Ciphertext-Only Cryptanalysis of Hagelin M-209 Pins and Lugs

George Lasry Mishmar Hayarden 5 Givataim, 5358205 Israel george.lasry@gmail.com Nils Kopal University of Kassel Pfannkuchstr. 1 34121 Kassel, Germany nils.kopal@uni-kassel.de

Arno Wacker University of Kassel Pfannkuchstr. 1 34121 Kassel, Germany arno.wacker@uni-kassel.de

Abstract

The Hagelin M-209 was a mechanical encryption device used by the US Army during World War 2 and the Korean War, as well as by other armies and in embassy settings. In this paper, we present a 4-stage hillclimbing algorithm for recovering the full key settings from ciphertext only. The two first stages are based on a divide-and-conquer approach developed by Geoff Sullivan. In our performance evaluation, we show that our method is able to recover key settings from messages as short as 750 letters, compared to 2000 - 2500 letters with prior methods. With this method, we solved several M-209 ciphertext-only challenges, including the final exercises in W. Barker's 1977 book, Cryptanalysis of the Hagelin Cryptograph.

Keywords: Hagelin, M-209, Hillclimbing, Ciphertext-Only Attack, Cryptanalysis, Cryptographic Challenge, NSA

1 Introduction

The trigger for our research was an intriguing story published by Dennis Ritchie in 2000 [19]. The story also appeared in Cyberpunk, Outlaws and Hackers on The Computer Frontier by Katie Hafner and John Markoff in 1991 [9]. Dennis Ritchie (1941-2011) is best known as the inventor of the UNIX operating system (together with Ken Thomson), and of the C programming language. According to his account, in the late 1970s, James Reeds, Robert Morris and Dennis Ritchie developed a ciphertextonly method for recovering keys from Hagelin M-209 messages. Ritchie provides some details about the method, and why it was not published. Robert Morris (1932-2011) was an early contributor to the UNIX operating system, with a special interest in cryptography and in the M-209 device in particular. In 1978, he published a paper describing a manual method for recovering Hagelin M-209 key settings from known plaintext [14]. James Reeds was at UC Berkeley at the time, and he later joined Bell Labs. According to Ritchie, their ciphertext-only method was able in most cases to recover key settings from an encrypted message with 2500 letters. It could also recover keys from only 2000 letters in half of the cases. The first part – the recovery of the pin settings, was statistical in nature and was developed by Reeds. The second part, the recovery of the lugs setting, was more combinatorial in nature and was based on Morris's prior work as well as on Reeds's ideas. Ritchie wrote the software code to implement and test the method. The trio also wrote a paper which they submitted to Cryptologia [17]. According to Ritchie, the manuscript was also sent to the NSA for review. After some time he and Morris received a visit from a "retired gentleman from Virginia". This gentleman suggested that the publication of the paper may "cause difficulties" for the US and for other countries, who may still be using similar equipment. At his request, the publication was indefinitely postponed. Morris later joined the NSA and became chief scientist at the National Computer Security Center. Ritchie was the head of Lucent Technologies System Software Research Department when he retired in 2007. Reeds went on developing encryption systems, and in 1998, he solved the ciphers in the third book of Trithemius's Steganographia [18]. The trio's paper about M-209 ciphertext-only cryptanalysis has never been published.

Our initial research goal was to try and reconstruct the Reeds-Ritchie-Morris method. However, the clues from Ritchie's account and other sources were not sufficient to achieve this goal. Another motivation was an attempt to solve M-209 cryptographic challenges. Those include the final exercises from Barker's book, Cryptanalysis of the Hagelin Cryptograph, published in 1977 [1]. To our best knowledge, no solution has been published for those exercises. Another series of challenges with increasing difficulty was published by Jean-Francois Bouchaudy in his M-209 Challenge website [3].

We eventually came up with two novel cryptanalytic attacks on the Hagelin M-209, both based on hillclimbing algorithms. The first one is an automated known-plaintext attack, which we already presented in a separate paper [12]. The second one is a ciphertext-only method, which can recover key settings from messages as short as 750 characters. With those methods, we solved almost all of Bouchaudy's M-209 challenges, including advanced challenges in the bonus section. We also solved Barker's book final exercises. In this paper we present our ciphertext-only method.

The rest of this paper is organized as follows: In Section 2 we present the description of the Hagelin M-209 device and its logical functioning, as well as some background information about the Hagelin C-Series and various versions of the M-209 operating instructions. Based on this description, we present in Section 3 an analysis of the keyspace of the M-209 as well as prior ciphertext-only attacks. After that, a description of our novel 4-stage hillclimbing algorithm for a ciphertext-only attack follows in Section 4. In Section 5, we present an evaluation of our attack, including performance measurements, an analysis of the work factor, and how we solved the challenges. Finally, we summarize our findings in Section 6, including conclusions about the security of the Hagelin M-209 device.

2 The Hagelin M-209 Encryption Device

The Hagelin M-209, also known as CSP-1500, is a portable and compact mechanical encryption device derived from the earlier C-38 which was developed by Boris Hagelin in 1938. The M-209 is built only of mechanical components, and does not require any power source. About 140 000 M-209 units were produced in total. We present here a functional description of the M-209 device, and how it differs from other Hagelin C-Series encryption devices. We also present the operating instructions which refer to key selection.

2.1 Functional Description

We reproduce here the functional description of the device, which we presented in our prior article about a known-plaintext attack [12]. We show the mechanical internals of the M-209 device in Figure 1, and a functional diagram in Figure 2. The M-209 functions as a stream cipher, with a pseudo-random displacement sequence generator, and a Beaufort encoder, i.e. a Caesar cipher with an inverted alphabet, as shown in Figure 1 (C). The pseudo-random displacement generator consists of two parts: A rotating cage with 27 bars (see Figure 1 (A) and Figure 2), and a set of six wheels (see Figure 1 (B) and Figure 2). The wheels are non-replaceable, unlike in later Hagelin models. Wheels #1, #2, #3, #4, #5, and #6 have 26, 25, 23, 21, 19 and 17 letters, respectively. Next to each letter, there is a pin which can be set to an effective or non-effective state. On each wheel, one of the pins is in the active position, against the bars of the cage. At each step of the encryption or decryption process, all the wheels rotate by exactly one step. Each bar in the cage has two movable lugs. Each lug may be set against any of the six wheels, or set to the neutral position (0), but both lugs may not be set against the same wheel. According to operating instructions (see Section 2.3), at least one of the two lugs should be set for each bar. When both lugs on a bar are set (to different wheels), the bar and the two wheels are involved in *lug overlap*, a feature which significantly increases the cryptographic security of the device.

The operator usually changes the settings of the wheel pins and the lugs on a daily basis, according to key lists distributed periodically. For each message, he selects the initial position of the 6 wheels (see the device with cover closed on the left side of Figure 2). The operator encrypts the message



Figure 1: The Hagelin M-209 Machine and its Mechanical Internals¹

letter by letter. He selects a plaintext letter using the disk on the left side of the device, and presses a power handle. The disk has 27 symbols, A to Z and a space symbol. Space symbols are internally replaced by the letter Z. When the power handle is pressed, all wheels rotate by one step, thus replacing the 6 pins in the active position. In addition, the cage performs a full revolution around its 27 bars. For each bar, if any one of the two lugs was set against a wheel for which the pin in the active position is in effective state (see Figure 2), the bar is engaged and it moves to the left. The displacement used for encoding the current letter is equal to the number of bars engaged, and may have a value from 0 to 27. This displacement is then applied to the current plaintext letter, using a Beaufort scheme (see Figure 2), to form the ciphertext letter, as follows:

$$CiphertextLetter[i] = (Z - PlaintextLetter[i] + Displacement[i]) \mod 26$$
(1)

The letter A is represented by the number 0, B by 1, ... Z by 25. The device prints the ciphertext letters on a paper tape, on the left side of the device. For convenience, the ciphertext is printed in spaced groups of 5 letters each. To decrypt a message, the operator selects the decryption mode, using a handle on the left of the device. The decryption process is essentially the same, except that Z symbols in the decrypted text are replaced by spaces, and the printed plaintext is not divided into 5-letter groups. Because the wheels have different number of pins, and those numbers are co-prime, the displacement sequence will not repeat itself until $26 \cdot 25 \cdot 23 \cdot 21 \cdot 19 \cdot 17 = 101, 405, 850 \approx 2^{27}$ steps.

2.2 The Hagelin C Series

The M-209 was the most successful of the Hagelin C-Series of encryption devices. We present here a survey of other C-Series devices and how they differ from the M-209. Earlier models, the Hagelin C-35 and C-36, were developed in 1935 and 1936, respectively. These two devices had only 5 wheels and 25 bars. Those bars had only one lug each. Furthermore, the position of the lugs was fixed. To increase cryptographic security, movable lugs were introduced in the C-38 [10], the immediate predecessor of the M-209. As with the M-209, the C-38 had 27 bars. Furthermore, each bar had two lugs, allowing for lug overlap. The main difference between the C-38 and the M-209 was the slide function, available in the C-38 and C-36. At the last stage of the encryption, a user selectable slide value is added (modulo 26) to the encrypted character. The encryption/decryption formula for the C-38 and C-36 is therefore:

 $CiphertextLetter[i] = (Slide - PlaintextLetter[i] + Displacement[i]) \mod 26$ (2)

¹Source: Wikipedia http://commons.wikimedia.org/wiki/File:M209B-IMG_0553-black.jpg and Wikimedia Commons http://en.wikipedia.org/wiki/File:M209B-IMG_0557.JPG. Created on 2/11/2011 by user *Rama* http:// commons.wikimedia.org/wiki/User:Rama



Figure 2: Functional Diagram of the Hagelin M-209

The slide feature requires the cryptanalyst to repeat his work for all 26 possible slides values. It was not considered, however, to significantly add to the security of the device. It was therefore not included by the US Army in the design of the M-209. With the M-209, the slide value is fixed and equal to Z (25).

There were several variants of the C-36 and C-38 designs [15]. The Swedish version of the C-38 had 29 bars instead of 27. The C-446 device was equivalent to the C-38, but also included a second printing apparatus, for the plaintext. The Italian Navy version, with the misleading name of C-38m, had 6 wheels and 27 bars, each bar with a single movable lug. One or two of the bars could be selected to implement a simple slide function with only two possible values, 1 or 2 [6]. The Hagelin BC-38 and BC-543 were functionally equivalent to the C-38 but also included a keyboard and an internal motor.

Post-war C-Series models, such as the C-52/CX-52, had also 6 wheels, selected from a set of 12 wheels. In addition, they had 32 bars, 5 of which could be used to generate an irregular movement of the wheels. In contrast, in all earlier models, the movement of the wheel was regular, one step for each plaintext/ciphertext character. Irregular wheel stepping significantly adds to the cryptographic security of the device. A lower-end device, the CD-57, was the most compact of all the C Series devices and could easily fit in a pocket. Its 6 wheels were selectable from a set of 12 wheels, but often the device was supplied with a fixed set of 6 wheels [15]. Instead of bars, it had 6 adjustable displacement disks, one per each wheel. Those disks were functionally equivalent to a cage of 40 bars, each bar with a single movable lug (and therefore no lug overlap).

The Hagelin C-Series devices were extensively used by armies and in embassy settings, from the late 1930s and until the 1950s, and in some countries probably until the 1970s.

2.3 Operating Instructions

Several versions of the US Army operating instructions for the M-209 device were in effect during the the 1940s and 1950s. Those contain detailed guidelines on how to create new keys. The motivation for those guidelines was to prevent the operators from selecting degenerate or cryptographically weak

Year	Details
1942	TM 11-380 Technical Manual Converter M-209, 33 pages, April 27, 1942 [26]
1943	TM 11-380B Technical Manual Converter M-209, 42 pages, September 20, 1943 [27]
1944	TM 11-380 Technical Manual Converter M-209, M-209A, M-209B, 78 pages, March 17, 1944 [28]
1947	TM 11-380 Technical Manual Converter M-209, M-209A, M-209B, 170 pages, May 1947 [29][1]
1951	Update dated April 10, 1951. We could not find the document.
1953	Mentioned in correspondence from Crypto-Aids Division to C/SEC, April 8, 1953 [4]

Table 1: Versions of the M-209 Operating Instructions

settings. For example, if there are no bars with a lug in front of wheel #6, this effectively turns the system into a weaker 5-wheel system. The guidelines were also designed to hide any statistical characteristics which may be useful for cryptanalysis. On the other hand, those guidelines also have the effect of restricting the size of the operational keyspace. In our ciphertext-only attack, we take advantage of those restrictions. We provide here a list of the known versions of the operating instructions, as well as their guidelines for selecting keys.

Known Versions of the Operating Instructions: There are at least five or six known versions of the TM 11-380 Technical Manual, produced by the US War Department [16]. The list is presented in Table 1. We were able to obtain copies of the 1942, 1943 and 1944 manuals, and the correspondence mentioning the 1953 revision. However, we could not obtain a copy of the 1951 version, but based on the April 1953 correspondence, the 1951 revision is unlikely to have introduced major chances in key selection procedures. The May 1947 version is included in its entirety in Barker's book [1].

Pin Setting Guidelines: All versions specified that at least 40% of the wheel pins should be in effective state, but no more than 60%. Also, on any wheel there should be no more than 6 consecutive pins set to the same state (effective or non-effective). This restriction was removed in the 1953 version.

Lug Count Guidelines: Starting with the 1942 version, all versions provided guidelines regarding the allowed *Lug Count Patterns*, which we describe below. To do so, we first define the concept of *Lug Count*, which we denote by *LC*. LC[w] is the number of lugs set in front of wheel w, with $1 \le w \le 6$. According to the 1942 version of the technical manual, the requirements for Lug Counts are as follows:

- 1. $0 < LC[w] \le 13$, for each wheel w.
- 2. $28 \leq \Sigma LC[w] \leq 39$. This means that there should be at least one bar with two lugs set (lug overlap), since the total number of bars is 27, and the number of overlaps is equal to $\Sigma LC[w] 27$.
- 3. There should be an equal mix of even and odd values for LC[w], i.e. three of the wheels should have even lug counts, and the other three should have odd lug counts. A consequence of this rule is that $\Sigma LC[w]$ is always odd, and the number of overlaps, $\Sigma LC[w] - 27$, is therefore always even.
- 4. For each number S from 1 to 27, there should be at least one combination of wheels for which $\Sigma LC[w] = S$. One of the consequences of this rule is that there should always be one wheel, w_1 , for which $LC[w_1] = 1$, i.e. there is exactly one lug set in front of w_1 .

We also introduce the concept of Lug Count Patterns. A Lug Count Pattern is also an array with 6 elements of lug counts. However, in a Lug Count Pattern, the elements are ordered according to their lug count values, starting from the lowest count, and not by the wheel number. While a pattern specifies lug counts for the 6 wheels, a Lug Count Pattern alone does not specify how those counts are assigned to the specific wheels. By running a simulation on all possible patterns, we found that there are only 58 such valid Lug Count Patterns which comply with the 1942 version of the guidelines. Those Lug Count Patterns are listed in Appendix A.1. It can be seen that for all those 1942 version patterns, there is always a wheel with 1 lug associated to it, denoted by w_1 , and another wheel w_2 with two lugs associated to it. Pattern $\{1, 2, 4, 5, 6, 11\}$, which is pattern #11 in the list in Appendix A.1, is an example of such a valid pattern. It complies with all the rules set in the 1942 version of the operating manual. According to this pattern, one wheel, w_1 , will have one lug associated to it, another wheel (w_2) will have two lugs, one wheel (w_3) will have four lugs, and so on. The total lug count for this pattern is 1 + 2 + 4 + 5 + 6 + 11 = 29, which means that there are 29 - 27 = 2 overlaps, i.e. two bars with two lugs set, all the remaining 27 - 2 = 25 bars having only one lug set.

There is a one-to-many relationship between Lug Count Patterns and Lug Count arrays. Each Lug Count array is associated with a Lug Count Pattern which we may obtain simply by sorting the elements of the Lug Count array. For example, the Lug Count arrays $LC_1 = \{6, 1, 5, 2, 11, 4\}$ and $LC_2 = \{11, 4, 6, 2, 5, 1\}$ are both associated to the Lug Count Pattern $\#11 = \{1, 2, 4, 5, 6, 11\}$. Inversely, if we apply the Lug Count Pattern $\#11 = \{1, 2, 4, 5, 6, 11\}$, and map wheel w_1 (the first wheel in the pattern) to physical wheel #2, w_2 to #4, w_3 to #6, w_4 to #3, w_5 to #1, and w_6 to #5, we obtain the Lug Count array $\{6, 1, 5, 2, 11, 4\}$.

In 1943, the restriction that there should be an equal mix of even and odd values for LC[w], both equal to 3, was relaxed, and patterns with 2, 3 or 4 even values of LC[w] were allowed. According to our simulations, this increases the number of allowed Lug Count Patterns from 58 in the 1942 version to 334 in the 1943 version. This also allows for an overlap count of only 1. Also, the wheels with the lowest lug counts, w_1 and w_2 , may both have a lug count of 1. With the 1943 version, w_2 , always had a lug count of 1. An example of such a pattern is $\{1, 1, 2, 3, 8, 13\}$. This pattern has an overlap count of 1, and both w_1 and w_2 have a lug count of 1.

In the 1944 version, the patterns were divided into two groups, Group A and Group B. The patterns in Group B were considered to be less secure and could be used for no more that 10% of the keys. The 1944 manual included a comprehensive list of all 144 Group A and 204 Group B allowed lug count patterns, 348 in total. In the 1947 version, patterns with a single lug overlap were not allowed anymore and were removed from the list. The new list contained only 134 Group A and 167 Group B patterns, 301 patterns in total. One of the Group B patterns in the list is in fact erroneous and cannot be used, leaving only 300 valid patterns. The 1953 version extended the list of patterns by allowing $LC[w] \leq 14$ instead of $LC[w] \leq 13$. According to our simulations, we estimate the number of valid patterns for the 1953 version to be approximately 469.

In addition, all the versions specified that for each one of the 27 bars, at least one of the two lugs should be set in front of one of the wheels.

Lug Overlap Distribution Guidelines: The 1943 and later versions of the manual also specified guidelines on how the overlaps should be distributed among the wheels, as follows:

- 1. Most of the wheels (4 or more) should be involved in lug overlaps.
- 2. Overlaps should involve wheels which are side-by-side (such as #3 and #4) as well as wheels which are separated (such as #2 and #5).
- 3. Many small overlaps, for several pairs of wheels, are preferable to many overlaps for a single pair of wheels. The 1944 and later versions strengthened this requirement by limiting the number of overlaps for a single pair of wheels to 4 at most.

The 1947 and 1953 versions specified that at most one wheel may be in a *Total Overlap* state, i.e. that all lugs in front of that wheel are on bars with overlaps (with the two lugs set). In addition, the 1953 version added overlap distribution guidelines designed to produce more random displacement values.

Other Guidelines: The 1947 version included a guideline that messages longer than 100 groups of 5 letters (500 characters) should be split into separate shorter messages. All the versions specified that the initial position of the 6 wheels should be different for each message.

In the detailed description of our ciphertext-only attack we show, for simplicity, how it is applied to the 1942 version, as the method is quite complex. However, our attack is not restricted to the 1942 version, as we show in the performance evaluation in Section 5.

3 Cryptanalysis

In this chapter we first analyze the keyspace of the Hagelin M-209 machine. We also present an overview of prior ciphertext-only attacks on the Hagelin M-209. We describe in more detail the ciphertext-only method developed by Geoff Sullivan [22], as we rely on some of its concepts in the design of our own method.

3.1 Keyspace

We reproduce here the keyspace analysis of the device, which we presented in our prior publication about a known-plaintext attack [12]. The settings of the device consists of the *Wheel Settings* which include the wheel pins and the initial position of the wheels, and of the *Lug Settings* - the settings of the lugs of the 27 bars. The overall keyspace consists of the combination of the keyspaces of the wheel settings and of the lug settings.

3.1.1 Wheel Settings Keyspace

Wheels #1, #2, #3, #4, #5, and #6 have 26, 25, 23, 21, 19 and 17 pins respectively, with a total of 131 pins. Each one of the pins may be set to either effective or ineffective. Therefore, the size of the keyspace for the wheel pin settings is 2^{131} .

In addition, the initial position of each one of the 6 wheels may be set by the operator. There are $26 \cdot 25 \cdot 23 \cdot 21 \cdot 19 \cdot 17 = 101,405,850 \approx 2^{27}$ distinct initial wheel position settings. Usually, the operator modifies the initial wheel positions for each new message, while the pin and lug settings are changed daily. He also encrypts the 6 letters representing the initial positions of the wheels, and sends them encrypted, as part of the message preamble. There are various methods to encrypt the initial wheel positions, such as using the daily pin and lug settings and default "AAAAAA" initial wheel positions. In some rare cases, the initial positions of the wheels are sent in clear, or somehow they are known to the cryptanalyst. In those cases, it is necessary to take into account the initial wheel positions, as after recovering the pin and lug settings for one message, other messages on the same day and network may easily be decrypted, by just replacing the initial wheel positions. In our algorithm, we can either use the initial wheel positions, if they are not known. This is possible since any set of pin settings in conjunction with the default "AAAAAA" initial wheel positions. To illustrate this, consider the following sample wheel pin settings, given the initial wheel positions "BBBBBBB":

```
Wheel 1: 01101110011000111100001101
Wheel 2: 001111000111010010100110
Wheel 3: 0010111011110111101111
Wheel 4: 010011101111011110111
Wheel 5: 01011101111011101
Wheel 6: 01101110111101111
```

Listing 1: Example of Pin Settings with Initial wheel positions "BBBBBB"

In this example, in wheel #1, pins #2,#3,#5,#6,#7,#10,#11,#15,#16,#17,#18,#23,#24 and #26 are in effective state, and all the other pins are in ineffective state. By rotating those wheel pin settings (for "BBBBBB") by one step to the right, using a cyclic rotation, we can obtain wheel pin settings for the case of default "AAAAAA" initial wheel positions, which are cryptographically equivalent. Those equivalent pin settings are shown below:

```
      Wheel
      1:
      1011011100110001110000110

      Wheel
      2:
      000111100011101010101

      Wheel
      3:
      1001011101111011110

      Wheel
      4:
      10100111011110111101

      Wheel
      5:
      001011101111011101

      Wheel
      6:
      1011011110111101
```

Listing 2: Equivalent Pin Settings with Initial wheel positions "AAAAAA"

For the attack presented here, as the initial wheel positions are either known, or assumed to be "AAAAAA", they do not affect the size of the wheel settings keyspace, which remains 2^{131} .

3.1.2 Lug Settings Keyspace

Each one of the 27 bars in the cage has two movable lugs. Each lug can be set to be in front of any one of the 6 wheels, but the two lugs of the same bar cannot be set to be in front of the same wheel. Also, in practical uses of the device, at least one of the lugs of each bar is always set. In the notation commonly used for lug settings, the lowest wheel number is specified first (e.g. 1-4, rather than 4-1), and if only one of the lugs is set, the number 0 is used instead of the first wheel (e.g. 0-1). An example of lug settings is shown in Table 2.

Bar	1	2	3	4	5	6	7	8	9
Lug Settings	1-4	3-4	0-1	0-2	0-2	0-2	0-3	0-3	0-3
Bar	10	11	12	13	14	15	16	17	18
Lug Settings	0-3	0-3	0-3	0-3	0-3	0-3	0-3	0-3	0-3
Bar	19	20	21	22	23	24	25	26	27
Lug Settings	0-4	0-5	0-6	0-6	0-6	0-6	0-6	0-6	0-6

Table 2: An Example of Lug Settings

There are 21 possible ways of settings the two lugs of a bar, as follows:

- 1. Only one of the two lugs is set to be against one of the 6 wheels, and the second is set to the neutral position (0). There are 6 possible such settings: 0-1, 0-2, 0-3, 0-4, 0-5 and 0-6.
- 2. Both lugs are set. This case is known as *lugs overlap*. There are $\frac{(6\cdot 5)}{2} = 15$ possible lug settings for a bar, with overlap. In the example shown in Table 2, bars #1 and #2 have lug settings with overlap, 1-4 and 3-4 respectively.

In theory, there should be 21^{27} possibilities to set up the bars lugs, or approximately 2^{118} . From the cryptographic perspective, however, many of those settings are equivalent. In the encryption process, each one of the 27 bars independently contributes to the total displacement value applied by the Beaufort encoder to the current input letter. In the example shown in Table 2, bar #1 has lugs set to wheels #1 and #4 (1-4), and bar #3 has only one lug set to wheel #1 (0-1). Those settings are cryptographically equivalent to the settings shown in Table 3, where bar #1 has only one lug set to wheel #1 (0-1), and bar #3 has lugs set to wheels #1 and #4 (1-4).

Bar	1	2	3	4	5	6	7	8	9
Lug Settings	0-1	3-4	1-4	0-2	0-2	0-2	0-3	0-3	0-3
Bar	10	11	12	13	14	15	16	17	18
Lug Settings	0-3	0-3	0-3	0-3	0-3	0-3	0-3	0-3	0-3
Bar	19	20	21	22	23	24	25	26	27
Lug Settings	0-4	0-5	0-6	0-6	0-6	0-6	0-6	0-6	0-6

Table 3: Lug Settings Equivalent to the Settings in Prior Example

What actually matters is the number of bars with each one of the 21 distinct types of lug settings. We can, therefore, represent any lug settings for the device, by keeping a count of the bars with each one of the 21 possible types of lug settings. This concise form also represents any other equivalent set of lug settings. This is illustrated in Table 4 which shows this non-redundant alternative representation of the same lug settings as in Table 2 or Table 3. We shall use this concise and non-redundant representation throughout this paper, for our analysis and in our algorithms.

Lug Settings Type	0-1	0-2	0-3	0-4	0-5	0-6	1-4	3-4
Number of Bars	1	3	12	1	1	7	1	1

Table 4: Non-redundant Representation of the Lug Settings from the Prior Example

After discarding the redundant settings, the cryptographically relevant size of the keyspace for the lug settings can now be calculated as follows: We need to distribute k = 27 indistinguishable elements (the bars) into n = 21 distinguishable buckets (the 21 distinct possible lug settings per each bar). Hence, according to the "bars and stars" formula, the number of cryptographically distinct lug settings is as follows:

$$C(n+k-1,k) = C(21+27-1,27) = \frac{47!}{20! \cdot 27!} \approx 2^{43}$$
(3)

3.1.3 Additional Constraints on the Lug Settings Keyspace

The keyspace for the lug settings is further reduced by operating procedure constraints. The effect of operating procedure constraints on the size of the lug settings keyspace is difficult to accurately quantify. We still may provide an upper bound, by taking into account some of the constraints.

We compute here an upper-bound for the size of the keyspace of the lug settings, when applying the constraints derived from the 1942 version of the operating instructions. From Appendix A.1, we can see that there may be only 2, 4, 6, 8, 10, or 12 overlaps. We compute the upper limit for the number of possible lug settings with exactly v lugs overlaps, for each possible value of v. Each one of the v bars (with lugs overlap) may each be set to one of 15 bar lugs setting types (e.g. 1-2, 1-3, etc...). According to the "bars and stars" formula, there are C(15 + v - 1, v) possible lug settings for those v bars. The remaining 27 - v bars may each be set to one of the 6 bar lug setting types without overlap (0-1, 0-2, 0-3, 0-4, 0-5 or 0-6), and therefore there are C(6 + (27 - v) - 1, 27 - v) possible lug settings for those 27 - v bars. Hence, the upper limit for the number of possible lug settings for all bars, with v overlaps, is as follows:

$$C(15 + v - 1, v) \cdot C(6 + (27 - v) - 1, 27 - v) \tag{4}$$

In Table 5 we show the number of possible settings for each one of the allowed overlap values. The total number - an upper limit for the number of possible lug settings, is about 2^{38} , compared to 2^{43} without operating procedures constraints.

The analysis for the case of the 1943 and later versions of the operating instructions is more complex. On the one hand, there are more lug count patterns, from 300 to 469 vs. only 58 for 1942. On the other hand, there are new restrictions on how the overlaps are to be distributed. Each revision from 1943 had increasingly restrictive guidelines for overlap distribution. We therefore estimate that the size of the effective keyspace for the lug settings is probably comparable or smaller than for the 1942 version, especially for the latest versions (1947 and later).

3.1.4 Combined Keyspace

The full, cryptographically relevant, keyspace of the Hagelin M-209 is therefore the combined keyspace for the wheel pin settings and for the lug settings, i.e. approximately $2^{131} \cdot 2^{38} = 2^{169}$.

3.2 Prior Ciphertext-Only Attacks on the M-209

In addition to the method developed by Reeds, Ritchie, and Morris, which was never published, several publications describe other ciphertext-only attacks on the M-209. According to the declassified

Number of Overlaps v	Possible Key Lug Settings	
2	17 100 720	2^{24}
4	300 736 800	2^{28}
6	2549632800	2^{31}
8	13591504080	2^{33}
10	51647715504	2^{35}
12	149732980800	2^{37}
Total	217839670704	2^{38}

Table 5: Number of Possible Key Lug Settings per Number of Lugs Overlaps

TICOM I-175 and DF-120 reports [24][23], the German cryptographic services in World War 2 were able to recover key settings from ciphertext in special cases, such as messages "in-depth". Those special cases included messages sent with the same key settings, messages retransmitted because of key setting errors, or messages with closely related initial wheel settings. They also developed statistical methods to determine whether messages were in-depth, fully or partially, as well as mechanical/electrical devices to facilitate the process. In addition, they investigated a more generic statistical attack. According to the TICOM Report I-45 [25], they were only able to solve a synthetic message in German with 5 000 letters.

In his 1977 book [1], Barker describes a ciphertext-only attack based on the statistical analysis of letter frequency distribution patterns, applied to each one of the pins of a certain wheel. For example, wheel number #6 has 17 pins. This wheel completes a full rotation cycle every 17 letters. According to Barker's method, the analyst gathers letter frequency statistics for each one of the 17 pins of wheel #6. For pin #1, he gathers statistics for ciphertext letters at positions 1, 18, 35, and so on. For pin #2, he gathers statistics for letters at positions 2, 19, 36, and so on. Same applies to the remaining pins of wheel #6. For a message of length 2500, the analyst may obtain letter frequency statistics for a sample of $N = \frac{2500}{17} = 147$ letters, for each pin of wheel #6. The other wheels (#1, #2, #3, #4 and #5) also rotate, but with different cycles, and letter statistics for their pins are gathered accordingly. Next, for each possible pair of pins p_a and p_b in a given wheel, the analyst performs a Chi test on F_a and F_b , which are the letter frequency distribution for pins p_a and p_b , respectively, as follows:

$$\Sigma_{l=A}^{Z} \frac{F_a[l]F_b[l]}{N_a N_b} \tag{5}$$

 N_a and N_b are the total numbers of samples, for pins p_a and p_b , respectively. This Chi test value indicates how close the letter frequency distributions for pins p_a and p_b are. For any given wheel, the letter frequency patterns for pins in effective state are expected to differ from letter frequency patterns for pins in ineffective state. This is expected since only pins in effective state affect the displacement sequence, while pins in ineffective state do not. The analyst uses the Chi test to divide the pins of the wheel into two distinct classes. One class contains pins likely to be in ineffective state, and the second pins likely to be in effective state. Barker describes techniques to identify the classes, and to handle ambiguous cases. The analyst repeats the process for the other wheels, factoring in the findings from previous wheels, until the pin settings for all wheels have been recovered. He finally recovers the lug settings. Barker demonstrates this technique on a theoretical 4-wheel device, and does not provide any quantitative analysis about the method's performance.

In their 1982 book [2], Beker and Piper present a similar method. They propose different techniques to divide the pins into classes, and to solve ambiguities. They demonstrate the method on a M-209 sample message with 3 000 letters. The plaintext of this sample message has an usually high number of space symbols, internally converted to Z symbols. Baker and Piper claim that their method generally works with 2 500 letters, and often with only 2 000, but they do not provide any detailed quantitative analysis. Interestingly enough, those numbers are similar to the numbers from Ritchie's account. On the other hand, Ritchie claimed that the Reeds-Ritchie-Morris method was different from Barker's method [19]. As the method developed by Beker and Piper is similar to Barker's, this comment probably applies to their method as well.

In [20], Rivest presents a theoretical analysis showing that 8 000 letters are required for cryptanal-

ysis using Barker's Chi test method. He nevertheless concludes that in practice 2000 to 4000 letters are usually enough.

Finally, a method for ciphertext-only recovery of M-209 settings is presented by Geoff Sullivan in [22]. Using a divide-and-conquer approach, the method incrementally recovers pin and lug settings, one or two wheels at a time. It requires a text of at least 2 500 to recover most of the key settings. Since our method relies on similar concepts for Stage 1 and Stage 2, we describe Sullivan's method in more detail in the following section, as well as its limitations.

3.3 Overview of Sullivan's Ciphertext-Only Method

The method developed by Sullivan is a divide-and-conquer approach. It tries to incrementally recover pin settings of certain wheels, while isolating the effects of the other wheels [22]. Sullivan's method relies on lug setting restrictions derived from the 1942 version of the operating instructions, as described in Section 2.3. As a result of those restrictions, there is always one wheel, denoted by w_1 , in front of which there is exactly one lug. Similarly, there is always one wheel, denoted by w_2 , in front of which there are exactly two lugs. The goal of the first stage in the Sullivan method is to identify the most likely w_1 and w_2 wheels, out of the $6 \cdot 5 = 30$ options.

We present here the core concept of Sullivan's first stage algorithm. As depicted in Figure 2, at each step of the decryption (or encryption) process, the lugs on the bars are affected only by the 6 pins in the active position, one per wheel. Those 6 pins constitute a boolean vector of size 6, therefore having $2^6 = 64$ possible vector values. If we know the correct pin settings for wheels w_1 and w_2 , the correct state of the active pins of wheels w_1 and w_2 is also known, at each position of the ciphertext message. There are $2^4 = 16$ possible values for the remaining four wheels for which pin settings are unknown. Assuming those 16 values are equally distributed, there is a probability of 1/16 that all the active pins of the remaining four wheels are in their ineffective case. For all such positions in the message, only wheels w_1 and w_2 affect decryption (or encryption).

If we set the pin settings of w_1 and w_2 to their correct settings, the pin settings of the other wheels as ineffective, assign one bar with a lug in front of w_1 and two bars with a lug assigned to w_2 , and decrypt the ciphertext using this putative key, we therefore may obtain a decrypted text with about 1/16 of the letters correctly decrypted. The Index of Coincidence for the resulting decrypted text is expected to be higher than the Index of Coincidence of a text decrypted using random wrong keys.

The algorithm of the first stage of Sullivan's method takes advantage of this characteristic. For each possible $\{w_1, w_2\}$ selection, it performs a hillclimbing search for the optimal pin settings of wheels w_1 and w_2 . The pins of the other four wheels are kept in ineffective state. One bar has one lug set to w_1 , and two bars with one lug each set to w_2 . The lugs on the other 27 - 3 = 24 bars are in neutral position. At each step of hillclimbing, the algorithm inverts the state of one of the pins of either w_1 or w_2 , from effective to ineffective, or vice versa. It then decrypts the message, and if the Index of Coincidence score improves, it keeps the new settings. Otherwise, it rolls back and discards the change. It completes the first stage by applying this hillclimbing algorithm to all the possible $\{w_1, w_2\}$ pairs, and selecting the $\{w_1, w_2\}$ pair with the highest Index of Coincidence after hillclimbing. It also keeps the putative pin settings of w_1 and w_2 for the second stage.

The goal of the second stage of Sullivan's method is to find the pin settings for the other four wheels, one wheel at a time. The first cycle of the second stage consists of finding the optimal pin settings for a third wheel, as well as the most likely number of lugs in front of that third wheel. To do so, Sullivan's second stage algorithm first sets the pin settings of wheels w_1 and w_2 to those found in the first stage. It also sets up one bar with one lug set to w_1 , and two bars with one lug each set to w_2 . It then tests each one of the remaining four wheels, as follows: for each such candidate wheel w, it tests different assumptions about the number of lugs which are in front of it, starting from 3 lugs and up to 13 lugs, while assuming there are no bars with lugs overlaps. It applies a hillclimbing search with the Index of Coincidence as the fitness score, for the optimal settings of the pins of candidate wheel w. During hillclimbing, only the states of the pins of candidate wheel w are changed. For each candidate wheel w, it keeps the Index of Coincidence value and the pin settings obtained by hillclimbing, as well as the optimal number of lugs. When all candidate wheels have been tested, it selects the candidate wheel for which hillclimbing achieved the highest Index of Coincidence. Sullivan's second stage algorithm repeats the whole process to find the best fourth wheel from the remaining 3 wheels, using the settings obtained in the first stage and in prior cycles of the second stage. It terminates when all the wheels have been processed. The results of the second stage include putative pin settings for all the wheels, as well as putative lug settings, although those lug settings are inaccurate since they do not take into account lugs overlaps. Sullivan suggests an additional stage of hillclimbing for improving the recovered lug settings.

While Sullivan's method may often find most of the correct pin and lug settings, it has several limitations. First, the algorithm is likely to fail if lugs set to wheels w_1 or w_2 are in bars with overlaps. In general, lugs overlaps in any of the bars may disrupt the whole process. In addition, the second stage depends on correctly selecting w_1 and w_2 in the first stage. Similarly, errors in one of the cycles of the second stage while recovering pin settings or lug settings are likely to propagate and disrupt the next cycles. Sullivan demonstrates his method on a 2500 letters message. The Index of Coincidence he obtains after decrypting the ciphertext with the recovered pin and lug settings is relatively low, 0.045, v.s. an expected plaintext Index of Coincidence of 0.0738. Therefore, the decrypted text is probably hard if not impossible to read. In the next section, we describe our approach. In particular, we describe how we addressed the limitations of Sullivan's method by improving and extending the scope of the two stages, and adding two additional stages.

4 Our Four-Stage Hillclimbing Ciphertext-Only Attack

The method we developed consists of a 4-stage hillclimbing algorithm. For simplicity, we describe our method in the context of the key setting guidelines specified in the 1942 version of the operating instructions (see Section 2.3). Our method, however, is not restricted to the 1942 version. It can be applied to later versions as well, as described in Section 4.6.

The main challenge for any ciphertext-only attack is the need to recover both the lug settings and the pin settings while none of the two are known. As shown by Sullivan in [22], it is possible to recover the pin settings using hillclimbing, if the lug settings are known. Similarly, we were able to implement a simple hillclimbing method to recover the lug settings once the correct pin settings are known. In our attack described here, we try to incrementally recover elements of the correct lug settings and pin settings, using a 4-stage algorithm, described in high-level in Figure 3.



Figure 3: High-Level Flow Diagram – 4-Stage Algorithm

The first two stages improve Sullivan's divide-and-conquer 2-stage approach, by significantly extending their scope. Stage 1 is similar to the first stage of Sullivan's first stage. But instead of trying to find the best w_1 and w_2 , we keep the results for all $6 \cdot 5 = 30$ combinations of w_1 and w_2 , as input to Stage 2. In Stage 2, we extend the process of recovering pin settings for the remaining 4 wheels to a set of 41760 representative *Sample Lug Settings*. The outcome of Stage 2 is used as starting points for Stage 3, in which we perform a comprehensive hillclimbing search for improved pin settings and lug settings. We perform hillclimbing on all 41760 cases from Stage 2, and use the Index of Coincidence as the fitness score. At the end of Stage 3, we expect to have found most of the correct pin and lug settings. In Stage 4, we perform a more in-depth hillclimbing process, using bigram and monogram (N-Gram) statistics for scoring. Stage 4 is performed only on selected cases from Stage 3 which are most likely to converge, and is intended to fully recover the pin and lug settings. For longer messages, Stage 3 is sometimes enough. The whole process of running Stages 1, 2, 3 and 4, is usually repeated up to 100 times, or until a solution is found. Each stage of the algorithm is described in detail in the following sections.

4.1 Stage 1

Stage 1 is similar to the first stage of Sullivan's method. As a result of restrictions imposed by the rules in the 1942 version of the operating procedures (see Section 3.1.3), we also assume that there is always one wheel, w_1 , for which there is one bar with one lug set to it. We assume there is another wheel w_2 , for which there are two bars, each one with one lug set to w_2 . But instead of looking for the best $\{w_1, w_2\}$ combination as in Sullivan's method, we process in Stage 1 all $6 \cdot 5 = 30$ combinations of $\{w_1, w_2\}$. We keep the results for all those 30 combinations as input for Stage 2. When processing a pair of $\{w_1, w_2\}$ wheels, we assume that all the pins on the other four wheels are ineffective. We also assume that the lugs on the remaining 27 - 3 = 24 bars are set to neutral(0). This means that only wheels w_1 and w_2 affect the decryption process. We then run a hillclimbing algorithm which searches for the optimal pin settings for wheels w_1 and w_2 . As a fitness score, we use the Index of Coincidence of the putative plaintext obtained by decrypting the ciphertext with the putative key settings. The rationale for Stage 1 is presented in Section 3.3, where we describe Sullivan's method. More details are available in the pseudo-code in Appendix A.3.

4.2 Stage 2

As with Sullivan's second stage, Stage 2 of our algorithm processes the remaining 4 wheels one wheel at a time, to find optimal pin settings. The main difference is about how lug settings are deduced. The inaccurate recovery of the lug settings was one of the main limitations of Sullivan's method, especially in the case of lug overlaps. In practice, lug settings always involve lug overlaps. The core element of our solution to overcome those limitations is the novel concept of the *Sample Lug Settings*, which we introduce in Section 4.2.1. After that we present the Stage 2 algorithm in Section 4.2.2.

4.2.1 Sample Lug Settings

The full keyspace of the key lug settings is about 2^{38} , therefore it is not feasible to run a hillclimbing search for pin settings, for each possible case of lug settings in the keyspace. Furthermore, this is not necessary. It is possible to recover most of the correct pin settings, if we only know an "approximate" set of lug settings, which includes most but not necessarily all of the correct lug settings.

Our approach is to build a set of sample lug settings, from which at least one is likely to be close enough to our unknown lug settings. To do so, we start with the 58 valid Lug Count Patterns (see Appendix A.1), which comply with the restrictions derived from the 1942 version of the operating instructions. We then apply each pattern to each possible mapping of the wheels in the pattern, w_1 to w_6 , to the physical wheels, #1 to #6. Since there are 6! = 720 ways to map the 6 wheels, we now have a set of $720 \cdot 58 = 41760$ Sample Lug Settings, or $\approx 2^{15}$. For each Sample Lug Settings, we now have a precise lug count for each physical wheel. The last missing part is how to distribute the lug overlaps between the specific wheels. For that purpose we define a set of arbitrary but simple rules, which are described in detail in Appendix A.2.

A different way to look at the same concept, is to consider each case of Sample Lug Settings as a representative of the group of possible lug settings, which share the same lug counts and the same mapping of lug counts to physical wheels, but differ in the way the lug overlaps are distributed among the wheels. The correct (and unknown) lug settings belong to one of those 41 760 groups. The hope is that the Sample Lug Settings case, which represents that group, will not be too distant from the correct lug settings. The Sample Lug Settings therefore constitute a representative subset of all possible lug settings. This set is distributed over the whole lug settings keyspace, and it covers lug overlaps as well.

We expect at least one of the 41760 sample cases to be not too far away from the correct lug settings, either the sample with the same lug counts and mapping of wheels, or even another sample. Using approximate or partially correct lug settings enables hillclimbing to recover most of the correct pin settings, so the processing of at least one of those 41760 Sample Lug Settings cases should yield partially correct pin settings. As the number of the Sample Lug Settings cases is only $\approx 2^{15}$ compared to the full lug settings keyspace size of 2^{38} , it is computationally feasible to process all of them and run a hillclimbing search for pin settings for each case. Note that this approach may be fine-tuned, using new rules, different from the rules used in this work and described in Appendix A.2. Another approach could be to add more representative lug settings samples for each group, rather than just a single one. This obviously requires more computing power, or more time to run the full algorithm.

An additional benefit of using Sample Lug Settings, is that lug settings assumptions are not dependent on the success of earlier steps in either Stage 1 or 2. Each Sample Lug Settings case specifies a full set of lug settings for all the lugs and their associated wheels, and for each case, the lug settings are kept unchanged throughout the cycles of Stage 2. This is in contrast to Sullivan's second stage algorithm, where lug settings are recovered incrementally, and errors usually propagate to later cycles.

As described in Section 4.6, both the number of valid Lug Count Patterns, and the number of Sample Lug Settings, differ according to the version of the operating instructions.

4.2.2 Stage 2 Algorithm

For each case of Sample Lug Settings, Stage 2 starts with the putative pin settings for w_1 and w_2 which we obtained in Stage 1. We set the bars and the lugs in accordance with the Sample Lug Settings case, and we process each one of the remaining four wheels, one by one. We start with the wheels which have a lower number of lugs associated to it (according to the Sample Lug Settings). At each cycle, we obtain putative pin settings for the wheel, using a hillclimbing search and the Index of Coincidence as the fitness score. We also keep the wheel putative pin settings for the next cycles. When we have processed all the wheels, we keep the putative pin settings and the lug settings for the Sample Lug Settings case. The outcome of Stage 2 is a set of putative key settings, for each one of the 41 760 cases. More details about Stage 2 are available in the pseudo-code in Appendix A.4.

4.3 Stage 3

In Stage 3, we seek to improve the approximate putative pin and lug settings (the Sample Lug Settings) produced in Stage 2. We do this by running a combined hillclimbing search for optimal pin settings and for optimal lug settings. We use the Index of Coincidence on the putative decrypted text as the fitness score. We repeat this search for each one of the 41 760 cases produced by Stage 2. In contrast to Stage 2, in this stage the search for improved lug settings is no longer limited to the Lug Settings Samples. Rather, those Lug Settings Samples from Stage 2 serve as starting points for the search for improved putative lug settings. Furthermore, there is no incremental processing of one wheel at a time, as in Stage 2. The search is over the full key space, repeatedly testing whether a change in the setting of any pin, in any wheel, or in the settings of any bar, may improve the fitness score. The outcome of Stage 3 is a set of improved putative key settings for each one of the 41 760 cases, as well as the Index of Coincidence value for each case. Often, if the message is long enough, the improved putative settings for one of the cases may already contain the correct key settings. More details about Stage 3 are available in the pseudo-code in Appendix A.5.

4.4 Stage 4

Stage 4 is similar to Stage 3, with two major differences. First, we perform Stage 4 only on selected cases from Stage 3, rather than on all the 41760 cases. We select only those cases for which the

Message Length	Index of Coincidence Threshold
2000 or more	0.050
1500 - 2000	0.051
1250 - 1500	0.052
1000 - 1250	0.053
750 - 1000	0.055
less than 750	0.057

Table 6: Index of Coincidence Noise Threshold per Message Length

Index of Coincidence value obtained in Stage 3 is higher than a "noise threshold". If the Index of Coincidence of a decrypted ciphertext is above this threshold, there is a high probability that most of the putative key settings elements are correct. The Index of Coincidence threshold depends on the length of the message, and was found empirically. Threshold levels for various messages lengths are shown in Table 6. If none of the cases from Stage 3 had a sufficient Index of Coincidence, we still try and process the top 3 best cases from Stage 3 (those with the highest Index of Coincidence values), although the probability for finding the correct key settings is lower.

In addition, we use a different scoring method for hillclimbing in Stage 4. In Stages 1, 2 and 3, we used the Index of Coincidence as the fitness score. For hillclimbing in Stage 4, we use N-Gram statistics. N-Gram statistics are more accurate than the Index of Coincidence, but they are effective only if most of the key elements are already correct. We therefore use N-Gram statistics only for the last stage of our algorithm, for those selected cases in which most of the key elements are already correct. Furthermore, the longer the N-Grams, e.g. trigrams vs. bigrams, the more accurate the score. On the other hand, higher level N-Grams are also less effective in the presence of key setting errors. We therefore used a combination of monogram statistics and bigram statistics, as a compromise to balance between accuracy and effectiveness in the presence of setting errors. To compute the fitness score for Stage 4, we first decrypt the message with the putative pin and lug settings. We then compute the sum of the counts of all bigrams in the decrypted text, multiplied by their respective log-frequencies in the English language. To this score, we add the sum of the counts for all monograms, multiplied by their respective log-frequencies in the English language. If hillclimbing of Stage 4 results in a N-Gram score above a certain value, there is a very high probability that all key settings are correct. In this case, we stop, and there is no need to process the remaining cases.

4.5 Summary of the Four Stages

The 4-stage algorithm is relatively complex, yet each one of the stages plays a critical role. We made some experiments, such as skipping Stage 3, or running only Stages 3 and 4, and in almost all cases, solutions could not be found. The main purpose of Stage 1 and Stage 2 is to provide good starting points for Stage 3, with approximate putative pin and lug settings. In Stage 2, we use a set of 41760 Sample Lug Settings, and we search for the optimal pin settings under each one of those lug settings cases. The Sample Lug Settings are designed to provide a reasonable coverage and well distributed starting points over the lug settings space. In Stage 3, we hillclimb to try and improve putative pin settings as well as lug settings. The search for improved lug settings in Stage 3 is no more limited to the Sample Lug Settings as in Stage 2. In Stage 3 we use the Index of Coincidence as the fitness score, even though the N-Gram statistics are more accurate. We do so because at this stage we may still have a high number of errors. We conclude the process in Stage 4 with a similar hillclimbing using N-Gram statistics, which are more likely to allow for final convergence. Stage 4 runs only on those cases which are most likely to converge. For longer messages (e.g. 2000 letters or more) and few lug overlaps, a single pass of the 4 stages is often enough to find the solution. More passes are usually required for shorter messages and/or more lug overlaps. For the performance analysis presented in the next section, we allowed the algorithm to run up to 100 times, or until the key settings were recovered.

4.6 Other Versions of the Operating Instructions

For simplicity, in the previous sections, the description of our ciphertext-only attack assumed that the 1942 version of the operating instructions was in effect. The method, however, can also be applied to other versions, as described here:

- 1. With the 1942 version, there are 58 valid Lug Count Patterns. With other versions, there are different sets of patterns. As mentioned in Section 2.3, there are 334, 348, 300 and 469 patterns for the 1943, 1944, 1947 and 1953 versions, respectively.
- 2. Accordingly, while we run our algorithms on $720 \cdot 58 = 41\,760$ Sample Lug Settings with the 1942 version, we run the algorithms on 240 480, 250 560, 216 000 and 337 680 Sample Lug Settings for the 1943, 1944, 1947 and 1953 versions, respectively. This has the effect of requiring more processing time to cover this increased number of Sample Lug Settings.
- 3. In addition, in the Lug Count Patterns for the 1942 version, the wheels with the lowest lug counts, w_1 and w_2 , always had a lug count of 1 and 2, respectively. With the 1943 and above version, while w_1 will always have a lug count of 1, w_2 may have a lug count of either 1 or 2, and not just 2. The algorithm of Stage 1 handles both cases.

5 Performance Evaluation

We present here a performance evaluation of our methods, in terms of success rate and work factor, under several scenarios - message length, number of lug overlaps, and the version of the operating instructions. In addition, we show how we solved publicly available M-209 ciphertext-only challenges using this method.

5.1 Success Rate

We tested the algorithm on simulated messages in English. The length of the messages varied from 750 to 2500, and the number of lug overlaps was 2, 6 or 12. The keys were randomly generated and were designed to comply with the requirements of the operating instructions. The operating instructions for 1943, 1944 and 1947 are very similar, with incremental changes designed to increase cryptographic security. On the other hand, those later versions differ significantly from the instructions in the 1942 version. We therefore tested the performance of our ciphertext-only attack in the context of the 1942 version and of the 1947 version. Those two versions cover the period from 1942 to 1951. Some details are missing about the 1951 and 1953 rules, and we did not include them in our measurements. We estimate that the performance of the algorithms in the context of those later versions would not differ too much from the performance with the 1947 version.

For each case of a given length and number of lug overlaps, we processed a batch of 100 simulated messages. For each message we ran the four stages up to 100 times or less if the key was recovered earlier. The success rate of the algorithm is displayed in Figure 4. For comparison, we show the performance of the Reeds-Ritchie-Morris algorithm. According to Ritchie, their method could recover key settings for most of the messages with 2 500 letters, and for half of the messages with 2 000 letters. In Figure 4, we interpreted 'most' as 90%. Our multistage hillclimbing method performs significantly better. It can also be seen that using messages shorter than 750 letters, and at least 6 lug overlaps, significantly increases the security of the M-209. Surprisingly, the improvements introduced up to the 1947 version had almost not effect on the success rate of the algorithm for 2 and 6 overlaps, while they reduced the success rate for 12 overlaps by about 10%, e.g. 49% instead of 59% for messages of with 1 250 letters.

5.2 Work Factor Analysis

We also analyzed the work factor for the algorithm. When running the algorithm, the vast majority of the time is spent while decrypting the ciphertext after each change on the key, and computing the Index of Coincidence. In Stages 1 and 2, almost all key changes are changes on the pin settings. Stages 3 and 4, there are also changes on lug settings, but those account for less than 10% of the key settings changes. Furthermore, almost all of the changes in pin settings consist of changing the state



Figure 4: Success Rate for Full Key Recovery vs. Message Length and Number of Overlaps

of a single pin. To shorten the time processing changes in pin settings, we introduced an optimization in the decryption process. Instead of decrypting again the entire ciphertext after each change in the state of a pin, we only decrypt those characters of the message which are in positions affected by the change. For example, if we modify the state of a single pin in wheel #1, which has 26 pins, we only need to update the decrypted text at every 26th position. This optimization reduced the processing time by more than 70%.

We measured the total numbers of decryptions required to fully recover the key settings. In Figure 5, we show the average number of decryptions, only for those cases where the full key settings were successfully recovered. The numbers are shown on a log_2 scale. For comparison, the size of the full keyspace is 2^{169} . On a 3.4 GHz Intel Core i7 quad-core PC and using multiple threads, the solution for a long message (2 500 letters) with 2 lug overlaps usually requires a few minutes. For a short message (1 000 letters) encrypted with key settings with 12 lug overlaps, several hours are usually required.

The time required for a solution for a key complying to the 1947 version of the operating instructions is about 4-5 times longer than the time required with the 1942 version. This is expected, as there are 300 valid patterns for the 1947 version vs. 58 for the 1942 version, about 5 times more.



Figure 5: Average Number of Decryptions Required for Full Key Recovery

5.3 Solving Challenges

Using our method, we also solved several ciphertext-only challenges (without messages 'in-depth') from the M-209 Challenge site [3] of Jean-Francois Bouchaudy, including a message with 1 000 characters (Bonus #11). For another message with only 620 letters, key settings could not be recovered.

In the example given by Beker and Piper [2], the message has 3000 letters and the lug settings have 6 lug overlaps. Our method recovered the correct key settings in a few minutes. It was also able to recover the correct key settings after truncating the message and leaving only the first 1000 letters.

We also found solutions for the final exercises, #60 to #63, in Barker's book [1]. They consist of 4 messages, of lengths 1035, 1145, 520 and 545 respectively. They were encrypted using the same lug settings and pin settings, but with different (and unrelated) wheel starting positions. The cryptograms were created by Greg Mellen (1926-1998), Editor Emeritus of Cryptologia, and published in 1977. Barker proposes an extension to his Chi test method for the solution of the exercises, however, we were not successful in applying the suggested method to the 4 messages. To date, we are not aware of any published solution to those exercises, which conclude a series of exercises with increasing difficulty.

We applied our ciphertext-only method to the two longer cryptograms, setting our program to apply the rules from the 1944 version of the operating instructions. We let our program run for two days, without success. After changing the settings of the program to instead apply the rules from the 1942 version, a solution was produced in two hours, for the first message (with 1035 letters). The lug settings include 6 overlaps. Wheels w_1 (physical wheel #6) and w_2 (physical wheel #2) are both involved in lug overlaps. This may have affected the algorithm performance. The full lug settings are as follows: 0-1 0-1 0-2 0-3 0-3 0-3 0-3 0-3 0-4 0-4 0-4 0-4 0-4 0-5 0-5 0-5 0-5 0-5 1-2 3-4 3-4 4-5 4-5 5-6. Using those known lug settings, a simple hillclimbing search for the pin settings produced solutions for the other 3 messages.

All four plaintexts are excerpts from classical cryptography publications. The first message starts with *DECIPHERING IS BOTH A SCIENCE AND AN ART*, from William F. Friedman's An Introduction to Methods for the Solutions of Ciphers [7]. The second message starts with *WE CAN DISTINGUISH RATHER CLEARLY THREE KINDS OF COMMUNICATION WHICH ARE TO BE ENCIPHERED*, from the Manual of Cryptography by Luigi Sacco (English version) [21]. The third message starts with *AN EXAMINATION OF EITHER PLAIN TEXT OR CRYPTOGRAPHIC TEXT*, from Solomon Kullback's Statistical Methods in Cryptanalysis [11]. The last message starts with *IT IS A CURIOUS TWIST IN HUMAN PSYCHOLOGY*, from Friedman's Elements of Cryptanalysis [8].

6 Conclusion

In this paper we presented a complex but effective 4-stage hillclimbing method for the ciphertext-only cryptanalysis of Hagelin M-209 messages. Our method uses a mix of keyspace reduction techniques and targeted search heuristics, such as the novel concept of representative Sample Lug Settings. Our performance evaluation shows that our method is significantly more effective than any other published statistical method.

In addition, we may draw several conclusions in regards to the cryptographic security of the device. M-209 messages longer than 750 letters are susceptible to our ciphertext-only attack. On the other hand, the lug overlap feature of the device, especially with 12 overlaps, significantly increases its operational security.

Operating instructions, originally intended to increase operational cryptographic security, also have the effect of restricting the size of the keyspace, and therefore allowing for this kind of targeted search attack. Revisions introduced in 1943 and later versions to improve the cryptographic security of the device do not significantly affect the performance of the attack described here. A guideline to limit message length to 500 and split longer messages was eventually introduced but only in the 1947 version.

The method presented here is applicable, possibly with some minor changes, to all the C-Series models developed in the 1930s and 1940s. Further research is needed to evaluate the applicability of our attack to Hagelin encryption devices developed in the 1950s, such as the C-52/CX-52 or the CD-57. Another direction for further research could be to fine-tune the set of representative Sample

Lug Settings, or to improve the performance of the whole algorithm by simply extending the set to cover more cases, using more computing power.

Acknowledgements

We would like to thank Jean-Francois Bouchaudy for publishing the M-209 challenges and for his knowledgeable contributions about the M-209 and other Hagelin devices. We would like to thank our reviewers for their invaluable insights and for kindly providing access to some key historical documents.

About the Authors

George Lasry is a computer scientist in the high-tech industry in Israel. Prior to this, he worked for many years in the development of communications systems, and also managed R&D and sales organizations. His primary interest in cryptographic research is the application of specialized optimization techniques for the computerized cryptanalysis of classical ciphers and cipher machines. Using such a technique, he solved in November 2013 [13] the Double Transposition (Doppelwürfel) cipher challenge which was published by Klaus Schmeh in 2007.

Nils Kopal is a Ph.D. student with the research group "Applied Information Security" (AIS) at the University of Kassel. He studied applied computer science at the University of Duisburg-Essen in Duisburg with focus on distributed and secure systems. He finished his studies with a master's degree. With the AIS group he is the responsible coordinator for the development of the open source elearning tool CrypTool 2.0 [5] which illustrates classical and modern cryptographic and cryptanalytic concepts.

Professor Arno Wacker is the head of the research group "Applied Information Security" (AIS) with the University of Kassel in Germany. He is also the lead of the open source project CrypTool 2.0 [5] and member of the steering group of MysteryTwister C3. His main research interests are modern security protocols for decentralized distributed systems, e.g. for volunteer computing scenarios, and computerized cryptanalysis of classical ciphers. Additionally, he teaches classical and modern cryptology in classes at the University of Kassel and at special workshops for pupils at local schools.

A Appendices

A.1 Lug Count Patterns

We show in Table 7 the full set of Lug Count Patterns, 58 in total, which comply with the rules set in the 1942 version of the Technical Manual [26]. For example, pattern #51 specifies that there should be one wheel, denoted as w_1 , with one bar with one lug set in front of that wheel, as well as another wheel (w_2) with a count of 2 lugs. In addition, it specifies that there should be one wheel (w_3) with 3 lugs associated to it, one wheel with 6 lugs (w_4) , one wheel with 12 lugs (w_5) , and the last one with 13 lugs (w_6) . This pattern has 10 lug overlaps.

Note that a pattern does not specify which physical wheel is w_1 and has a lug count of 1, which wheel is w_2 , and so on. Furthermore, while it does specify the total number of overlaps, it does not specify how those overlaps are distributed between the wheels.

An important observation from Table 7 is that lug counts for w_1 and w_2 are always 1 and 2, respectively, for any one of the 58 patterns. Note that this is true only for the 1942 version of the operating instructions. The patterns for 1943 and later versions also allow for cases where w_2 has a lug count of 1 as well. In addition, in the 1942 version, the number of overlaps is always even (2, 4, 6, 8, 10, or 12) while in later versions the number of overlaps may also be odd. Furthermore, in the 1943 and 1944 versions, there are patterns with only one overlap. Such patterns were considered as less secure, and were removed in 1947. Finally, there are about 5-6 times more patterns for the later versions, varying from 300 (1947) to about 469 (1953), compared with 58 in the 1942 version.

Pattern No	w1	$\mathbf{w2}$	w3	w4	w5	w6	Overlaps
1	1	2	3	4	9	10	2
2	1	2	3	4	8	11	2
3	1	2	3	4	7	12	2
4	1	2	3	4	6	13	2
5	1	2	3	5	8	10	2
6	1	2	3	5	6	12	2
7	1	2	3	6	8	9	2
8	1	2	3	6	7	10	2
9	1	2	4	5	8	9	2
10	1	2	4	5	7	10	2
11	1	2	4	5	6	11	2
12	1	2	4	6	7	9	2
13	1	2	3	4	10	11	4
14	1	2	3	4	9	12	4
15	1	2	3	4	8	13	4
16	1	2	3	5	8	12	4
17	1	2	3	6	9	10	4
18	1	2	3	6	8	11	4
19	1	2	3	6	7	12	4
20	1	2	3	7	8	10	4
21	1	2	4	5	9	10	4
22	1	2	4	5	8	11	4
23	1	2	4	5	7	12	4
24	1	2	4	5	6	13	4
25	1	2	4	6	7	11	4
26	1	2	4	7	8	9	4
27	1	2	3	4	11	12	6
28	1	2	3	4	10	13	6
29	1	2	3	5	10	12	6
30	1	2	3	6	10	11	6
31	1	2	3	6	9	12	6
32	1	2	3	6	8	13	6
33	1	2	3	7	8	12	6
34	1	2	4	5	10	11	6
35	1	2	4	5	9	12	6
36	1	2	4	5	8	13	6
37	1	2	4	6 C	9	11	6
38	1	2	4	0	(13	0 C
39	1	2	4	(7	9	10	0
40	1	2	4	((8	11	0
41	1	2	う 2	0 6	11	12	ð 0
42	1	2	ა ი	0	10	13	8
43	1	2	3	(F	10	12	8
44	1	2	4	0 F	11	12	8
40	1	2	4	C C	10	15 19	ð 0
40	1	2	4	0	9	15	ð
41	1	2	4	1	10	11	ð
48	1	2	4	1	9	12	ð
49 50	1	2	4	0	0	13	0
50	1	2	4	8 6	9	11	ð 10
50	1	2	3	0 F	12	10	10
52	1	2	4	0 6	12	15	10
55		2	4	0	11	13	10

54	1	2	4	7	11	12	10
55	1	2	4	7	10	13	10
56	1	2	4	8	9	13	10
57	1	2	4	7	12	13	12
58	1	2	4	8	11	13	12

 Table 7: Lug Count Patterns for 1942 Operating Instructions

A.2 Sample Lug Settings Rules

Sample Lug Settings are generated by applying each one of the valid Lug Count Patterns (see Appendix A.1) to a all 6! = 720 possible mappings of the six wheels in the pattern, w_1 to w_6 , to the physical wheels #1 to #6. For example, for the 58 Lug Count Patterns derived from the 1942 version of the operating instructions, there are $58 \cdot 720 = 41760$ Sample Lug Settings. There 334, 348, 300 and 469 patterns for the 1943, 1944, 1947 and 1953 versions, respectively. Accordingly, we generate 240 480, 250 560, 216 000 and 337 680 Sample Lug Settings when using the 1943, 1944, 1947 and 1953 versions, respectively. This has the effect of requiring more processing time to cover this increased number of Sample Lug Settings.

In addition, we apply arbitrary (but simple) rules for the allocation of lug overlaps. First, no overlap is allowed for the lugs set to wheels w_1 and w_2 . In addition, lugs in front of wheel w_3 may share bars (lug overlap) only with lugs set in front of wheel w_4 . Same applies to wheels w_5 and w_6 , respectively. Furthermore, the number of overlaps is divided between the pairs $\{w3, w4\}$ and $\{w5, w6\}$ in proportion to the number of lugs set to w_3 and w_5 in the Lug Count Pattern, respectively. Note that those rules for distributing the overlaps among the various wheels when generating the Sample Lug Settings, were primarily designed for simplicity. They may be incompatible with the guidelines specified in certain versions of the operating instructions (1943 and later). Therefore, other rules, better tailored to match the guidelines from the later versions of the operating instructions, may yield better results.

To illustrate the concept, we show in Table 8 how to derive sample lug settings from Lug Count Pattern $\#51 = \{1, 2, 3, 6, 12, 13\}$. We apply the pattern to the following order of the wheels: $w_1 = \#5$, $w_2 = \#6$, $w_3 = \#1$, $w_4 = \#3$, $w_5 = \#2$ and $w_6 = \#4$, where #1 to #6 are physical wheels numbers. There is one bar with a lug set in front of wheel $w_1 = \#5$, without sharing ("0-5" lug settings). There are two bars with a lug set in front of $w_2 = \#6$, without sharing ("0-6" settings). The sum of the elements in Lug Count Pattern #51 is 37, and as there are 27 bars in the M-209, there are 10 lug overlaps in this pattern. Since wheel w_3 (#1) has a count of 3 lugs in the pattern, and w_5 (#2) has a count of 12 lugs, we allocate the 10 overlaps at a ratio of 3:12=2:8 between the pairs $\{w3, w4\}$ and $\{w5, w6\}$. Since $w_3 = \#1$ and $w_4 = \#3$, there are two bars with the lug settings "1-3". Similarly, there are 8 bars with the settings "2-4". All the remaining bars are without overlap. For example, wheel w_5 (mapped to #2) has a count of 12 lugs in the Lug Count Pattern. Since 8 of them are on shared bars (with overlap), the remaining 4 are on non-shared bars, so we have 4 bars with the lug settings "0-2". The full Sample Lug Settings are displayed in Table 9 (only entries with Number of Bars > 0 are displayed).

A.3 Pseudo Code for Stage 1

```
// Stage 1 Algorithm.
//
Input: ciphertext
// The ciphertext.
//
Output: stage1_w1_pins_settings
// Array of size 30, each entry contains
// Array of size 30, each entry contains
// per each combination of w1, w2 found by Stage 1 hillclimbing,
// per each combination of {w1,w2}.
stage1_w2_pins_settings
// Same, for pin settings of w2.
```

	w1	w2	w3	w4	w5	w6	Comment
Physical Wheel	Wheel	Wheel	Wheel	Wheel	Wheel	Wheel	Physical wheel num-
#	#5	#6	#1	#3	#2	#4	bers shown in this line
Lug Counts Pat							Total Count = 37 ,
Lug Counts 1 at-	1	2	3	6	12	13	Overlaps = 37 - 27 =
(#31)							10
Luce with Bar							The 10 overlaps are al-
Sharing	0	0	$10 \cdot \frac{3}{3+}$	$\frac{3}{12} = 2$	$10 \cdot \frac{1}{3+}$	$\frac{2}{12} = 8$	located at a ratio of 3 :
Sharing							12 = 2:8
Luce without							The remaining lugs are
Bar Sharing	1	2	3-2=1	6-2=4	12-8=4	13-8=5	on bars without over-
Dai Sharing							lap

Table 8: Applying Lug Count Pattern #51 to Create Sample Lug Settings

Type	Number of Bars
0-1	1
0-2	4
0-3	4
0-4	5
0-5	1
0-6	2
1-3	2
2-4	8

Table 9: An Example of Sample Lug Settings

```
Also with 30 entries, one per each combination of \{w1, w2\}
for each ordered pair \{w1, w2\} from wheels \{1, 2, 3, 4, 5, 6\} {
   lug\_settings = neutral
                                      // Lugs on all bars in neutral position.
   allocate_bars(lug_settings,w1,1) // Allocate 1 bar with 1 lug set to w1.
   allocate_bars (lug_settings, w2,2) // Allocate 2 bars with 1 lug set to w2.
   best_ic = 0
   // Repeat hillclimbing 5 times and keep the best results for pair \{w1, w2\}.
   for (cycles = 0, cycles < 5, cycles++) {
      // Start the cycle with random settings for pins of w1 and w2.
      // The pins in the other 4 wheels are kept in ineffective state.
      // No change in the lug settings.
      pin_settings[w1] = random_pins_settings(w1)
pin_settings[w2] = random_pins_settings(w2)
      for each wheel w not in \{w1, w2\} {
         pin_settings [w] = ineffective
      }
      decrypted_text = decrypt(ciphertext, pin_settings, lug_settings)
      ic = index_of_coincidence(decrypted_text)
      // Repeatedly look for changes in w1 and w2 pin settings which
      // improve the Index of Coincidence. If no more improvement
      // can be achieved, start a new cycle.
      do {
         improved = false
```

```
// We process only pins of wheels w1 and w2.
         for each wheel w in \{w1, w2\} {
            for each pin p in wheel w {
               // Invert the state of the pin. Decrypt and check whether
               // the Index of Coincidence has improved.
               toggle_pin(pin_settings,w,p)
               decrypted_text = decrypt(ciphertext, pin_settings, lug_settings)
               new_ic = index_of_coincidence(decrypted_text)
               if (new_ic > ic) \{
                  // Improvement was achieved with the change in pin settings.
                  // Keep the change.
                  improved = true
                  ic = new_ic
                  if (ic > best_ic) {
                     // If this is also the best Index of Coincidence so far
                     // for pair \{w1, w2\}, then keep pin settings of w1 and w2
                     // in the relevant entry in the output arrays.
                     best_ic = ic
                     w1w2\_index = index\_of\_pair(w1,w2)
                     stage1_w1_pins_settings[w1w2_index] = pin_settings[w1]
                     stage1_w2_pins_settings[w1w2_index] = pin_settings[w2]
                  }
               else 
                  // No improvement - roll back the change.
                  toggle_pin(pin_settings,w,p)
               }
            \} // end of: for each pin ...
         } // end of: for each wheel ...
     } while (improved)
  \} // end of: for cycles ...
} // end for: each ordered pair ...
// Return optimal pin settings for wheels w1 and w2,
// per each possible \{w1, w2\} combination (total of 30 entries).
return (stage1_w1_pins_settings, stage1_w2_pins_settings)
 / End of Stage 1 Algorithm.
```

A.4 Pseudo Code for Stage 2

```
//
// Stage 2 Algorithm.
//
Input: ciphertext
// The ciphertext.
// The ciphertext.
stage1_w1_pins_settings, stage1_w2_pins_settings
// Pin settings for w1 and w2, obtained in Stage 1.
// Both are arrays of size 30, one entry per each
// combination of {w1,w2}.
//
Output: stage2_lugs_settings
// Array (size 41760) with the sample lug settings.
stage2_pins_settings
// Array with the best pin settings for each
// one of 41760 sample lug settings. Each entry contains the pins
// settings for all 6 wheels.
//
//
//
```

```
// Precompute the 58 Lug Count Patterns.
// See Section 3.1.3 and Appendix 1.
lugs_counts_patterns = precompute_lugs_counts_patterns()
// Iterate on all 30 possible pairs of \{w1, w2\}.
for each pair {w1,w2} from wheels {1,2,3,4,5,6} {
   // We use the pin settings for w1 and w2 as obtained from Stage 1.
   w1w2\_index = index\_of\_pair(w1,w2)
   pin_settings [w1] = stage1_w1_pins_settings [w1w2_index]
   pin_settings [w2] = stage1_w2_pins_settings [w1w2_index]
   // Iterate on each one of the 58 patterns.
   for each pattern in lugs_counts_patterns {
      // Iterate on each one of the 4! = 24 orderings
      // of the remaining 4 wheels.
      for each order {w3, w4, w5, w6} of wheels not in {w1, w2} {
         // We apply the Lug Count Pattern to the specific ordering
         // of the 6 wheels \{w1, w2, w3, w4, w5, w6\}, to obtain the
         // corresponding Sample Lug Settings. See Appendix 2.
         lug_settings = get_sample_lugs_settings(pattern, {w1, w2, w3, w4, w5, w6})
         // Also keep the sample lug settings case into the output array.
         index = sample_lugs_settings_index(lug_settings)
         stage2_lugs_settings[index,w] = lug_settings[w]
         // The pins for the wheels not yet processed are ineffective.
         for each wheel w not in {w1,w2}{
            pin_settings[w] = ineffective
         }
         // We process the 4 remaining wheels one by one.
         for each wheel w in wheels {w3, w4, w5, w6} {
            best_ic = 0
            // Run hillclimbing 3 times, keep the best results.
            for (cycles = 0, cycles < 3, cycles++) {
               // Start cycle by using random pin settings for w.
               pin_settings [w] = random_pins_settings (w)
               decrypted_text = decrypt(ciphertext, pin_settings, lug_settings)
               ic = index_of_coincidence(decrypted_text)
               // Continue as long as the Index of Coincidence can be
               // improved by changing wheel w pin settings.
               do {
                  improved = false
                  For each pin p of wheel w {
                     // Invert the state of the pin, and check whether
                     // the change improves the Index of Coincidence.
                     toggle_pin(pin_settings,w,p)
                     decrypted_text = decrypt(ciphertext, pin_settings,
                                               lug_settings)
                     new_ic = index_of_coincidence(decrypted_text)
                     if (new_ic > ic) {
                        // The change in settings of pin improved the
                         // Index of Coincidence. Keep the change.
                        improved = true
```

```
ic = new_ic
                          if (ic > best_ic) {
                             // This is also higher than the best Index of
                             // Coincidence so far for w (for the specific
                             // case of Sample Lug Settings).
                             //% \left( {{\rm Keep}} \right) the pin settings of wheel w in the
                             // relevant entry in the output array.
                             best_ic = ic
                             index = sample_lugs_settings_index(lug_settings)
                             stage2_pins_settings[index,w] = pin_settings[w]
                          }
                      } else {
                          // No improvement - roll back the change.
                          toggle_pin(pin_settings,w,p)
                       }
                   } // end of: for each pin \ldots
                } while (improved)
             } // end of: for cycles ...
            // We use the best pin settings for wheel w, when processing
             // the next wheels.
            index = sample_lugs_settings_index (lug_settings)
             pin_settings [w] = stage2_pins_settings [index,w]
         } // end of: for each wheel \ldots
  } // end of: for each order ...
} // end of: for each pattern ...
} // end of: for each ordered pair ...
//\ {\rm Return} best pin settings, and sample lug settings
// for each one of the 41760 Sample Lug Settings cases.
return (stage2_pins_settings, stage2_lugs_settings)
// End of Stage 2 Algorithm.
```

A.5 Pseudo Code for Stage 3

//	
// Stage 3	Algorithm.
//	
Input:	ciphertext
//	The ciphertext.
	stage2_pins_settings
//	One entry, containing the pin settings
//	for all wheels as found in Stage 2, per each Sample Lugs
//	Settings case. There are 41760 cases.
	stage2_lugs_settings
//	One entry, per each Sample Lug Settings
//	case, as processed in Stage 2 (41760 cases).
//	
Output:	stage3_pins_settings
//	One entry, containing improved pins
//	settings for all wheels as found in Stage 3, for each one
//	of the 41760 cases from Stage 2.
	stage3_lugs_settings
//	One entry, containing improved lugs
//	settings as found in Stage 3, for each one of the
//	41760 cases from Stage 2.
	stage3_ic

```
The best Index of Coincidence as found by Stage 3,
                 for each one of the 41760 cases from Stage 2.
// We process each case from the 41760 received from Stage 2.
// For each such case, we try to improve the pin settings.
// We also try to improve the lug settings.
for index from 1 to 41760 {
   stage3_ic[index] = 0
   // Run hillclimbing 10 times and keep the best results.
   for (cycles = 0, cycles < 10, cycles++) {
      // Start with the pin settings obtained from Stage 2.
      pin_settings = stage2_pins_settings[index]
      // Start with the lug settings obtained from Stage 2.
      // Note: Those are in fact the Sample Lug Settings.
      lug_settings = stage2_lugs_settings[index]
      decrypted_text = decrypt(ciphertext, pin_settings, lug_settings)
      ic = index_of_coincidence(decrypted_text)
      do {
         improved = false
         // Test transformations on lug settings. We do so by changing
         // the counts of the various bar Lug Setting Types
         // (see Section 3.1.2).
         // We have 6 types of lug settings for a bar % \left( {\left[ {{{\rm{A}}} \right]_{\rm{A}}} \right)
         // without overlap (0-1, 0-2, \ldots, 0-6),
// and 15 types with overlap (1-2, 1-3, \ldots, 5-6).
         for each curr_type in all_bar_lugs_settings_types {
             // Check that we have bars with such lug settings.
            if (count_types(lug_settings, curr_type) == 0) continue
            for each new_type in all_bar_lugs_settings_types {
                if (curr_type == new_type) continue
                // We change the lug settings of a single bar.
                // Decrease by one the number of bars with curr_type.
                // Increase by one the number of bars with new_type.
                decrease_type(lug_settings, curr_type)
                increase_type(lug_settings, new_type)
                decrypted_text = decrypt(ciphertext, pin_settings, lug_settings)
                new_ic = index_of_coincidence(decrypted_text)
                // Test whether the Index of Coincidence improved
                // after the change.
                if (new_ic > ic) {
                   // Improved, keep the change.
                   improved = true
                   ic = new_ic
                   if (ic > stage3_ic[index]) {
                      // This is also higher that the best Index of Coincidence
                      // so far for the case. Keep the lug settings and the
                      //\ {\rm pin} settings , and the Index of Coincidence value ,
                      // in the relevant entries in the output arrays.
                      stage3_ic[index] = ic
```

```
stage3_pins_settings [index] = pin_settings
                     stage3_lugs_settings [index] = lug_settings
                  }
               } else {
                  // Change did not improve - roll back.
                  // Increase by one the number of bars with curr_type.
                  // Decrease by one the number of bars with new_type.
                  increase_type(lug_settings, curr_type)
                  decrease_type (lug_settings, new_type)
            } // end of: for each curr_type
         } // end of: for each new_type
         // Test transformations on pin settings, in any of the 6 wheels.
         for each wheel w in \{1, 2, 3, 4, 5, 6\} {
            for each pin p of wheel w {
               // Invert the state of the pin.
               toggle_pin(pin_settings,w,p)
               decrypted_text = decrypt(ciphertext, pin_settings, lug_settings)
               new_ic = index_of_coincidence(decrypted_text)
               // Test whether the Index of Coincidence improved
               // after the change.
               if (new_ic > ic)
                  // Improved, keep the change.
                  improved = true
                  ic = new_ic
                  if (ic > stage3_ic[index]) {
                     // This is also higher that the best Index of Coincidence
                     // so far for the case. Keep the lug settings and the
                     // pin settings, and the Index of Coincidence,
                     // in the relevant entries in the output arrays.
                     stage3_ic[index] = ic
                     stage3_pins_settings [index] = pin_settings
                     stage3_lugs_settings [index] = lug_settings
                  }
               } else {
                  // Change did not improve - roll back the change.
                  toggle_pin(pin_settings,w,p)
               }
            } // end of: for each pin ...
         } // end of: for each wheel ...
      } while (improved)
     // end of: for cycles ...
  }
} // end of: for index
// Return results (pins, lugs and Index of Coincidence) for each one
// of the 41760 cases.
return (stage3_pins_settings, stage3_lugs_settings, stage3_ic)
// End of Stage 3 Algorithm.
```

References

[1] Wayne G. Barker. Cryptanalysis of the Hagelin Cryptograph, volume 17. Aegean Park Press, Laguna Hills, CA, 1977.

- [2] Henry Beker and Fred Piper. Cipher Systems: The Protection of Communications. Northwood Books London, 1982.
- [3] Jean-Francois Bouchaudy. The M-209 Challenge, 2014. http://www.jfbouch.fr/crypto/ challenge, [Accessed: September, 28th, 2014].
- [4] Crypto-Aids Division. Preparation of OLYMPUS and MARS Keys, 1953. Memorandom to C/SEC, April 8, 1953, NARA N36-10(11), Declassified Jan 17, 2012, E 013526.
- [5] CrypTool 2.0 Team. CrypTool Portal Cryptography for Everybody, June 2014. http://www.cryptool.org/, [Accessed: September, 28th, 2014].
- [6] Augusto Donini, Luigi andi Buonafalce. The Cryptographic Services of the Royal (British) and Italian Navies. Cryptologia, 14(2):97–127, 1990.
- [7] William F. Friedman. An Introduction to Methods for the Solution of Ciphers. Riverbank Laboratories, Department of Ciphers, Geneva, IL., 1918.
- [8] William F. Friedman. Elements of Cryptanalysis (Cryptographic Series). Aegean Park Press, Laguna Hills, CA, 1976.
- [9] Katie Hafner and John Markoff. Cyberpunk, Outlaws and Hackers on The Computer Frontier. Simon & Schuster, New York, NY, 1991.
- [10] Boris Hagelin and David Kahn. The Story of the Hagelin Cryptos. Cryptologia, 18(3):204–242, 1994.
- [11] Solomon Kullback. Statistical Methods in Cryptoanalysis (Cryptographic Series). Aegean Park Press, Laguna Hills, CA, 1976.
- [12] George Lasry, Nils Kopal, and Arno Wacker. Automated Known-Plaintext Cryptanalysis of Short Hagelin M-209 Messages. *Cryptologia*, 2014. Submitted to Cryptologia, and accepted for publication.
- [13] George Lasry, Nils Kopal, and Arno Wacker. Solving the Double Transposition Challenge with a Divide-and-Conquer Approach. *Cryptologia*, 38(3):197–214, 2014.
- [14] Robert Morris. The Hagelin Cipher Machine (M-209) Reconstruction of the Internal Settings. Cryptologia, 2(3):267–289, 1978.
- [15] Crypto Museum. Crypto Museum website, Crypto AG Hagelin cipher machines, 2014. http: //www.cryptomuseum.com/crypto/hagelin/index.htm, [Accessed: December, 7th, 2014].
- [16] Jerry Proc. Crypto Machines website, M209(CSP-1500), 2014. http://http://jproc.ca/ crypto/m209.html, [Accessed: December, 7th, 2014].
- [17] James Reeds, Dennis Ritchie, and Robert Morris. The Hagelin Cipher Machine (M-209): Cryptanalysis from Ciphertext Alone, 1978. Unpublished technical memorandum, Bell Laboratories, submitted to Cryptologia.
- [18] Jim Reeds. Solved: The Ciphers in Book III of Trithemius Steganographia. Cryptologia, 22(4):291–317, 1998.
- [19] Dennis Ritchie. Dabbling in the Cryptographic World A Story, 2000. http://cm.bell-labs. com/who/dmr/crypt.html, [Accessed: September, 28th, 2014].
- [20] Ronald L Rivest. Statistical Analysis of the Hagelin Cryptograph. Cryptologia, 5(1):27–32, 1981.
- [21] Luigi Sacco. Manual of Cryptography (Cryptographic Series). Aegean Park Press, Laguna Hills, CA, 1996.
- [22] Geoff Sullivan. Cryptanalysis of Hagelin Machine Pin Wheels. Cryptologia, 26(4):257–273, 2002.

- [23] TICOM. DF-120, Report on the Solution of Messages in Depth M-209 Traffic, 2014. http:// www.ticomarchive.com/the-targets/gdna-army/related-documents, [Accessed: December, 7th, 2014].
- [24] TICOM. I-175, Report by Alfred Pokorn of OKH/CHI on M-209, 2014. http://www. ticomarchive.com/the-targets/okw-chi/related-reports, [Accessed: September, 28th, 2014].
- [25] TICOM. I-45, OKW/Chi Cryptanalytic Research on Enigma, Hagelin and Cipher Teleprinter Machines, 2014. http://www.ticomarchive.com/the-targets/okw-chi/related-reports, [Accessed: September, 28th, 2014].
- [26] War Department. TM-11-380, Technical Manual, Converter M-209, 1942. http://maritime. org/tech/csp1500inst.htm, [Accessed: September, 28th, 2014].
- [27] War Department. TM-11-380 B, Technical Manual, Converter M-209B, 1943.
- [28] War Department. TM-11-380, Technical Manual, Converter M-209, M-209A, M-209 B (Cipher) , 1944. http://http://www.ilord.com/m209manual.html, [Accessed: December, 7th, 2014].
- [29] War Department. TM-11-380, Technical Manual, ConverterM-209, M-209A, M-209 B (Cipher) , 1947.