CHAOCIPHER: SOLVING EXHIBITS 1 and 4.
{December 2010)

Author: Michael J. Cowan

ABSTRACT: Chaocipher was invented over 90 years ago by J F Byrne, who claimed it was unbreakable. Frustrated by a lack of interest in US official circles, Byrne published in 1953 large amounts of plain and matching ciphertext -- though not a description of his enciphering process -- to bear out his contention that the Chaocipher system was 'absolutely indecipherable'. A particular section issued a challenge to decipher an encrypted message for which he offered a prize of $5000. Nobody claimed the prize, neither in the 3 months of the challenge nor in the 55 years thereafter. Things changed in May 2010 when the Byrne family donated their archive to the National Cryptologic Museum, which described the enciphering system. Armed with this knowledge I describe how to solve the challenges posed by Byrne, using a computer.

1. Introduction.

92 years ago John F. Byrne invented a machine that enciphers plaintext to a ciphertext that is almost completely random. He called his cipher Chaocipher.

Byrne was convinced that Chaocipher was indecipherable. In the 1920's he offered his invention to numerous officials in the US State Department and War Department (where W.F.Friedman was involved) and later to the Navy Department in 1937. None of them was interested enough to adopt it. Frustrated by this lack of interest, he published a book 'The Silent Years' in 1953[1] with a chapter describing his views of the overwhelming merits of Chaocipher. He declined to reveal his machine or enciphering system but he supplied an immense amount of plain and matching ciphertext and challenged the world to break his cipher, offering a prize of $5000.

On the face of it, with so much plain and matching ciphertext, it should have been a simple affair to deduce how the enciphering was done. In the event there were no claimants from that day to this.

The years have taken their toll and both John F Byrne and his son, who helped in the Chaocipher project, have died without publishing details of the invention. But then in May 2010 members of the Byrne family very kindly donated two boxes of papers and artefacts to the National Cryptologic Museum[2], which explain the machine and the algorithm.

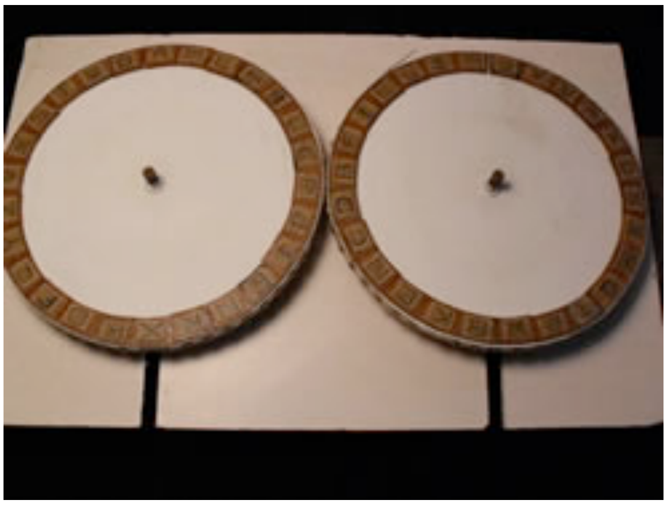
Using knowledge of how the Chaocipher machine works I devised computer algorithms to find Byrne's settings and to solve his challenges. I will explain how I did this, describe errors I found in Byrne's encipherments and reveal the so-far undeciphered parts of his challenges.

1

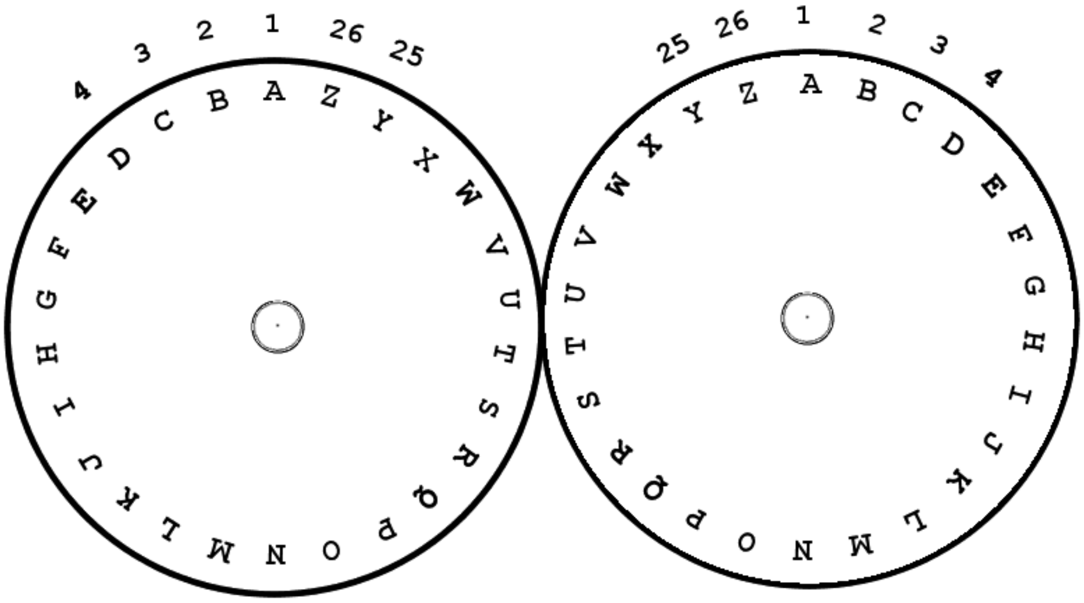
2

2. The Chaocipher algorithm and machine.

Byrne's basic machine comprised two wheels, each with the letters of the alphabet around their periphery. One wheel drove the other during encipherment. A mock-up built by John Byrne jr[3] is illustrated below.



The letters around the periphery could be adjusted. I have made the following diagram of the machine with the normal alphabet, which was a starting position sometimes used by Byrne.


4 3 2 1 26 25
A Z Y X W V U T S R Q P O N M L K J I H G F E D C B
25 26 1 2 3 4
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

The alphabet runs clockwise round the right wheel and anticlockwise round the left wheel, with the letter 'A' at position number 1 of each wheel. This arrangement can also be represented as below, the letter at the number 1 position always being shown first:

```
right ABCDEFGHIJKLMNOPQRSTUVWXYZ
left  ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

FIGURE 1

[3]

I am going to number all the positions on the wheel, as below, because as we shall see encipherment involves moving the letters to new position:

```
                1111111112222222
      12345678901234567890123456
right ABCDEFGHIJKLMNOPQRSTUVWXYZ
left  ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The diagram above shows some of these numbered positions. These numbered positions always remain the same, but with rotation of the wheels the letters opposite the numbers change.

To encipher the plain letter 'C':

rotate the right wheel to bring 'C' to position 1. Because the wheels are in contact, this also rotates the left wheel by a similar amount, and we get:

```
                1111111112222222
      12345678901234567890123456
right CDEFGHIJKLMNOPQRSTUVWXYZAB
left  CDEFGHIJKLMNOPQRSTUVWXYZAB
```

The letter at position 1 of the left wheel is now noted as the cipher letter, which on this occasion is also 'C'.

As a final step in the encipherment cycle, the wheels are disengaged and the letters on each wheel are permuted, though each wheel in its own way:

Right Wheel

before	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
after	26	1	2	14	3	4	5	6	7	8	9	10	11	12	13	15	16	17	18	19	20	21	22	23	24	25

Left Wheel

before	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
after	1	14	2	3	4	5	6	7	8	9	10	11	12	13	15	16	17	18	19	20	21	22	23	24	25	26

FIGURE 2

The permutations change the wheels to the following. Note that the leftmost letter always represents the letter at position 1 in the wheel - here 'D' is at position 1 of the right wheel and 'C' of the left:

```
                  1111111112222222
        12345678901234567890123456
right   DEGHIJKLMNOPQFRSTUVWXYZABC
left    CEFGHIJKLMNOPDQRSTUVWXYZAB
```

And that simple process is the secret of Chaocipher.

From now on I will just show the alphabets without the numbers, the leftmost letter of each alphabet being always at the number 1 position.

The way the permutations were actually made on Byrne's machine are described in Moshe Rubin's paper 'CHAOCIPHER REVEALED: THE ALGORITHM'

Moshe Rubin © 2 July 2010 (Updated 30 July 2010) which can be found at
http://www.mountainvistasoft.com/chaocipher/ActualChaocipher/Chaocipher-Revealed-Algorithm.pdf

A full step-by-step illustration of the encryption of the 10 letters 'CHAOCIPHER' is given in Appendix 1 and a computer program in C language for enciphering in Chaocipher is given in Appendix 2.

After enciphering just the ten letters CHAOCIPHER, the wheels become

```
rght  RDABCKEGILNOPQFSHTUMJVWXYZ
left  SHKTUVOMXYBCFGIJLNZAWPDEQR
```

and the shuffling effect of the permutation steps is already visible.

3. Solving Exhibit 1.

In his 1st Exhibit, Byrne presents 13,615 letters of ciphertext.

He only provides part of the equivalent plaintext, as follows:

-he tells us that the initial 5,500 letters are 100 repeats of the sentence
   ALLGOOD,QUICKBROWNFOXESJUMPOVERLAZYDOGTOSAVETHEIRPARTY.
To encipher the punctuation marks, Byrne represents a comma as 'Q' and a period as 'W';

-he does not give the next 262 plain letters;

-he gives 7,836 letters of the Declaration of Independence and the Gettysburg Address, again using letters to replace punctuation marks;

-he does not give the final 17 plain letters.

Downloads of the plain and ciphertexts are available at 'The Chaocipher Clearing House', excellently run by Moshe Rubin, at this web address:

http://www.mountainvistasoft.com/chaocipher/Chaocipher-ASCII-versions.htm

In the plaintext Moshe has helpfully included a '?' to denote each plain letter omitted by Byrne.

3.1 finding the letters in each wheel.

The challenge is to find the unknown plain letters by deciphering those cipher letters that match them. To do this we must find how Byrne setup the wheels - in other words, what were the letters around the periphery of the right and left wheels when he began to encipher Exhibit 1? With that knowledge we can decipher from start to finish of the ciphertext and find the unknown plain letters.

To find the setup wheels I developed a Maze algorithm in 2009[5] when working on another, sadly wrong, algorithm for Chaocipher. It uses

5

successive pairs of plain/cipher letters from Exhibit 1. The algorithm starts with no letters in the wheels and finds the way to fit each pair into the wheels so that there are no conflicts.

The starting pair is pt=A,ct=C (I am going to use the abbreviations pt and ct for plaintext and ciphertext) and these clearly can be fitted as follows:

```
       1.3.5.7.9.1.3.5.7.9.1.3.5.
right: A-------------------------
left : C-------------------------
```

and after the subsequent permutation, following the rules already given, the wheels become:

```
       1.3.5.7.9.1.3.5.7.9.1.3.5.
right: -------------------------A
left : C-------------------------
```

Now the next pair is pt=L,ct=L. Clearly the plain 'L' can be put anywhere in the right wheel, from position 2 to position 25, with the cipher 'L' inserted at the same position in the left wheel. Thus there are 24 possibilities. This is like entering a maze and finding that at the first junction there are 24 branches to choose from.

The algorithm takes a branch and progresses to the next junction where it tries to fit the next plain/cipher pair. Again it will find a number of possibilities and will chose one to follow.

Proceeding in this way a conflict is bound to be met at some stage when trying to fit the next pair. For example, the plain letter may already exist in the right wheel at a certain position but there may be no space in the left wheel at that position for the cipher letter. Or vice versa. In that case the algorithm chooses another branch from the last junction and attempts to proceed. If all branches at the last junction fail, then the algorithm backs up to the previous junction and tries another branch. This is very much like finding one's way through a maze by trial and error, which is why I call it a Maze algorithm.

Eventually the algorithm will successfully fill both wheels with letters after placing a certain number of pairs. That number will depend on having encountered all the letters of the alphabet in both plain and cipher texts.

For Exhibit 1 the wheels are filled with the 85th pair, where pt=R, ct=M. Here are the plain and cipher letters:

plain:
ALLGOODQQUICKBROWNFOXESJUMPOVERLAZYDOGTOSAVETHEIRPARTYWALLGOODQQUICKB
ROWNFOXESJUMPOVER

cipher:
CLYTZPNZKLDDQGFBOOTYSNEPUAGKIUNKNCRINRCVKJNHTOAFQPDPNCVLTVFICOTSSLWYY
IHBICFUTHXNUVKGIM

Here are the wheels after R/M have been fitted.

```
right wheel KTBUCOFIMVSHQGDPWXJYLZRANE
left wheel  IXLPJBQTKNRGUOFYCHZVEDMWAS
```

The computer program is given in Appendix 3. This program took 35 minutes to find this solution of the wheels on my 2.4 GHz computer .

3.2 Solving the complete Exhibit 1 ciphertext.

Now we can use this discovery of the letters in the wheels after the 85th pair to decipher the rest of the ciphertext. Of course Byrne gave us most of this decipherment, as explained earlier, except for two sections which we now can decipher:

(i) the 262 plain letters from positions 5500 to 5761, which Byrne omitted. Here is the decrypt:

```
ZENSHRINEDINTHISARCANUMQTOWHICHNONEWHODOESNOTPOSSESSTHEKEYMAYENTERQTH
EDECLARATIONOFINDEPENDENCEANDLINCOLNXSBEAUTIFULORATIONATGETTYSBURGARE
HEREREJINFORMEDWITHANINVISIBLEQINTANGIBLEANDIMPERCEPTIBLESOULWJWFWBYR
NEQANDMAPHJAGEFRWBEGUNAUGUSTSIXTEENQONENINETHREESEVENWZ
```

We can parse the decrypt, using the letters Byrne employed as punctuation (see p 279 Silent Years)

Z=paragraph	Q=comma	X=apostrophe
W=period	U=semicolon	(except QQ used in line 112)
V= colon	J= hyphen	H=dash

"Enshrined in this Arcanum, to which none who does not possess the key may enter, the Declaration of Independence and Lincoln's beautiful oration at Gettysburg are here re-informed with an invisible, intangible and imperceptible soul. J.F.Byrne, and **maphjagefr**. Begun August sixteen, one nine three seven."

I have highlighted the ten letters **maphjagefr** because they make no sense to me. Perhaps they represent the name of a person who helped Byrne with Chaocipher, and whose name Byrne has re-encoded with a private code - in a similar way as he has done in Exhibit 4, as we shall see later. These ten letters remain a mystery.

(ii) the 17 final plain letters, again omitted by Byrne, decipher to CORDIALTHANKSTOLO. To whom 'LO' refers is a matter of speculation, and is another mystery.

3.3 Finding the starting wheels.

In section 3.1 we found the letter orders in the wheels after the 85th pair was fitted. It is possible to 'wind back' from this position to find the letter orders for enciphering the initial (0'th) pair which was pt=A, ct=C.

Again this is done with a computer program, and I have given mine in Appendix 5. The algorithm is:

-with the wheels decoupled, reverse the permutations of Figure 2, which gives the correct letters in the wheels for the previous plain/cipher pair. It also puts the pair at position number 1 the right/left wheels respectively;

-with the wheels coupled, rotate left wheel to bring the last-but-one cipher letter to position number 1 of the left wheel, rotating the right wheel by a similar amount. This is the correct shift because the cipher letter at the top of the left wheel does not move during permutation (as can be seen in Figure 2).

From this we find the letters in the starting wheels were:

```
rw:  AYZNBQDSEFGHLWIKCMOPRTUVJX
lw:  CPEDQRSTIXYLMOZABFVGUHWJKN
```

We can see immediately that this is the correct setting for the first encipherment of plain A to cipher C in Exhibit 1.

Byrne has chosen this starting position by following a rather curious procedure. He has actually started with the normal alphabet that I have shown earlier in Figure 1. Then he has chosen a keyword. He has enciphered the first letter of the indicator using the right wheel (normal practice) but has enciphered the second and third letters using the left wheel and so on as follows:

```
Keyword                       THINKTHINK
Wheel used for plain letter:  RLLRLLLRRLR
```

The result is a letter order as follows:

```
rw: CMOPRTUVJXAYZNBQDSEFGHLWIK
lw: BFVGUHWJKNCPEDQRSTIXYLMOZA
```

which enciphers plain A to cipher C, after plain A in the right wheel is shifted to the top, following normal procedure, with the two wheels in contact. This is exactly what we discovered using the program in Appendix 3.

Byrne could have achieved the same effect by enciphering in the normal way the keyword TILNOYHIVK, but he presumably thought it more effective to choose an easily remembered keyword and a wheel order of RLLRLLLRRLR. To me this seems of doubtful value.

As we shall see Byrne has used a similar, though modified, procedure for his main challenge in Exhibit 4.

In this computer age it is very simple to find Byrne's keyword using a genetic algorithm, once the starting wheels have been found. A program for this purpose is given in Appendix 4.

4. Solving Byrne's challenge of Exhibit 4.

Exhibit 4 is the central challenge that Byrne posed, for which he offered a reward. He presented it in his book 'Silent Years', on pages 283 and 284, and included the plain and cipher texts under the heading of 'Exhibit 4'. These can again be downloaded from Moshe Rubin's 'The Chaocipher Clearing House'.

Byrne gives us 1,908 letters of ciphertext. He tells us this contains three sections:

(1) encipherment of a portion of a speech by General MacArthur, for which he gives 1,800 letters of plaintext.
(2) encipherment of "full and complete instructions to an initiate for decipherment", though we are not told the length nor where this is placed in the ciphertext.
(3) encipherment of a little over a dozen words that he has added somewhere within the last two lines of ciphertext (that include 93 letters).

Byrne says that anyone who can send him the decrypt of the words he has added will have provided evidence of being able to decipher the whole of the ciphertext. But his challenge will only be met when a decrypt of the whole of the ciphertext is provided.

4.1 Solving the end of Exhibit 4.

The computer solving program given in Appendix 2 for Exhibit 1, will also find the wheels used for Exhibit 4 given a stretch of matching plain and cipher texts. But here there is a rub. Byrne has not told us where the 1800 letters of plaintext fit with the 1,908 letters of ciphertext.

I began by trying to match the start of the plaintext to various starting positions in the ciphertext, but could not get a solution. I then turned to trying to match the final 300 letters of plaintext towards the latter part of the ciphertext.

Knowing that Byrne had added up to 93 letters after the plaintext of MacArthur's speech I set out by matching the start of the last 300 letters of plaintext with the start of the last 393 letters of ciphertext. After 37 seconds my program found this was an impossible choice to solve. So then I started 392 letters before the end, which again was rejected in short order. Finally on the fifth choice of cipher, starting 388 letters before the end, I got a solution after 1 min 40 secs:

```
pt=B  ct=D for position=178
right  ZHOVIKMGNUDFPQCT.JALYBRWSE
left   YUNEXKHRBSCPTAFLJQVZGDMOWI
```

These alphabets are correct at the 178th letter from the start of the plain and cipher texts I used (given in Appendix 5), which are plain B and cipher D. The alphabet for the right wheel is missing a letter – this is evidently 'X' and it is missing because it was not present in the 300 letters of plaintext.

Now one can encipher to the very end, inserting these alphabets and starting at the point in the ciphertext where plain B = cipher D (position 1500 for the plaintext and position 1520 for the ciphertext, as shown in Appendix 5). In this way the final unknown plain letters can be deciphered as:

```
THESEPOLITICALANDSOCIALCOTHEPEOPLESOFASIANOWSEEDAWNOFNEWOPPORTUNITYQA
NDPOLITICALFREEDOMZ
```

Parsing as before, we get

"These political and social co the peoples of Asia now see dawn of new opportunity, and political freedom."

Most of these words have been selected from MacArthur's speech (in bold below) but the others have been added by Byrne:

"These political and social co **the peoples of Asia now see dawn of new opportunity**, and **political freedom.**"

What Byrne intended with the word 'co' is not clear to me.

4.2 Solving the start of Exhibit 4.

Having now solved to the end of Exhibit 4, it is also possible to go backwards from the B-D entry point, permuting in reverse as in Appendix 5, and so deciphering all the way back to the beginning of the ciphertext. This reveals that MacArthur's speech is preceded by 20 letters of other plaintext, though these letters appear to be gibberish:

```
          1st 20 letters              MacArthur's speech
cipher  PMRGAHTMRZABMGAKMAAC VEHRNWQSJLDIWLUKKTGYRVSAEBPWFNRKPDPQTQJ
decrypt MHSCBUUFWREUEOJVNAFG BEYONDPOINTINGOUTTHESEGENERALTRUISMSQIS
```

To me the gibberish was inexplicable until reading an explanation that I found in one of the papers from the Byrne archive, posted at the National Cryptologic Museum[1]. This indicated that the initial 20 letters of ciphertext are not Chaocipher at all, but are the result of a monoalphabetic encryption with what Byrne called his 'indicating key' or 'private code'.

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

The first 20 letters are thus deciphered as follows:

```
ciphertext PMRGAHTMRZABMGAKMAAC
decrypt    TLCHRAOLCIRPLHRELRRZ.
```

The decrypt at first glance is still gibberish, though perhaps one can make out the word 'CHAOCIPHER' within it. But the Byrne paper explains that the decrypt is in fact the instructions for setting up the machine, after initially setting the wheels with the normal alphabet and turning them to bring the A's to the top position (as in Fig 1):

```
TL  - turn the left wheel to bring  'T' to the top;
CHR - turn the right wheel to bring 'C' to the top and then permute;
               ditto                'H'        ditto
AOL - turn the left wheel to  bring 'A' to the top and then permute;
               ditto                'O'        ditto
CIR - turn the right wheel to bring 'C' to the top and then permute;
               ditto                'I'        ditto
```

and so on until 'Z' is reached. This letter signifies the end of instructions. At that stage the wheels are as follows:

```
rght  STUKVWYZAFCDLXEHIJMNBOPQGR
left  PFGTHXIWJKLADESNOQRUVYZMBC
```

Now the machine is ready to encipher the rest of the plaintext. As shown above, after the initial 20 letters the next plain letter is

'B' that enciphers to 'V', in accordance with the wheel alphabets just shown.

We see here how Byrne set-up the machine to encipher Exhibit 4. He chose a keyword and a pattern of using either the right or the left wheel to represent the letters of this keyword. He then created a shorthand form of instructions, enciphered them with his private code and in the case of Exhibit 4 put the result as the initial cipher letters of his message. Finally he setup the wheels with normal alphabets, rotated the wheels to bring ‘A’ to position number 1 and carried out the instructions, creating nicely mixed alphabets on the wheels. These he used to begin enciphering his plaintext message.

(In Exhibit 1, as explained earlier) Byrne used a similar setup procedure but did not include the setup 'code' as the initial letters of the ciphertext.

Byrne expected a successful challenger to decrypt the whole of number four exhibit, including the first 20 cipher letters that as we have seen were not enciphered by Chaocipher at all. Byrne wrote on p 284 Of Silent Years that proof of successful decryption will only be accepted by "being able to decipher the whole of number four exhibit". In other words the successful candidate would have two different ciphers to solve.

Now it was clear to me why I could not solve the alphabets from the start of the ciphertext – because the start of the ciphertext was not Chaocipher! But it did not explain failure at start positions further into the ciphertext. By comparing the decrypt obtained from the discovered alphabets with the plaintext given by Byrne, I saw that Byrne had changed the letters he used for punctuation from those described on p 279 of 'Silent Years' and described earlier in this article. Instead of representing an apostrophe as 'X' and a hyphen as 'J', he inverted the substitutions. That was enough to disrupt my solver, which was putting an 'X' into the right wheel at a position where Byrne was using a 'J'. This was an unexpected example of one of the weaknesses of Chaocipher.

Some comments on the value of Chaocipher.

As a matter of interest, in Exhibits 1 and 4 Byrne termed position number 1 as the 'Zenith' and position 14 as the 'Nadir'. Actually the same ciphertexts will be obtained whichever particular position is used as the Zenith as long as:

-the Nadir is at the Zenith position plus 13 (mod 26). So if the Zenith is at position number 1 then the Nadir is at position 14.
-the same positions for Zenith and Nadir are used for both wheels.

Chaocipher is a clever idea because part of the enciphering alphabet is changed for every letter. The part that changes depends on the plaintext letter being enciphered. Since the latter is unknown to the cryptanalyst, and is in essence random, the changes appear to be random also. Consequently the security of Chaocipher is impressive. A break usually requires 80 or more letters of plaintext, far more than will be available from a probable word. A depth is of limited use because different plaintexts will quickly create totally different permutations.

On the negative side, Chaocipher was an extremely tedious cipher to operate, with the painstaking task of rearranging selected letters in both wheels after every encipherment. John Byrne jr commented that "perfect accuracy was essential when enciphering a message because one error would distort the rest of the cipher". He had "memories of tense and tedious hours striving for perfect accuracy while working with my father and his 'contraption' at the dining room table."[7]

To try and mechanise the process, Byrne had worked for 6 months with a draughtsman in 1920 to produce a design and engineering drawings for a machine with a keyboard, but its estimated cost was so high that it was never built. (This will be the subject of a futute paper). No doubt these negative factors were responsible for the lack of interest from those whom Byrne approached and for their rejection of Chaocipher.

When looked at overall in Byrne's era, the security advantage of Chaocipher is nullified because encipherment is slow and difficult, and any error disrupts the rest of the message. In that sense Chaocipher had little value compared with other systems then available.

In the present era the practical drawbacks can be overcome by using a computer. But on the other hand a computer considerably enhances the cryptanalytic potential. I expect that, under normal traffic conditions, algorithms will be developed to break Chaocipher with 'probable words' of reasonable length, such as the 20 to 30 letters that were available during WW2 for breaking ENIGMA. Partial solutions may then possibly be extended by Brute Force. But at the moment this is all speculation on my part and the feat has yet to be accomplished in practice.

Acknowledgements

I acknowledge with thanks the generosity of the National Cryptologic Museum for releasing papers from the Byrne Archive, to their Librarian Rene Stein for her sterling efforts in copying and scanning these documents and to David D'Auria for the photo of the enciphering machine. Also to Moshe Rubin for his excellent Chaocipher Clearing Site, where there exists a fund of invaluable information for the enthusiast to draw on.

Appendix 1 – example of step by step encipherment.

In this case plain CHAOCIPHER is ciphered to CGYLYCJZMS, starting with normal alphabets in the wheels, with 'A' at the top (that is at position number 1).

```
wheels at start
right ABCDEFGHIJKLMNOPQRSTUVWXYZ
left  ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

7

```
engage wheels, move plain to top of right wheel   pt=C  ct=C

right CDEFGHIJKLMNOPQRSTUVWXYZAB
left  CDEFGHIJKLMNOPQRSTUVWXYZAB
---------------------------------
disengage and permute the wheels
right DEGHIJKLMNOPQFRSTUVWXYZABC
left  CEFGHIJKLMNOPDQRSTUVWXYZAB

engage wheels, move plain to top of right wheel   pt=H  ct=G

right HIJKLMNOPQFRSTUVWXYZABCDEG
left  GHIJKLMNOPDQRSTUVWXYZABCEF
---------------------------------
disengage and permute the wheels
right IJLMNOPQFRSTUKVWXYZABCDEGH
left  GIJKLMNOPDQRSHTUVWXYZABCEF

engage wheels, move plain to top of right wheel   pt=A  ct=Y

right ABCDEGHIJLMNOPQFRSTUKVWXYZ
left  YZABCEFGIJKLMNOPDQRSHTUVWX
---------------------------------
disengage and permute the wheels
right BCEGHIJLMNOPQDFRSTUKVWXYZA
left  YABCEFGIJKLMNZOPDQRSHTUVWX

engage wheels, move plain to top of right wheel   pt=O  ct=L

right OPQDFRSTUKVWXYZABCEGHIJLMN
left  LMNZOPDQRSHTUVWXYABCEFGIJK
---------------------------------
disengage and permute the wheels
right PQFRSTUKVWXYZDABCEGHIJLMNO
left  LNZOPDQRSHTUVMWXYABCEFGIJK

engage wheels, move plain to top of right wheel   pt=C  ct=Y

right CEGHIJLMNOPQFRSTUKVWXYZDAB
left  YABCEFGIJKLNZOPDQRSHTUVMWX
---------------------------------
disengage and permute the wheels
right EGIJLMNOPQFRSHTUKVWXYZDABC
left  YBCEFGIJKLNZOAPDQRSHTUVMWX

engage wheels, move plain to top of right wheel   pt=I  ct=C

right IJLMNOPQFRSHTUKVWXYZDABCEG
left  CEFGIJKLNZOAPDQRSHTUVMWXYB
---------------------------------
disengage and permute the wheels
right JLNOPQFRSHTUKMVWXYZDABCEGI
left  CFGIJKLNZOAPDEQRSHTUVMWXYB

engage wheels, move plain to top of right wheel   pt=P  ct=J

right PQFRSHTUKMVWXYZDABCEGIJLNO
left  JKLNZOAPDEQRSHTUVMWXYBCFGI
---------------------------------
disengage and permute the wheels
right QFSHTUKMVWXYZRDABCEGIJLNOP
```

```
left  JLNZOAPDEQRSHKTUVMWXYBCFGI

engage wheels, move plain to top of right wheel   pt=H  ct=Z

right HTUKMVWXYZRDABCEGIJLNOPQFS
left  ZOAPDEQRSHKTUVMWXYBCFGIJLN
--------------------------------
disengage and permute the wheels
right TUMVWXYZRDABCKEGIJLNOPQFSH
left  ZAPDEQRSHKTUVOMWXYBCFGIJLN

engage wheels, move plain to top of right wheel   pt=E  ct=M

right EGIJLNOPQFSHTUMVWXYZRDABCK
left  MWXYBCFGIJLNZAPDEQRSHKTUVO
---------------------------------
disengage and permute the wheels
right GILNOPQFSHTUMJVWXYZRDABCKE
left  MXYBCFGIJLNZAWPDEQRSHKTUVO

engage wheels, move plain to top of right wheel   pt=R  ct=S

right RDABCKEGILNOPQFSHTUMJVWXYZ
left  SHKTUVOMXYBCFGIJLNZAWPDEQR
--------------------------------
------------------------------------------------------------
```

APPENDIX 2

```
/*
 Discover alphabets from plain and ciphertext.
 Includes initial pt & ct for solving Exhibit 1.

*/

 #include <conio.h>
 #include <fstream.h>

 char rw[500][30],lw[500][30],plain[1000],code[1000];
 char primus,inter,rb[30],lb[30];
 char rt[30],lt[30];

 int j,k,m,n,p,q,t,x,y,z;
 int len,flag,move,ptr[500],flag_r,flag_l;
 int nop,adv;
 int noo[500],pos[500][30];
 int score,bestscore;
 int len_r,len_l;
        //.....these arrays are for permuting the alphabets.....
 int index_r[30] ={0,2,3,5,6,7,8,9,10,11,12,13,14,15,4,16,17,18,19,
                  20,21,22,23,24,25,26,1};
 int index_l[30] ={0,1,3,4,5,6,7,8,9,10,11,12,13,14,2,15,16,17,18,
                 19,20,21,22,23,24,25,26};

 double nok;
 time_t start_time,end_time;
  float lapse;

  void get_options(void);
  void permute(void);
  void get_score(void);


 main()

 {



strcpy(plain,"ALLGOODQQUICKBROWNFOXESJUMPOVERLAZYDOGTOSAVETHEIRPARTYW
ALLGOODQQUICKBROWNFOXESJUMPOVER");

strcpy(code,"CLYTZPNZKLDDQGFBOOTYSNEPUAGKIUNKNCRINRCVKJNHTOAFQPDPNCVL
TVFICOTSSLWYYIHBICFUTHXNUVKGIM");


    start_time=time(NULL);

   len=strlen(plain);
   len_r=len_l=0;   flag_r=flag_l=0;
   for(j=0;j<500;j++) ptr[j]=0;

   //             1.3.5.7.9.1.3.5.7.9.1.3.5.
   strcpy(rw[0],"*..........................");//right wheel
   strcpy(lw[0],"*..........................");//left wheel

   nop=26;//nr of positions in alphabets

   //..put 0'th pair into wheels in permed position
   rw[0][nop]=plain[0];  lw[0][1]=code[0];
   adv=1; //flag to advance to next node
   j=0;
   cout<<"start    "; for(m=1;m<nop+1;m++) cout<<rw[j][m];
   cout<<endl;
   cout<<"          "; for(m=1;m<nop+1;m++) cout<<lw[j][m];
   cout<<endl<<endl;
   j=0;
   while(j<len-1)
    {

         j++;    // position in text
         nok++;  //counter
         if(j<0)
           {cout<<"Run failed - review assumptions"<<endl; getch();}
         //...get last wheels, permed....
         for(m=1;m<nop+1;m++)
              {rw[j][m]=rw[j-1][m]; lw[j][m]=lw[j-1][m];}

         if(adv>0)  //advance to next node
             { get_options(); ptr[j]=-1;}

         if(noo[j]>0)  // nr of options for placement
             {
              ptr[j]++; // pointer to options
              if(ptr[j]<noo[j])
                {
                 //..put the pair into current alphabets..
                 rw[j][pos[j][ptr[j]]]=plain[j];
                 lw[j][pos[j][ptr[j]]]=code[j];
                 get_score();  // nr of letters in current alpha't
                 permute();
```

```
               adv=1;     // flag to go forward to next node
              }
           else                //ptr advanced too far
             {j=j-2; adv=0;}  //so go back to previous node
          }
       else {j=j-2;adv=0;}  //no options,back to previous node
   }

cout<<endl<<"end";
getch();

return 0;
}

//----------------------------------------------------------

 void get_options(void)
   {
     int m,x;
     //...determine whether pt is in rt whl and if ct in lt whl,,,,
       flag_r=flag_l=0;
       for(x=1;x<nop+1;x++)
             if(plain[j]==rw[j][x]) {flag_r=x;break;}
       for(x=1;x<nop+1;x++)
             if(code[j]==lw[j][x]) {flag_l=x;break;}

       if(flag_r>0 && flag_l>0)   //pt and ct present in rw and lw
           if(flag_r!=flag_l) noo[j]=0;  // different positions so
                                         // no options
           else {noo[j]=1; pos[j][0]=flag_r; } // else just 1 option

       else if(flag_r>0 && flag_l==0) //pt exists, ct doesn't
          {
           if(lw[j][flag_r]=='.')
             {noo[j]=1; pos[j][0]=flag_r;}//OK if space for ct
           else noo[j]=0;
          }

       else if(flag_l>0 && flag_r==0)    //ct exists, pt doesn't
           {
            if(rw[j][flag_l]=='.')
              {noo[j]=1; pos[j][0]=flag_l;}//OK if space for pt
            else noo[j]=0;
           }
       else     // a number of slots free - list them in pos[j][]
           {
           noo[j]=0;  //nr of options
           for(p=1;p<nop+1;p++)
             if(rw[j][p]=='.' && lw[j][p]=='.')
               {pos[j][noo[j]]=p; noo[j]++;}

           }

   }
//.......................................................
 void permute(void)
   {
     int m;
         //......move pt & ct to top of wheel ...

         for(m=1;m<nop+1;m++) if(rw[j][m]==plain[j]) break;
```

```
          //cout<<"m="<<m<<endl;
          for(p=1;p<nop+1;p++)
            {
             x=p+m-1; if(x>nop) x-=nop;
             rb[p]=rw[j][x]; lb[p]=lw[j][x];
            }


          //...permute the buffers back into the wheels....
           for(m=1;m<nop+1;m++) rw[j][m]=rb[index_r[m]];
           for(m=1;m<nop+1;m++) lw[j][m]=lb[index_l[m]];


    }
  //.........................................
  void get_score(void)
    {
       score=0;
             for(m=1;m<nop+1;m++)
               {
                if(rw[j][m]!='.') score++;
                if(lw[j][m]!='.') score++;
               }

       if(score>bestscore) //display best results to date
        {
         bestscore=score;
         end_time=time(NULL);   lapse=end_time-start_time;
         cout<<"time="<<lapse<<"  nok="<<nok<<" position="<<j<<"
score="<<score<<"  pt="<<plain[j];
         cout<<"  ct="<<code[j]<<endl;
         cout<<"right wheel ";
         for(m=1;m<nop+1;m++) cout<<rw[j][m];cout<<endl;
         cout<<"left wheel  ";
         for(m=1;m<nop+1;m++) cout<<lw[j][m]; cout<<endl;
         cout<<endl;
        }

    }
---------------------------------------------------------------------
```

Appendix 3

```
/*
 Wind back wheels from 85th position to find
 starting alphabets
*/

 #include <conio.h>
 #include <fstream.h>

 char rw[30],lw[30],plain[1000],code[1000];
 char primus,inter,rb[30],lb[30],srb[30],slb[30];
 char rt[30],lt[30];

 int j,k,m,n,p,q,t,x,y,z;

 int shift,start,pos,len;
        //.....these arrays are for permuting the alphabets.....
```

```
             //old pos  0 1 2 3 4 5 6 7 8 9 10
11.......................................25.
  int new_pos_r[30] =
{25,0,1,13,2,3,4,5,6,7,8,9,10,11,12,14,15,16,17,18,19,20,21,22,23,24
};


                //     26 1 2 14 3 4 5 6 7  8  9 10 11 12 13 15 16 17
18 19 20 21 22 23 24 25

  int new_pos_l[30] =
{0,13,1,2,3,4,5,6,7,8,9,10,11,12,14,15,16,17,18,19,20,21,22,23,24,25
};


  void get_options(void);
  void permute(void);
  void get_score(void);


 main()

 {




strcpy(code,"CLYTZPNZKLDDQGFBOOTYSNEPUAGKIUNKNCRINRCVKJNHTOAFQPDPNCVL
TVFICOTSSLWYYIHBICFUTHXNUVKGIM");
   len=strlen(code);


   strcpy(rw,"KTBUCOFIMVSHQGDPWXJYLZRANE");  //for exh1 ending pt=R,
ct=M
   strcpy(lw,"IXLPJBQTKNRGUOFYCHZVEDMWAS");


   start=len-1;

   for(pos=start-2;pos>-1;pos--)
    {
         //....depermute .....
        cout<<"step 1, depermute"<<endl;
        for(m=0;m<26;m++) rb[m]=rw[ new_pos_r[m] ];
        for(m=0;m<26;m++) lb[m]=lw[ new_pos_l[m] ];
        for(m=0;m<26;m++) cout<<rb[m]; cout<<endl;
        for(m=0;m<26;m++) cout<<lb[m]; cout<<endl<<endl; //getch();

        //....find position of code[pos]...
        cout<<"step 2, shift alphabets so code["<<pos<<"] at lw[0]
="<<code[pos]<<endl;
        for(shift=0;shift<26;shift++) if(lb[shift]==code[pos]) break;

        for(m=0;m<26;m++)
          {
           x=m+shift;x=x%26;
           rw[m]=rb[x]; lw[m]=lb[x];
          }
        for(m=0;m<26;m++) cout<<rw[m]; cout<<endl;
        for(m=0;m<26;m++) cout<<lw[m]; cout<<endl<<endl; //getch();



    }
                //....make final depermute.....
        cout<<"final depermute"<<endl;
        for(m=0;m<26;m++) rb[m]=rw[ new_pos_r[m] ];
        for(m=0;m<26;m++) lb[m]=lw[ new_pos_l[m] ];
        for(m=0;m<26;m++) cout<<rb[m]; cout<<endl;
        for(m=0;m<26;m++) cout<<lb[m]; cout<<endl<<endl;

 cout<<endl<<"end";
 getch();

 return 0;
 }

//-----------------------------------------------------------
```

Appendix 4

```
/*
 genetic 1

*/

 #include <conio.h>
 #include <fstream.h>

 char rw[30],lw[30],alpha[60000][20],code[1000],beta[20];
 char primus,inter,rb[30],lb[30];
 char target_r[30],target_l[30];;

 int j,k,m,n,p,q,t,x,y,z,r1,xx,yy;
 int len,flag,move,pop,repeat;


 int score,bestscore,scorea[60000];

        //.....these arrays are for permuting the alphabets.....

             //old pos  0 1 2 3 4 5 6 7 8 9 10
11.......................................25.
  int new_pos_r[30] =
{25,0,1,13,2,3,4,5,6,7,8,9,10,11,12,14,15,16,17,18,19,20,21,22,23,24
};


               //     26 1 2 14 3 4 5 6 7  8  9 10 11 12 13 15 16 17
18 19 20 21 22 23 24 25

  int new_pos_l[30] =
{0,13,1,2,3,4,5,6,7,8,9,10,11,12,14,15,16,17,18,19,20,21,22,23,24,25
};

  void get_random_alpha(void);
  void decrypt(void);
  void decrypt_beta(void);
  void get_score(void);


 main()

 {
   randomize();
                  //0.2.4.6.8.0.2.4.6.8.0.2.4.
   strcpy(target_r,"CMOPRTUVJXAYZNBQDSEFGHLWIK");
   strcpy(target_l,"BFVGUHWJKNCPEDQRSTIXYLMOZA");


   len=10;    bestscore=0;  pop=60000;

   for(j=0;j<pop;j++)
     {
          get_random_alpha();
          decrypt();
          get_score();
          scorea[j]=score;
    }

  repeat=1;
  while(repeat>0)
     {
       xx=random(pop);
       yy=random(pop);
       r1=random(len);
       for(m=0;m<r1;m++) beta[m]=alpha[xx][m];
       for(m=r1;m<len;m++) beta[m]=alpha[yy][m];
       decrypt_beta();
       get_score();
       if(score>scorea[xx])
          {for(m=0;m<len;m++) alpha[xx][m]=beta[m];
scorea[xx]=score;}
       if(score>scorea[yy])
          {for(m=0;m<len;m++) alpha[yy][m]=beta[m];
scorea[yy]=score;}
     }

 cout<<endl<<"end";
 getch();

 return 0;
 }

//-------------------------------------------------------

 void get_random_alpha(void)
   {
     for(p=0;p<len;p++)
       {x=random(26); alpha[j][p]=x+65;}

   // for(p=0;p<len;p++)cout<<alpha[j][p]; cout<<endl; getch();
   }
 //.....................................................

 void decrypt(void)
    {
     int k,p,x;
   //         01.3.5.7.9.1.3.5.7.9.1.3.5
   strcpy(rw,"ABCDEFGHIJKLMNOPQRSTUVWXYZ");//right wheel
   strcpy(lw,"ABCDEFGHIJKLMNOPQRSTUVWXYZ");//left wheel



   for(k=0;k<len;k++)
     {
       for(move=0;move<26;move++) if(alpha[j][k]==rw[move]) break;

       //...now move both wheels by this amount & copy into
buffer.....
       for(p=0;p<26;p++)
         {
          x=p+move; x=x%26;
          rb[p]=rw[x]; lb[p]=lw[x];
         }




       //...now permute.....

       for(p=0;p<26;p++) rw[new_pos_r[p]]=rb[p];
       for(p=0;p<26;p++) lw[new_pos_l[p]]=lb[p];

     }


    }
  //....................................................
  void get_score(void)
    {
      int z;

      score=0;
      for(z=0;z<26;z++) if(rw[z]==target_r[z]) score++;
      for(z=0;z<26;z++) if(lw[z]==target_l[z]) score++;

      if(score>bestscore)
        {
         bestscore=score;
         cout<<"score="<<score<<" keyword= ";

         for(m=0;m<len;m++) cout<<beta[m];cout<<endl;
         cout<<"right wheel "; for(z=0;z<26;z++)
cout<<rw[z];cout<<endl;
         cout<<"left wheel  "; for(z=0;z<26;z++)
cout<<lw[z];cout<<endl;
          cout<<endl;
        }
    }

  //..............................................

  void decrypt_beta(void)
    {
     int k,p,x;

   //         01.3.5.7.9.1.3.5.7.9.1.3.5
   strcpy(rw,"ABCDEFGHIJKLMNOPQRSTUVWXYZ");//right wheel
   strcpy(lw,"ABCDEFGHIJKLMNOPQRSTUVWXYZ");//left wheel



   for(k=0;k<len;k++)
     {
       for(move=0;move<26;move++) if(beta[k]==rw[move]) break;

       //...now move both wheels by this amount & copy into
buffer.....
       for(p=0;p<26;p++)
         {
          x=p+move; x=x%26;
          rb[p]=rw[x]; lb[p]=lw[x];
         }




       //...now permute.....

       for(p=0;p<26;p++) rw[new_pos_r[p]]=rb[p];
       for(p=0;p<26;p++) lw[new_pos_l[p]]=lb[p];

     }


    }
  //.......................................................
```

Appendix 5

Sections of plain and ciphertext that yield after 78 letters the wheels:

```
pt=B  ct=D for position=178
right  ZHOVIKMGNUDFPQCT.JALYBRWSE
left   YUNEXKHRBSCPTAFLJQVZGDMOWI
```

plaintext, beginning at position 1500 from start of Exhibit 4:
INTHEDEVASTATIONLEFTINWARJSWAKEZWORLDIDEOLOGIESPLAYLITTLEPARTINASIANT
HINKINGANDARELITTLEUNDERSTOODWWHATTHEPEOPLESTRIVEFORISTHEOPPORTUNITYF
ORALITTLEMOREFOODINTHEIRSTOMACHSQALITTLEBE

ciphertext, beginning at position 1520 from start of Exhibit 4:
TZOFEPLGFIHADTYQAAYIYIOMWXYQHPGXTVOFXKHBNOFWLNKEFGJUWWPMHSMYDHHAJARKZ
JDVJYOMAEZRGSSTDCUOCEPBYILUOGBTVHDMRBNIDMGUHWEHZAGRQKOTCQEASCVDECNEPL
GGNWTRHXVVHLYLUQLIKHSZOJEIXBHWPSTAWCEVDYDW

References.

[1] http://www.mountainvistasoft.com/chaocipher/Silent-Years-Chapter-21-Chaocipher.pdf
[2] The National Cryptographic Museum has kindly provided a direct link to many items of interest donated by the Byrne family, including all aspects referred to in this paper:
http://www.nsa.gov/about/cryptologic_heritage/museum/index.shtml

[3] This scanned photo, of an artefact donated by the Byrne family, was taken by David D'Auria and is kindly provided with permission by the National Cryptologic Museum Foundation.
[4] For plain and ciphertexts of all Byrne's Exhibits see the summary made by Moshe Rubin at
 http://www.mountainvistasoft.com/chaocipher/Chaocipher-ASCII-versions.htm
[5] See Progress report Nr 13 at Moshe Rubin's 'The Chaocipher Clearing Site" at
http://www.mountainvistasoft.com/chaocipher/chaocipher-013.htm
[6]
http://www.nsa.gov/about/_files/cryptologic_heritage/museum/library/macarthur_speech.pdf
[7] John Byrne, Cipher A. Deavours and Louis Kruh. "Chaocipher enters the computer age when its method is disclosed to Cryptologia editors". Cryptologia, 14(3): 193-197. Can be ordered at
http://www.informaworld.com/smpp/content~db=jour~content=a741902642. Retrieved 10/12/2010.