 00 00 00 00 00, and repeating the process with NUUSYGEJKY, out comes 'thequickbr'.

Remembering that the people at Bletchley Park had to keep track of millions of such alphabets, we can appreciate the work that was involved. However, the sheer volume of German traffic was such that parts of the sequence of alphabets were used over and over again, and once these parts were identified a breakthrough could be made. Often the starting settings of the rotors for the day were transmitted in clear or in a low-grade cipher, and this made the task somewhat easier. In due course, Bletchley was able to reconstruct the internal wiring of the rotors, which was as good as possessing their own machine, though of course new rotors were introduced from time to time.

Returning to the ENIGMA3 listing, the rotor connections are given as DATA statements, each of which contains a mixed alphabet:

```
46 REM  ENTRY AND EXIT POINTS FOR
47 REM  LETTERS TO AND FROM ROTOR 1:
48 REM  ABCDEFGHIJKLMNOPQRSTUVWXYZ
49 REM  ──────────────────────────
50 DATA"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
52 DATA"MTREIVUWKPOGNBFAZCYDHJSLQX"
53 REM ROTOR 2
54 DATA"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
56 DATA"YLOGSAFMXITJEHNCRWZDVQKPBU"
57 REM ROTOR 3
58 DATA"QWERTYUIOPASDFGHJKLZXCVBNM"
60 DATA"POBEVUZXCKDQFARTINGSMLYJHW"
61 REM ROTOR 4
62 DATA"ETAOINSHRDLUGPVZKJBXYFCQWM"
64 DATA"ZYXWVUTSRQPONMLKJIHGFEDCBA"
71 REM REFLECTOR
72 DATA"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
74 DATA"TVMFNDJOSGUYCEHQPWIAKBRZLX"
```

The letters in these alphabets have no significance other than to indicate which contact on one side of the rotor is connected to which one on the other. Thus in lines 50–52, the A's indicate that contact 0 on the input/output side of rotor 1 is connected to contact 15 on the side in contact with rotor 2. The rotors are shown with all settings zero. Thus with setting 00 00 00 00 00, the letter A passes through rotor 1 via the wire labelled A to position 15 of rotor 2. Here, the wire labelled P takes it to position 23, where it enters rotor 2 and passes through the wire labelled B. This brings it back to position 2, where is passes to rotor 4 wire A. This takes it up to the rightmost position (25), where it passes through the reflector via the wire linking Z and X. Returning to rotor 4 at position 24, it takes wire C to position 22. It then goes through rotor 3, wire Y to position 5, through rotor 2 wire A to position 0, and finally through rotor 1 wire M to position 12, which corresponds to the letter M on the fixed contacts. So with this rotor setting A is changed to M, and by the reverse route M is changed to A.

If we now advance rotor 1 setting by one, this is equivalent to sliding lines 50 and 52 one position to the left, deleting the letters A and M from the beginnings and replacing them at the ends of the alphabets. If this is done, the letter A enters rotor 1 at wire B, which takes it to wire M of rotor 2. After passing circuitously through the other rotors and reflector plate it returns to wire B of rotor 2, which connects to wire R of rotor 1, which outputs the letter Q. Thus A and Q correspond to each other in alphabet 01 00 00 00 00 00.

When writing the data lines for the rotor connections, letters were entered pretty well at random as can be seen. However, care was taken that there were no straight-across connections (two letters the same one above the other in a data-line pair). Some arrangements of 'wires' will look exactly the same at different rotational positions of the rotor, and one should check for these in a serious working version of the program. This could be done by turning each rotor through all 26 positions, keeping the others stationary, and listing the alphabets. If any are the same, then the wiring of that rotor should be changed. Also, the wiring of the reflector should be symmetrical, i.e. if A connects to T then T should connect to A, and so on. Otherwise, there is considerable freedom in selecting the rotor wiring, and more rotors could easily be added, with only a slight increase in execution time.

Before any encrypting or decrypting can be done with this program, the array W(5,26,2) must be loaded from the data statements reprinted above. The loading is done by subroutine

11000. This array (which may be integer-type) is three-dimensional, the dimensions being as follows:

First subscript:	Rotors 1 to 4, 5 being reflector plate.
Second subscript:	1–26 correspond to contact numbers 0–25.
Third subscript:	1 for movement towards the reflector, 2 for movement away from it.
Content:	Shift round the circle of contacts modulo 26.

On the subject of subscripts, the Sharp PC-3201 used, like many machines, will not accept zero subscripts. When using modulo arithmetic, this can be inconvenient. In this case, the numbers run from 0 to 25 naturally, yet subscripts 1 to 26 must be used. This can cause confusion, 'bad subscript' errors and such-like. The most convenient (if not the fastest executing) solution is to work exclusively in the modulo numbers, adding '+1' within the brackets whenever an array variable appears in a statement. In the present case, this only applies to the second subscript of the W array. Subroutine 11000 takes each pair of rotor-wiring alphabets in turn, and in each of them locates the letters A, B, C etc. in turn, using the useful Sharp string search instruction. This is easily enough duplicated using standard Basic string instructions. The variables P1 and P2 then contain the positions of the two ends of the A, B, C etc. wire relative to the zero position. The differences of these are found, and put forward into the W array.

Consider for example the A wire of rotor 1. This runs from position 0 on the keyboard side to position 15 on the reflector side. This means that in the inward direction the wire produces a shift of +15 positions, and in the outward (away from reflector) direction the shift is -15 positions, which modulo 26 is the same as +11. Therefore, W(1,1,1), corresponding to rotor 1, position 0, inward direction is made equal to 15, and W(1,15,2) (position 15, outward direction) is set to 11. This is done for all 26 positions of all 4 rotors and for the reflector (W(5, ,)). The values of W(5,I,1) and W(5,I,2) should be the same for all I values 1 to 26 if the reflector alphabets are correct. Line 11118 checks for errors in the alphabet and stops if any are found.

Once the W array has been set up from the data alphabets, the 'machine' may be used. Subroutine 12000 encrypts character Z$, returning a character Y$. If Z$ is a non-alphabetic character, it ignores it and returns the flag F set to zero to inform the calling routine that the character was non-encryptable. The character Z$ is converted into the number Z, being 0 for A, 25 for Z. The number Z is then processed through each of the rotors in turn, being increased by the amount of shift at the position where it is in the rotor where it is, in the direction where it is going. This is easily done with the statement:

```
Z=Z+W(ROTOR,MOD (Z+P(ROTOR),26),1)
```

Having reached the reflector plate, it returns through the four rotors, this being executed for each one:

```
Z=Z+W(ROTOR,MOD(Z+P(ROTOR),26),2)
```

Here the final subscript is 2, because the movement is now outwards from the reflector. After all the shifting, the final value of Z mod 26 gives the output character which is assigned to Y$.

The array P(5) holds the positions of the four rotors and of the reflector, which also rotates in this version of the machine. It did not in the original Enigmas, but we added this for fun. Software is a lot easier to change than hardware! Advancing of the rotors is done by subroutine 10000, called after the encryption of each valid character.

Enigma variations

Reconstructing Enigma on a small business computer provided some amusement for idle hours, and enabled us to recapture some of the drama of the war-time years. In the bitter cold of the Russian front, German soldiers are huddled in the back of a truck, numbed fingers clumsily stabbing at the keys of the machine and copying down the dimly-glowing letters. A slip of paper is passed to the wireless operator, and soon the message is winging its way through the ether to Berlin. But – 'somewhere in England', huge aerials capture the feeble signal, and a bored listener suddenly seizes pencil and paper, scribbling down the faint morse with one hand, clutching earphones to his head with the other. A motor-cycle despatch rider, with blackout-dimmed headlight, roars through the night to Bletchley Park... within the hour, the signal is on Churchill's desk. Such nostalgic fantasies are all very well, but what relevance does Enigma have to modern cryptography?

A mere reproduction of the machine used by the Germans, be it a computer program or a physical machine, would find few buyers in the 1980s. For a start, nobody today would want a cipher which can only handle the letters of the alphabet. Now that every schoolboy has the facilities of Bletchley Park in his room, the security of a system using only a few rotors would be negligible, particularly once the 'wiring' of the rotors became generally known, which is inevitable. None the less, the rotor machine principle – used not only by Enigma, but also by other manufacturers – is easily adapted to the electronic computer, and it is fast in comparison to more secure systems such as the RSA cipher. No particular effort was made to maximise speed in the computer simulation – for example, subroutine 12000 should be moved to the beginning of the program – yet it still works at several characters per second. The speed decreases only in direct proportion to the number of rotors used, so there is no reason why a large number should not be employed.

A tentative specification for an updated computer Enigma would use the full ASCII character set of 127 characters at least – let's say a good round 256, convenient for 8-byte machines. Number of rotors? Why not 256 as well. This means that the total sequence length would be 256 to the power of 256, which is a number 617 decimal digits long. This number is so astronomical that if every hydrogen atom in the universe sent a message every second throughout the entire age of the universe from the Big Bang right up until the last moment before the Big Crunch, the chances are still infinitesimal that any part of the sequence would ever be used twice. From the point of view of sheer codebreaking, the system would be secure enough; but the problems remain that the rotor wiring will eventually become public knowledge, and that the rotor settings must be distributed. For such a system, the rotor settings would be a block of 256 8-bit bytes, one to define the position of each rotor. This information for obvious reasons cannot be sent using the cipher itself, and a cryptanalyst already possessing the rotor wiring details who obtains it can then read the messages as easily as his morning paper. Frequent changes of rotor wiring are more practical with electronic rotor machines than with physical ones; but here again, the wiring details must be sent over some channel between users of the system, and no communication channel these days can be assumed secure. Whichever way you look at it, a cipher system of this type cannot function unless key information is sent over insecure channels – be it rotor-wiring details or actual rotor settings. The same problem applies to all one-part cipher systems – a fact which led to the development of public-key (or two-part) ciphers.