George Lasry

A Methodology for the Cryptanalysis of Classical Ciphers with Search Metaheuristics

kassel university press

George Lasry

A Methodology for the Cryptanalysis of Classical Ciphers with Search Metaheuristics

kassel university press

This work has been accepted by the Faculty of Electrical Engineering / Computer Science of the University of Kassel as a thesis for acquiring the academic degree of Doktor der Naturwissenschaften (Dr. rer. nat.).

Supervisor: Prof. Dr. Arno Wacker
Co-Supervisor: Prof. Bernhard Esslinger

Defense day: 16th October 2017

Bibliographic information published by Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at http://dnb.dnb.de.

Zugl.: Kassel, Univ., Diss. 2017
ISBN 978-3-7376-0458-1 (print)
ISBN 978-3-7376-0459-8 (e-book)
DOI: http://dx.medra.org/10.19211/KUP9783737604598
URN: http://nbn-resolving.de/urn:nbn:de:0002-404592

www.upress.uni-kassel.de

Printed in Germany

"After climbing a great hill, one only finds that there are many more hills to climb."

Nelson Mandela

Abstract

Cryptography, the art and science of creating secret codes, and cryptanalysis, the art and science of breaking secret codes, underwent a similar and parallel course during history. Both fields evolved from manual encryption methods and manual codebreaking techniques, to cipher machines and codebreaking machines in the first half of the 20th century, and finally to computer-based encryption and cryptanalysis from the second half of the 20th century. However, despite the advent of modern computing technology, some of the more challenging classical cipher systems and machines have not yet been successfully cryptanalyzed. For others, cryptanalytic methods exist, but only for special and advantageous cases, such as when large amounts of ciphertext are available.

Starting from the 1990s, local search metaheuristics such as hill climbing, genetic algorithms, and simulated annealing have been employed, and in some cases, successfully, for the cryptanalysis of several classical ciphers. In most cases, however, results were mixed, and the application of such methods rather limited in their scope and performance.

In this work, a robust framework and methodology for the cryptanalysis of classical ciphers using local search metaheuristics, mainly hill climbing and simulated annealing, is described. In an extensive set of case studies conducted as part of this research, this new methodology has been validated and demonstrated as highly effective for the cryptanalysis of several challenging cipher systems and machines, which could not be effectively cryptanalyzed before, and with drastic improvements compared to previously published methods. This work also led to the decipherment of original encrypted messages from WWI, and to the solution, for the first time, of several public cryptographic challenges.

Acknowledgements

I would like to thank my supervisor, Prof. Arno Wacker, for his ongoing support, helping me to transform a series of experiments into a full-fledged academic research, for his infinite patience, and his invaluable advice throughout the whole process. I would also like to thank Prof. Bernhard Esslinger for his advice and thorough review, not only catching embarrassing typos and mistakes, but also helping me better formulate some of the key ideas, and filling many gaps.

I would also like to thank my (other) co-authors and partners: Nils Kopal, for his ideas, support, good spirit and constant willingness to help, Dr. Ingo Niebel who contributed a well-researched, fascinating historical context, and an in-depth analysis of the original WWI messages deciphered as part of this research, as well as Moshe Rubin, for introducing me to the Chaocipher problem, and for providing a wealth of critical background information.

I am also grateful to the managers of the MysteryTwister C3 site (Prof. Esslinger, Wacker and May) for building and maintaining a set of high-quality, stimulating and inspiring crypto challenges. Those include the Double Transposition Challenge, for which I wish to thank Klaus Schmeh, as the work on his tough challenge and its solution generated new ideas that could be applied to additional problems. I also thank Dirk Rijmenants for his highly informative website, his crypto challenges which were the first I tried to solve, and for his kind assistance. I also thank Jean-François Bouchaudy, the owner of the Hagelin M-209 Challenge site, for creating a series of challenging and also entertaining problems, which I used as a testbed for developing new methods. I also thank Jean-François for his ongoing support, pushing me not to give up, and to try and break new limits in the cryptanalysis of the Hagelin M-209, until I was able to complete all of his challenges.

I also thank Frode Weierud, Jim Gillogly, Jim Reeds, Geoff Sullivan, Craig Bauer, Paul Gannon, Rene Stein and others to whom I apologize for having omitted, for their assistance in my research in numerous ways, such as providing copies of key documents, and bridging important gaps. I also thank Paul Reuvers and Marc Simons for their wonderful Crypto Museum site, and for their permission to reproduce their photos and diagrams of Enigma.

And last but not least, I would like to thank my loving wife Tali, for her ongoing moral and emotional support, and for allowing me to seclude myself for long hours in front of the computer, as well as my wonderful kids, Noam and Shay, who fill every day of my life with happiness and pride for being their father.

In memory of my parents, Marie and David Lasry
and dedicated to my family, my deepest source of inspiration
to my wife Tali, my son Noam and my daughter Shay
and to the Lasry, Feldman, Prizant, and Bocian families

Contents

List of Figures

List of Tables

Abbreviations

ADE	**A**ggregate **D**istance **E**rror (Hagelin M-209)
CCT	**C**omplete **C**olumnar **T**ransposition
DES	**D**ata **E**ncryption **S**tandard
FDA	**F**itness **D**istance **A**nalysis
FDC	**F**itness **D**istance **C**orrelation
GRASP	**G**reedy **R**andomized **A**daptive **S**earch **P**rocedures
HC	**H**ill **C**limbing
IC	**I**ndex of **C**oincidence
ICT	**I**ncomplete **C**olumnar **T**ransposition
IDP	**I**ndex of **D**igraphic **P**otential (double transposition)
ILS	**I**terated **L**ocal **S**earch
NSA	**N**ational **S**ecurity **A**gency
SA	**S**imulated **A**nnealing
SAT	(propositional) **SAT**isfiabilty
SLS	**S**tochastic **L**ocal **S**earch
TSP	**T**raveling **S**alesman **P**roblem
WWI	**W**orld **W**ar One
WWII	**W**orld **W**ar Two

1

Introduction

In this chapter, we present a short overview of the historical development of ciphers, as well as of codebreaking or cryptanalysis techniques. Despite the advent of modern computing, several of those classical ciphers remain unsolved, or there is no efficient cryptanalytic method for the cipher. We introduce the more recent application of optimization techniques, and more specifically, local search metaheuristics, for the cryptanalysis of classical ciphers. We also describe their limitations and the challenges in designing efficient techniques. We follow with an overview of this thesis contributions. The contributions include a novel, efficient methodology, validated with case studies, which allows for the cryptanalysis of several types of challenging ciphers, the decryption of historical messages which previously could not be read, and the solution for several leading historical cipher challenges. We conclude with the structure of this thesis.

1.1 Classical Cryptography

The use of secret codes or ciphers for the secure transmission of sensitive messages, either diplomatic, military or even personal, has been recorded as early as in Ancient Greece [1][2]. Techniques for codebreaking, or cryptanalysis, evolved in parallel to the development of cryptography. Earlier ciphers included simple substitution ciphers, e.g. the Caesar cipher and the monoalphabetic substitution cipher. The most effective attacks on classical ciphers, except getting hold of the keys, were primarily statistical in nature. The method of using frequency analysis for the solution of monoalphabetic substitution ciphers was introduced by A-Kindi, an Arab polymath, in the 9th century. Following those developments, the creators of new classical ciphers understood that to increase the cryptographic security of a cipher, an effort should be made to hide discernible statistical patterns in ciphertexts. To overcome the limitations of single (monoalphabetic) substitution, new ciphers used several substitution alphabets (polyalphabetic substitution), such as the Vigenère cipher. This cipher stood for 300 years as the "Le Chiffre indéchiffrable", or the indecipherable cipher. This cipher was solved only in the 19th century, where more sophisticated statistical methods were developed by Babbage and Kasisky. Those included statistical analysis of pairs of letters (digrams) or triplets of letters (trigrams), and the examination of repetitions. More sophisticated versions of the substitution ciphers, such as homophonic ciphers or Playfair, as well as new transposition ciphers, were introduced by armies and for diplomatic communications. Transposition ciphers, such as the Columnar Transposition

cipher, were introduced to hide digram or trigram statistics, by moving around the letters of the plaintext. Another mean to hide the language statistics was the use of a composite cipher, with several stages, such as the Double Transposition cipher, or combined substitution-transposition ciphers, such as the ADFGVX cipher. Both ciphers were introduced in World War I (WWI). Other means to deny the cryptanalyst from discerning useful statistical patterns included limiting the length of each message, and limiting the number of messages using the same key, to avoid "depths". The cryptanalysis of the new ciphers, proved even more challenging, and required more advanced statistical methods. In most cases, those ciphers could not be cryptanalyzed except for special cases or when a large amount of traffic was available. The period between the world wars also saw the transition of cryptanalysis from a science requiring mainly linguistic skills, to a science requiring mathematical abilities. Newer statistical methods were developed, including William Friedman's famous Index of Coincidence [3]. To overcome the limitations of manual ciphers, cipher machines made their appearance. Encryption machines developed in the 1920s, 1930s and 1940s were primarily of mechanical or electromechanical design, and used stepping or revolving rotors as their core encryption mechanism [4]. There were two main types and rotor mechanism for encryption machines. The first type included the German Enigma and the British Typex machines, and their rotors implemented a series of substitution steps. The rotor stepping mechanism was designed so that the substitution alphabet changes per each character, and the substitution sequence does not repeat itself for at least several tens of thousands to millions of cycles. This creates a polyalphabetic substitution system with a huge number of alphabets. Another but similar variant of those machines included the Japanese Red and Purple machines, which had stepping switches instead of stepping rotors. The second type of rotor machines included the Hagelin devices and the Lorenz SZ40/42 teleprinter encryption devices. Those devices also had stepping rotors, but those rotors had pins which controlled the generation of a pseudorandom key sequence, added to the plaintext to generate the ciphertext. Those devices tried to emulate the operation of a cryptographically secure one-time-pad stream cipher.

The introduction of encryption machines in the 1920s and 1930s also drove the use of other machines for their cryptanalysis, effectively creating a "war of machines against machines". While many ciphers were still solved by hand until World War II (WWII) and even afterwards, machines such as IBM tabulating machines and devices such as the Polish Bombe started to play a key role in cryptanalysis, culminating with the development of Turing Bombe and of the first fully electronic large-scale programmable computing system, the Colossus.

The 1960s saw the introduction of fully electronic encryption devices, and in the 1970s, of computer-based or integrated-circuit based encryption, the most notable event being the introduction of the Data Encryption Standard (DES). Together with the advent of public key cryptography, those developments marked the end of the classical cryptography era.

1.2 The Development of Cryptanalysis for Classical Ciphers

Extensive literature from the beginning of the 19th and the 20th centuries, as well as recently declassified NSA material, provide a wealth of information about manual cryptanalytic methods. Details about the historical methods may be found in [5], [6], [7] and [8]. The details of codebreaking machines developed before and during WWII, including the Turing Bombe and Colossus, have been published [9] [10], and fully functional replicas of those machines have been produced and are on display in museums. It is clear that cryptographic agencies such as the NSA made extensive use of general purpose computing starting from the 1950s, however from 1945 a secrecy curtain fell upon available sources. Scarce sources are available on those

subjects covering the period after 1945. Some recently NSA declassified material present the side-by-side evolution of computing in the industry and academic worlds from WWII until the 1970s, together with the use of increasingly substantial computer technology by the NSA [11]. The NSA often drove the specifications for the newest computer technology and systems. Unfortunately, almost no information is available on their use of computers to solve specific cipher systems, and even less about the details of computerized cryptanalysis of ciphers.

In the history of cryptography, there are several examples where one of the main cryptanalytic efforts was focused on identifying the details of the encryption system or machine. This is quite easy if a machine or a cipher manual falls into the hands of the party trying to cryptanalyze a cipher, but there are quite a few examples of complex systems being identified and analyzed only based on intercepted traffic, such as the Japanese Purple system by the US or the Lorenz SZ42 system by the British. The Kerckhoffs's principle, formulated in the 19th and reformulated by Claude Shannon in 1949 specifies that a cipher system should be secure even if it falls into the hands of the enemy [12] [13]. Furthermore, transmission channels should by definition be considered insecure, and the assumption is that the enemy is able to intercept encrypted communications. The security of a cipher system should therefore rely on the inability of the enemy to recover the encryption keys from intercepted encrypted traffic, rather than rely on his lack of knowledge about the encryption system.

Cryptanalysis methods of attacking ciphers may be divided into the following categories:

- **Ciphertext-only attacks**: This is the most generic type of attack. Only the ciphertext is available, as well as the knowledge of the cipher system, but not of the specific key used to encrypt the plaintext.
- **Known-plaintext attack**s: In some cases, some parts of the plaintext, or the full plaintext, may be known or guessed, in addition to the matching ciphertext. In this case, it is still useful to try and recover the original encryption key, in order to read other messages encrypted using the same or similar keys. This attack is feasible, for example, when stereotyped beginnings or endings are used, or if the message plaintext was obtained using other methods.
- **Chosen-plaintext attacks**: With this method, not only the attacker knows the plaintext, but he actually selects the plaintext or plaintexts that will be encrypted and transmitted as ciphertext. This method is often the only one available when attacking modern ciphers, and is outside the scope of our research, as it was not relevant for classical ciphers.

For any type of cipher, there is always the possibility, at least theoretically, of a **brute-force attack**. The only exception is the one-time-pad, as even if we can test all possible keys, there is no way of determining which of the plaintexts (as well as the keys used to decrypt the ciphertext) is the correct one. For other ciphers, however, the most straightforward method to recover the encryption key is simply to exhaustively try to decipher the encrypted text with all possible keys. This type of attack is easy to implement for very simple ciphers such as the Caesar cipher, but for most classical ciphers, the size of the key space is simply too large, at least for manual brute-force cryptanalysis. Starting from the 1930s, brute-force attacks were sometimes implemented using machines, or machine-generated catalogs, such as for the cryptanalysis of Enigma systems without a plugboard. The feasibility of brute-force attacks must often be reassessed when considering new technology such as modern computing. Brute-force attacks, however, are not the focus of this work, and neither are ciphers which are susceptible to brute-force attacks.

A common approach to tackle a keyspace of huge size is a **divide-and-conquer** approach. With such an approach, the cryptanalyst tries to recover some part or parts of the key while ignoring other parts of the same key. This approach was often used for the cryptanalysis of systems with multiple stages of encryption, such as Enigma. In our research, we shall often rely on such an approach, especially for some of the more challenging ciphers. Divide-and-conquer approaches may sometimes be combined with a "semi-brute-force attack" on some parts of the key, while ignoring the other parts.

We may also divide cryptanalytic methods according to their use of technology. With **manual methods**, the cryptanalysis process, including possibly tedious statistical analysis, is performed by hand, using only pen and paper. This was the only option for codebreaking during WWI. Manual methods were still in extensive use during WWII, and thereafter, until the advent of computers. Manual cryptanalysis of ciphers was often considered as not only a science but also an art, leaving a lot of room for intuition and experience. **Mechanized methods**, and mechanical or electro-mechanical machines were developed in the 1930s. The Polish Bombe was developed for creating catalogs of Enigma "cycle patterns". WWII saw the increased use as IBM tabulating machines to replace relatively simple but tedious and error-prone processing by clerks or codebreakers. More sophisticated machines like the British Turing Bombe and the Colossus, were built during WWII, and were designed from the start to implement processes which could not practically be carried manually [9] [10]. The last category is **computerized methods** using general purpose computers, with technologies such as early mainframe computers, supercomputers, personal computers, and even distributed cloud-based computers. Computers have been used by code-breaking agencies such as the NSA, as early as the 1950s, but very little is known about their use and specific method for the cryptanalysis of specific codes [11]. Starting from the 1980s, an extensive body of research about computerized cryptanalysis of classical ciphers is available in the public domain.

1.3 Cryptanalysis of Classical Ciphers as an Optimization Problem

The advent of modern computing has opened the door for techniques which in the past were too tedious to be performed manually, or too expensive as they required the design and production of expensive machinery such as the Colossus and the Enigma Bombe. In some cases, a exhaustive brute-force search over the entire keyspace is now possible, such as for a transposition cipher with less than 15 elements. But computer power on its own is not enough for most cases of the more challenging classical cipher systems and machines. This was the motivation behind the use of optimization techniques for the cryptanalysis of classical ciphers, mainly based *local search metaheuristics*.

We first describe why the use of local search metaheuristics is relevant for most classical ciphers, while it is not relevant for most of the modern ciphers. Despite their increasing sophistication, classical ciphers and cipher machines are unable to completely hide statistical patterns. Using Shannon's terminology, classical ciphers have low or limited *diffusion* [13], that is, they are limited in their ability to hide statistical patterns. Also, if the cryptanalyst knows almost all (but not all) of the correct key elements, decrypting the ciphertext using such an almost correct key will usually produce an almost correct plaintext. Furthermore, a key with fewer errors will most probably produce a text with fewer decryption errors. If the number of errors exceeds a certain level, the decrypted text may not be readable at all. But in some cases, even though the decrypted text might not be readable, it may still reveal very subtle statistical characteristics, such as an Index of Coincidence value slightly higher than for a sequence of random letters [3].

In general, it may be said that a good classical cipher is not a cipher which hermetically hides all statistical characteristics in ciphertexts, but rather better hides those characteristics (e.g. with better diffusion). Some ciphers have very little diffusion, such as the monoalphabetic substitution cipher. Some cipher machines were able to implement better diffusion via sophisticated encryption mechanisms, but their complexity was limited by the electromechanical nature of those machines. The level of diffusion of a specific cipher may also vary based on the key settings. For example, a transposition cipher with a long key, or with two rounds of transposition, has better diffusion than with a single transposition or with short keys.

It is important to note that modern encryption systems, such as DES or AES, completely hide any of those statistical patterns, and were designed with high levels of diffusion. It is enough to have only one bit wrong in the key, in order to generate decrypted texts which are statistically equivalent to a sequence of purely random symbols. Furthermore, even a small change in the plaintext may result in drastic changes in the ciphertext.

Prior research has shown that it is possible to map the cryptanalysis of certain classical ciphers, into optimization problems, and more specifically, into stochastic local search problems [14] [15]. The **search space** is the key space. The goal is to find the optimal key, i.e. the correct key. Candidate keys are evaluated or using some scoring function. The score is based on a statistical measure, usually applied to the text obtained by decrypting the ciphertext using the candidate key. Ideally, the score for a key with more elements that are correct, should be higher than for a key with less correct elements. Furthermore, for the fully correct key (the original encryption key), the scoring function should typically reach its maximum value. The other components are a method for **generating an initial key**, a set of perturbations or **transformations** applied on the current candidate key to generate new candidates in its neighborhood, a method to decide whether or not to **accept** new key candidates, a **stopping criteria**, and a strategy to cope with **local maxima**.

Several local search metaheuristics have been proposed and investigated, for the cryptanalysis of classical ciphers such as tabu search, genetic algorithms, simulated annealing [16] [15]. Other metaheuristics used for the cryptanalysis of classical ciphers include ant colony optimization algorithms [17], particle swarm optimization algorithms [18], and Markov Chain Monte Carlo [19]. In most cases, the local search metaheuristics were mostly implemented in a straightforward manner, with minimal tailoring to the specific problem. Some works also compare several metaheuristics side-by-side. In most cases, the metaheuristics are applied for simple substitution ciphers, or to transposition ciphers with short keys, e.g. 15 to 25 elements and focusing on the more simple case of complete transposition rectangles. In general, those earlier case studies did not demonstrate any significant improvement, except automation, when compared to what could already be achieved with manual methods [14] [16] [15].

A breakthrough was achieved in the 1990s with more elaborate hill climbing algorithms, tailored to the cryptanalysis of the specific cipher. In 1995, Gillogly presented the first ciphertext-only cryptanalysis of the **Enigma** machine [20]. In WWII, the Allies only had a known-plaintext solution, using the Turing Bombe. Gillogly's method was refined by Weierud and Sullivan and applied for the successful decryption of hundreds of original German Army messages [21]. It was also used for the decryption of original German Navy 4-Wheel Enigma messages, using distributed computing. Tailored hill-climbing algorithms were also successfully applied for the cryptanalysis of the Japanese **Purple** cipher [22], and for the cryptanalysis of the British **Typex** system [23]. Simulated annealing was successfully applied for the cryptanalysis of short **Playfair** ciphers [24]. In contrast with early works using local search metaheuristics, the results were by far superior to what could be achieved in the past with historical methods. Interestingly, so

far genetic algorithms, although extensively researched in the context of classical ciphers, have not proved to be superior to other metaheuristics for any of the classical ciphers. In comparative studies, genetic algorithms were found to be no better than or comparable to simulated annealing or tabu search/hill climbing when applied to the cryptanalysis of the simple substitution cipher and the columnar transposition cipher [14] [16].

1.4 Challenges

In this section, the challenges in applying local search metaheuristics for the cryptanalysis of classical ciphers are described.

Hill climbing (HC) is one of the most widely used local search metaheuristics for the cryptanalysis of classical ciphers. With classical ciphers, improving a key, so that more elements are correct, usually results in gradual improvements in the deciphered plaintext. This is in contrast with modern ciphers with high diffusion, in which even a single error in the key completely or almost completely corrupts the decrypted text. Hill climbing has been proposed for the computerized cryptanalysis of manual ciphers such as substitution ciphers and transposition ciphers [16] [15], as well as machine ciphers such as Enigma [20]. Hill climbing, in its most generic form, is simple to implement, and in many cases performs similarly or better than other local search metaheuristics such as genetic algorithms. Hill climbing, however, has its limitations. In its generic form, it is often not powerful enough for the cryptanalysis of some of the more challenging classical ciphers and cipher machines, when used with strong and secure parameters.

The best-known limitation of hill climbing is the risk of getting stuck into local maxima before reaching a desired global maximum. This limitation is often remedied using techniques such as stochastic shotgun restarts (see Section 3.3), tabu lists [16] [15], or simulated annealing (SA) (Section 3.4). SA is essentially a variant of HC, which also allows some downhill transformations [16]. But there are other requirements for an HC or SA algorithm to be effective.

The two main elements in the design of a hill climbing algorithm are the **scoring function** for candidate keys, and the **transformations** on a key to allow the search to move to new candidate keys in its neighborhood. Ideally, the scoring function should reach its maximum value for the correct key. It should also be monotonic, i.e. the closer the candidate key is to the correct key, the higher the score should be. The requirement for a monotonic scoring function can be challenging (see Section 3.2). Often, there will be non-monotonic areas, i.e. a better key (with fewer errors) resulting in a worse score. Those cases may prevent hill climbing from converging towards the correct key. Also, when starting with a candidate key with many errors, correcting one or a few key elements may not result in any discernible improvement of the scoring function. Those cases would likely prevent hill climbing from "taking off", and the algorithm would not be able to reach an area with a strong enough gradient, from which it is able to converge towards the maximum. There might be a need for several scoring functions, such as a coarse initial scoring function sensitive enough to allow the algorithm to leave a non-monotonic area or noisy area, and a more selective scoring function applied at later stages, to better converge once a certain number of key elements have been correctly recovered. There are several generic scoring functions commonly used for the cryptanalysis of historical ciphers, such as n-grams statistics, but the selection of an appropriate scoring function, or the design of a new one, is a complex task, and what makes a scoring function effective is not clear a priori (see Section 4.5).

Those are not the only problems associated with the application of a simplistic HC algorithm for the cryptanalysis of classical ciphers. Randomly selecting an initial key is often not enough.

Such initial keys may be of too low quality, i.e. too deep into a flat or non-monotonic area of the scoring function. More sophisticated methods for generating good initial keys may be needed. The transformations used to create new candidate keys in the neighborhood of the current key might be highly too disruptive, for example by reshuffling too many key elements from a previous stage, and most of the knowledge gained in the previous steps of the search may be lost. There might be too many transformations to check at each iteration, significantly increasing the search time. On the other hand, checking too few transformations may cause the algorithm to miss opportunities for convergence. In other cases, the search may not be able to progress towards the correct key, if there is no series of single transformations that can produce increasingly raising scores and at the same time move towards the final correct key.

The underlying cipher system might be too complex and require a divide-and-conquer approach, or complex multiple sequential or nested local search phases.

Another challenge is related to the cryptanalysis of short messages. For any scoring method, there is a minimal length of the message under which the score is noisy and insignificant. This minimal length for statistical significance is usually greater than the Unicity Distance. The Unicity Distance [13] is the minimum length for a ciphertext, that ensures, with some probability, that this ciphertext, obtained by encrypting a plaintext P_1 with key K_1, may not be obtained by encrypting another plaintext P_2 with another key K_2. Below a certain plaintext length, spurious high scores, i.e. bad keys with a high score, may be generated by the search algorithm. Paradoxically, the more powerful the search method, the more likely it is to produce spurious scores, sometimes higher than the score for the correct key (see Section 3.2.4).

Another set of challenges refers to the difficulty in general to apply any statistical cryptanalytic techniques to some of the more secure types of classical ciphers, or when they are used with secure parameters. The first challenge is the size of the key space, which not only (and obviously) precludes the use of brute-force techniques, but also enhances the complexity of any search algorithm. The level of difficulty is increased in cipher systems with variable key length, such as the Double Transposition cipher. The longer the key, the larger the key space. Furthermore, in a well-designed cipher, with good diffusion, and when used properly, it may not be possible to obtain any statistical measure which is significantly different from the same measure on a random stream of symbols. The challenge for the cryptanalyst is to identify a potential deviation from randomness, that may be detected using some statistical measure, and to design a scoring method and key transformations to take advantage of this deviation. This problem is intensified with shorter cryptograms, when a measured deviation from randomness (in a ciphertext decrypted with a candidate key) may be due to statistical noise rather than due to the proximity of the candidate key to the correct key.

One of the implications of the challenges described here is that the effectiveness of a cryptanalytic attack needs to be assessed in the context of several parameters:

- **Amount of ciphertext**: The length of the cryptogram, or the total length of all cryptograms, if several cryptograms are available. For the text of a known-plaintext attack, this refers to the number of known or guessed symbols.
- **Key complexity**: For some of the ciphers, the length of the key may vary. A longer key means a larger keyspace and larger search space. For some cipher machines, some parameters may be more secure than others, for example, the overlap feature of the Hagelin M–209 (the more lug overlaps, the more secure the cipher).

An example of an analysis of the performance of various algorithms under different ciphertext lengths and key lengths may be found in Section 5.2.2.

Despite the development of new computerized techniques, as well as the availability of increasingly massive computing resources, no computerized solution has been yet published for several major classical ciphers and cipher machines, such as for the Swiss NEMA [25], the US SIGABA and KL-7 machines [26] [27], and the Russian Fialka [28]. For others, solutions are restricted in performance and only address the most advantageous cases, such as for the Hagelin M-209 [29], the Hagelin CX-52 [30] and the Double Transposition Cipher [31]. For many, not only is there no published effective computerized ciphertext-only attack, but neither there is any published computerized known-plaintext attack. A known-plaintext attack is usually easier to develop than a ciphertext-only attack. Also, even though more recent works produced encouraging results, and proved to be more effective than historical methods, no framework or formal methodology has been yet proposed with regards to what makes a local search algorithm effective for the cryptanalysis of classical ciphers. The outcome of research described in this thesis is a comprehensive, formalized and generalized methodology, strongly validated with case studies, for the application of local search metaheuristics for the effective cryptanalysis of challenging classical ciphers.

1.5 Relevance of the Work

The study of classical ciphers and their cryptanalysis is of high value for several reasons. It has its own historical value, to better understand the evolution of ciphers and codebreaking throughout history, and their roles, often hidden, in the course of historical events. Also, in some cases, some historical messages have been preserved only in their encrypted form, and may be decrypted and read only if one is able to perform a successful cryptanalysis (codebreaking) on the cipher. There are several cases where modern computerized methods have allowed for the decryption of those otherwise inaccessible original documents, such as the decryption of original Enigma messages from WWII [21], and of WWI German ADFGVX messages described in this thesis.

In educational settings, such as computer science academic programs, or in high schools, the study of classical ciphers is often instrumental in generating interest among students for cryptography and for computer science in general. In those settings, some of the principles of modern cryptography are often best introduced and understood in the context of classical cryptography examples. Furthermore, classical cryptography offers more opportunities for immediate rewards and increased motivation, in the form of codebreaking exercises and challenges. It also is possible to draw some lessons from past failures of classical ciphers, such as over reliance on the complexity of a cryptographic system, or on its keyspace size, rather than thoroughly investigating its cryptographic security. This might be even more relevant today, as the two main elements of classical cryptography, substitution and transposition, are still in use today in various forms, as part of building blocks for modern ciphers.

The study of the cryptanalysis of classical ciphers using modern techniques also can help understanding historical codebreaking techniques, as modern and historical methods often rely on the same statistical properties.

Although classical ciphers, on the surface, seem easier to cryptanalyze than modern ciphers, there are still quite a few unsolved historical challenges left, as well as classical ciphers for which there are no known effective cryptanalysis methods.

1.6 Contributions

The first goal of this research is to identify the factors of a cryptanalysis algorithm based on local search metaheuristics that make it effective, or not effective, based on the analysis of case studies, some successful (see Sections 3.5.1, 3.5.2 and 3.5.3), some less (Section 3.5.4). The second goal is to develop a new methodology for efficient cryptanalysis and formulate principles and guidelines for the design of effective algorithms based on local search metaheuristics. The third goal is to validate the methodology, applying it to a series of challenging classical cipher problems, including for the solution of cipher challenges and the decryption of original historical encrypted documents.

The main contributions of this research and thesis include:

1.6.1 Contribution 1 – A New Methodology for the Efficient Cryptanalysis of Classical Ciphers using Local Search Metaheuristics

This thesis presents a new methodology built upon five main guiding principles, which include major modifications, extensions, adaptations, and other recommendations. Those are intended to turn hill climbing and simulated annealing into highly effective tools for the cryptanalysis of classical ciphers, especially for challenging cipher and cryptanalysis problems, when naive implementation of hill climbing or simulated annealing has failed to produce sufficient results. The five main guiding principles include:

1. Identifying the most effective metaheuristics, in most cases hill climbing (and for specific cases, simulated annealing). This includes special multistage or nested processes.
2. Reducing the search keyspace, either by a classical divide-and-conquer approach, or using other methods.
3. Design and selection of effective scoring functions, closely tailored to the specific problem at hand or to a specific stage of the solution, and which meet some criteria necessary for good performance.
4. Effective transformations on the key, which provide good search coverage while ensuring steady progress towards the solution.
5. Multiple restarts with high-quality initial keys.

Those principles are presented in more detail in Chapter 4.

1.6.2 Contribution 2 – New Effective Cryptanalytic Attacks on Several Challenging Classical Ciphers or Cipher Settings

As part of case studies for this research, we applied the methodology to the cryptanalysis of several classical ciphers for which there was currently no known computerized solution, or the solution was limited in scope and performance, such as a solution applicable only to short keys. The criteria for including a classical cipher as a case study in this research was the ability to show a major impact, either by allowing a solution for a cipher for which no solution exists,

or a major improvement compared to existing solutions. For each such cipher, we applied one or more of the guiding principles, and we evaluated the performance, including measurements of success rate under various conditions, comparison with previous methods, and a work factor evaluation.

Significant improvements were achieved for the following cipher cryptanalysis problems:

- Double Transposition cipher using keys longer than 15 elements [32].
- Single Transposition cipher with long keys, at least 25 elements [33].
- Single Transposition cipher with incomplete transposition rectangles [33].
- ADFGVX cipher using transposition keys longer than 15 elements [34].
- M-209 cipher machine, known-plaintext attack on short messages [35].
- M-209 cipher machine, ciphertext-only attack [36].
- Chaocipher, known-plaintext attack [37].
- Chaocipher, ciphertext-only attack on messages in depth [37].
- Enigma, recovery of key settings based on a small number of double indicators (paper to be submitted).

Those case studies are presented in Chapters 5, 6, 7, 8, 9, and 10. Most of those new efficient cryptanalytic methods are now the state-of-the-art in the domain of the cryptanalysis of the specific classical ciphers.

1.6.3 Contribution 3 – Decipherment of Historical Documents and Solutions for Cryptographic Challenges

Some of the case studies led to the successful deciphering of original historic cryptograms or the solution of public cryptographic challenges, including:

- The deciphering for the first time since 1918 of a collection of 600 original war-time German messages encrypted with the ADFGVX cipher. The decipherment of the messages has enabled historians to gain new insights into developments in the Eastern Front in the last part of WWI [34].
- The Double Transposition Challenge from 2007 by Klaus Schmeh and Otto Leiberich [38] [32].
- The Hagelin M-209 Cipher Challenge from 2012 by Jean-François Bouchaudy [36].
- Hagelin M-209 cipher challenges from 1977 by Robert Morris, Greg Mellen, and Wayne Barker [36] [35].
- Chaocipher Exhibit 6 Challenge by John Byrne, Cipher Deavours and Louis Kruh [37].
- Winning an Enigma international contest organized in 2015 in memory of the achievements of Polish mathematicians, involving the recovery key settings based on a small number of double indicators.

1.7 Structure of the Thesis

This thesis is structured as follows: In Chapter 2, we provide a general background about stochastic local search. In Chapter 3, we present the application of local search metaheuristics to the cryptanalysis of classical ciphers, including relevant prior work. In Chapter 4, we describe a new methodology for the effective cryptanalysis of classical ciphers with local search metaheuristics. After that, we present a series of case studies, one per chapter, illustrating the application of the new methodology to challenging classical cipher problems, and each includes a description of the cipher system, related prior work, new attacks based on the methodology, and their evaluation. The last chapter concludes the findings of the research, and includes suggestions for further research.

2

Stochastic Local Search

This chapter provides background about applying stochastic local search (SLS) algorithms for combinatorial problems. We start with an introduction to combinatorial problems. Next, we describe search algorithms that may be applied to their solution. We then focus on a particular family of search metaheuristics, stochastic local search metaheuristics, which are relevant to the cryptanalysis of classical ciphers. A comprehensive survey of stochastic local search and its applications can be found in *Stochastic local search: Foundations and applications, 2004* by Hoos and Stützle [39]. This chapter is a summary of the main concepts and their definitions, as presented in the book, extended with examples and applications from the domain of cryptanalysis.

2.1 Combinatorial Problems

Combinatorial problems arise in many areas of computer science. This includes tasks such as finding shortest round-trips in graphs, scheduling, and resource allocation. The cryptanalysis of classical ciphers may also be viewed as combinatorial problems. The solution of these problems typically involves finding orderings or assignments of a set of objects which satisfy certain conditions. For a scheduling problem, the individual objects could be the events to be scheduled, and their values could be the times at which a given event occurs. For most combinatorial optimization problems, the space of potential solutions for a given problem instance is exponential in the size of that instance. For some cryptanalysis problems, such as the columnar transposition cipher, the size of the problem is exponential w.r.t. to the length of the key (although the problem complexity also depends on other factors such as a complete or an incomplete transposition rectangle). In most cryptanalysis problems, however, the size of the key is constant, and the size of the problem is also constant, albeit usually very large.

Hoos and Stützle distinguish between *problems* and *problem instances*. A problem is a generic or abstract problem, such as "for any given set of points in a plane, find the shortest round-trip connecting these points". An instance of this (generic) problem would be to find the shortest round-trip for a specific set of points in the plane. The solution of such a problem instance would be a specific shortest round-trip connecting the given *specific* set of points. The solution of the generic problem however, is a method or algorithm which, given *any* problem instance, determines a solution for that instance. A cryptology-related generic problem could be, with

regard to a specific cipher system, to find the encryption key, given any pair of plaintext and its corresponding ciphertext (a.k.a. a "known-plaintext attack"). An instance of this problem would be to find the key for a specific plaintext-ciphertext pair.

For instances of combinatorial problems, they also distinguish between *candidate solutions* and *solutions*. Candidate solutions are potential solutions that may be encountered during an attempt to solve the given problem instance; but unlike solutions, they do not have to satisfy all the conditions from the problem definition. In our cryptology problem example, any valid encryption key would be a candidate solution, while only those candidate keys which produce the given ciphertext (when encrypting the given plaintext with) would qualify as solutions.

Many combinatorial problems can be characterized as *decision problems*: for these, the solutions of a given instance are characterized by a set of logical conditions. Given a graph and a number of colors, the problem of finding an assignment of colors to its vertices such that two vertices connected by an edge are never assigned the same color – the *graph coloring problem* – is an example of a combinatorial decision problem. Other prominent combinatorial decision problems include finding satisfying truth assignments for a given propositional formula, the *propositional satisfiability problem* (SAT) or scheduling a series of events such that a given set of precedence constraints is satisfied. Our cryptology-related problem of finding the key which produces a given ciphertext, when applied on the given plaintext, is also a combinatorial decision problem (and some of those problems may also be modeled as SAT problems).

Hoos and Stützle also distinguish between two variants of decision problems: the search variant, where the goal is, given a problem instance, to find a solution (or to determine that no solution exists); the decision variant, in which for a given problem instance, one wants to answer the question whether or not a solution exists. These variants are closely related, as clearly, algorithms solving the search variant can always be used to solve the decision variant. Interestingly, for many combinatorial decision problems, the converse also holds: algorithms for solving the decision variant of a problem can be used for finding actual solutions. With cryptanalysis problems, we are typically concerned about searching for solutions (keys) rather than answering the question whether or not a solution exists.

Other combinatorial problems are *optimization problems* rather than decision problems. Optimization problems can be seen as generalizations of decision problems, where the solutions are additionally evaluated by an objective function and the goal is to find solutions with optimal objective function values. For the graph coloring problem mentioned above, a natural optimization variant exists, where a variable number of colors is used and the goal is, given a graph, to find a coloring of its vertices as described above, using only a minimal (rather than a fixed) number of colors. Any combinatorial optimization problem can be stated as a maximization or as a minimization problem, where often one of the two formulations is more natural. Algorithmically, maximization and minimization problems are treated equivalently. For combinatorial optimization the goal is to find a candidate solution with minimal (or maximal, respectively) objective function value. An alternative goal could be to find a candidate solution whose objective function value is smaller than or equal to some value (for minimization problems, or greater than or equal to for maximization problems). The cryptanalysis problem of finding the key when only the ciphertext is known (a.k.a. a "ciphertext-only" attack, which is the most generic case), is also combinatorial optimization problem. The goal is to find a key which reproduces, after decrypting with it a given ciphertext, the most "plausible plaintext".

Two particularly interesting classes of problems are P, the class of problems that can be solved by a deterministic machine in polynomial time, and NP, the class of problems which can be solved by a nondeterministic machine in polynomial time. The question whether $P = NP$, is

one of the most prominent open problems in computer science. Since many important problems with practical applications are in *NP*, but no polynomial time deterministic algorithm is known, this problem is not only of theoretical interest. For these problems, the best algorithms known so far have exponential time complexity. Therefore, for growing problem size, the problem instances become quickly intractable, even with massive computing power. Many of these hard problems from *NP* are closely related to each other and can be translated into each other. A problem, which is at least as hard as any other problem in *NP* (in the sense that each problem in NP can be reduced to it) is called NP-hard. NP-hard problems can be regarded as at least as hard as every problem in *NP*. But they do not necessarily have to belong to the class *NP* themselves, as their complexity might be higher. NP-hard problems which are contained in NP are called NP-complete; these problems are the hardest problems in *NP*. The SAT problem, the *traveling salesman problem* (TSP), as well as many other well-known combinatorial problems, including the graph coloring problem, the *knapsack problem*, many scheduling problems, are NP-hard or NP-complete. It suffices to find a polynomial time deterministic algorithm for one single NP-complete problem to prove that $P = NP$. Today, most computer scientists believe that $P \neq NP$.[1]

Although many combinatorial problems are NP-hard, it should be noted that not every computational task which can be formulated as a combinatorial problem is inherently difficult. A well-known example of a problem that, at first glance, might require searching an exponentially large space of candidate solutions, is the Shortest Path Problem. A simple recursive scheme known as Dijkstra's algorithm, can find shortest paths in quadratic time with respect to the number of vertices in the given graph. While a problem may be NP-hard, a subclass of the same problem may not. For example, the SAT problem for 2-CNF formulae is polynomially solvable [42].

Another approach for searching solutions to a combinatorial problem is to accept suboptimal candidate solutions instead of trying to find only optimal solutions. This way, in many cases the computational complexity of the problem can be sufficiently reduced to make the problem practically solvable. In some cases, allowing a comparatively small margin from the optimal solution makes the problem deterministically solvable in polynomial time. Sometimes, however, even reasonably efficient approximation methods cannot be devised or the problem is a decision problem, to which the notion of approximation cannot be applied at all. In these cases, one further option is to consider probabilistic rather than deterministic algorithms, as described in the following sections.

2.2 Search Algorithms

In this section, we introduce some key concepts, distinctive strategies, and tradeoffs when implementing search algorithms for the solution of hard combinatorial problems.

[1] Some attempts have been made to build modern cryptographic systems which rely on the difficulty of solving NP-complete problems. Those efforts have not been successful [40] [41]. One reason is that those problems are a reduced version of the general problem (e.g. knapsack in the Merkle-Hellman cryptosystem). A second argument is that NP-completeness refers to the worst-case instance of a problem, whereas cryptographic security is required for any instance. In the literature survey performed for this thesis, the author did not find any prior work which demonstrates that a specific cryptanalytic problem can be mapped into another NP-complete problem.

2.2.1 Search Algorithms for Hard Combinatorial Problems

Basically all computational approaches for solving hard combinatorial problems can be characterized as **search** algorithms. The fundamental idea behind the search approach is to iteratively generate and evaluate candidate solutions; in the case of combinatorial decision problems, evaluating a candidate solution means to decide whether it is an actual solution, while in the case of an optimization problem, it corresponds to determining the respective value of the objective function. Although for NP-hard combinatorial problems the time complexity of finding solutions can grow exponentially with instance size, evaluating candidate solutions can often be done much more efficiently, i.e., in polynomial time. For example, for a given TSP instance, a candidate solution would correspond to a round-trip visiting each vertex of the given graph exactly once, and its objective function value can be computed easily by summing up the weights associated with all the edges used for that round-trip.

2.2.2 Perturbative vs. Constructive Search

Typically, candidate solutions for instances of combinatorial problems are composed of atomic assignments of values to objects, such as the assignment of truth values to individual propositional variables in the case of SAT, or in cryptanalysis, the assignment of values to individual elements of the key. Given candidate solutions can easily be transformed into new candidate solutions by modifying one or more of the corresponding atomic assignments. This can be characterized as perturbing a given candidate solution. Hoos and Stützle classify search algorithms which rely on this mechanism for generating the candidate solutions to be tested as *perturbative* search methods. Applied to SAT, perturbative search would start with one or more complete truth assignments and then in each step generate other truth assignments by changing the truth values of a number of variables in each such assignment. Applied to the cryptanalysis of a columnar transposition cipher (see Chapter 5), perturbative search would start with some random transposition key, and in each step generate new transposition keys by swapping the contents of any two key elements.

While for perturbative approaches, the search typically takes place directly in the space of candidate solutions, it can sometimes be useful to also include partial candidate solutions in the search space, i.e. candidate solutions for which some atomic assignments are not specified. An example for such partial assignments is a partial round-trip for a TSP instance, which corresponds to paths in the corresponding graph that visit a subset of the vertices and can be extended into a full cycle by adding additional edges. Algorithms for solving this type of problem are called *constructive* search methods or construction heuristics. An example of a constructive search for cryptanalysis is the recovery of the Enigma plugboard settings (see Section 3.5.1). For some problem instances, it is known that a valid solution should include exactly 10 plugboard connections. A search may start with no connections at all, then incrementally add, replace or remove connections, until an optimal set of 10 connections is found. This implies that the search space also includes partial solutions, with less than 10 connections.

2.2.3 Systematic vs. Local Search

Hoos and Stützle also make the distinction between *systematic search* and *local search*: Systematic search algorithms traverse the search space of a problem instance in a systematic manner which guarantees that eventually either a solution is found, or, if no solution exists, this fact

is determined with certainty. This typical property of algorithms based on systematic search is called completeness. The equivalent of a systematic search for cryptanalysis is "brute force", in which all valid keys in the keyspace are systematically surveyed to find optimal solutions (the original encryption key).

One method to systematically survey all possible solutions is *backtracking*. With backtracking, the space of the possible complete solutions is represented by the leaves of a tree. The other nodes of the tree represent partial solutions. We start at the root of the tree, selecting the first child, and continue from there to the bottom of the tree, at which time we have a complete solution, which we evaluate. After that, we backtrack, going back to the previous node, and select the next child who has not been visited, and repeat the process from this child. We stop when all the nodes, and therefore, all the leaves (complete solutions), have been visited. Such backtrack algorithms tend to be exponential. Fortunately, it is often possible to prune some branches of the tree (subtrees), without the need to visit its nodes. Such an approach is commonly known as *branch & bound*.

Local search algorithms, on the other hand, start at some location of the given search space and subsequently move from the present location to a neighboring location in the search space, where each location has only a limited number of neighbors and each of the moves is determined by a decision based on local knowledge only. Typically, local search algorithms are incomplete, i.e., there is no guarantee that an existing solution is eventually found, and the fact that no solution exists can never be determined with certainty. Furthermore, local search methods can visit the same location within the search space more than once. In fact, many local search algorithms are prone to get stuck in some part of the search space which they cannot escape from without special mechanisms like a complete restart of the search process or some other sort of diversification steps.

The remainder of this chapter focuses on local search approaches and algorithms.

2.3 Stochastic Local Search

Many widely known and high-performance local search algorithms make use of randomized choices in generating or selecting candidate solutions for a given combinatorial problem instance. These algorithms are called *stochastic local search* algorithms, and they constitute one of the most successful and widely used approaches for solving hard combinatorial problems, such as the Traveling Salesperson Problem. As we describe in the next chapters, they also have been extensively used for the cryptanalysis of classical ciphers.

In this section, we describe the structure of a generic stochastic local search approach, and its components.

2.3.1 Overview of Stochastic Local Search

Local search algorithms generally work in the following way. For a given instance of a combinatorial problem, the search for solutions takes place in the space of candidate solutions, also called the *search space*. Note that the search space may include partial candidate solutions, in the context of constructive search algorithms. The local search process is started by selecting an initial candidate solution, and then proceeds by iteratively moving from one candidate solution to a neighboring candidate solution, where the decision on each search step is based on a limited

amount of local information only. In stochastic local search algorithms, these decisions as well as the search initialization can be randomized. According to Hoos and Stützle [39], a stochastic local search algorithm consists of the following elements:

- The *search space* of a problem instance, which is a set of *candidate solutions* (also called search positions, configurations, or states).
- A set of *feasible solutions*, included in the set of candidate solutions.
- An *initialization function*, to select an initial candidate, from which the search starts.
- A *neighborhood function*, which indicates for each candidate solution, the set of neighboring candidates, to which the search may move.
- An *evaluation function* applied on a candidate solution.
- A *step function*, specifying the local search steps, to decide on the next move from a candidate to one of its neighbors, usually relying on the evaluation function for guidance.
- A *termination* predicate determining whether the search is to be terminated upon reaching a specific point in the search space.

2.3.2 Evaluation Functions

One way to progress in a search is to randomly select one of the current solution neighbors. This might be sufficient for simple instances of a problem, but in most cases, a mechanism is needed to guide the search towards solutions in a more effective manner. This can be achieved using an evaluation function which maps each search space position (candidate solution) onto a scalar value in such a way that the global optima correspond to the solutions. Typically, this evaluation function is used for assessing or ranking candidate solutions in the neighborhood of the current search position. The efficacy of the guidance thus provided depends on properties of the evaluation function and its integration into the search mechanism being used. Typically, the evaluation function is problem specific and its choice is to some degree dependent on the search space, solution set, and neighborhood underlying the search approach under consideration. In Section 3.2.2, we introduce a number of evaluation functions commonly used for the cryptanalysis of classical ciphers. In the case studies, we introduce more specialized functions, tailored to the specific cipher problem.

Other commonly used terms referring to evaluation functions include “fitness function”, used in the context of maximization problems, and “cost function”, used in the context of minimization problems. In this thesis, we frequently use the term *scoring function*, and it is interchangeable with “evaluation function”.

2.3.3 Iterative Improvement

One of the most basic local search strategies is *iterative improvement*. Iterative Improvement starts from a randomly selected point in the search space, and then tries to improve the current candidate solution w.r.t. the evaluation function. The initialization function is typically implemented by generating a randomly selected initial state (uniformly, so that the probability of

picking any state is uniform). The step function selects the next state from the set of all neighboring candidate solutions. A (neighboring) candidate state will be selected only if its evaluation score is higher than for the current state.

This strategy (or *metaheuristic*) is also known as iterative descent or *hill climbing*. The latter name, which we use throughout this thesis, is motivated by the application of Iterative Improvement to maximization problems. In the case where for a given candidate solution none of its neighbors corresponds to an improvement w.r.t. the evaluation function, the step function cannot select a new candidate. A state with no neighboring state having a higher evaluation score is denoted as *local maxima*. Solutions which correspond to *global maxima* of the evaluation function, are also considered local maxima. In cases where a local search algorithm guided by an evaluation function encounters a local maximum that does not correspond to a solution, this algorithm can get stuck. Very few combinatorial problems may be mapped into search problems in which there is only one local maximum, which is also the global one. Unless the local maximum also corresponds to the global maximum, a strategy is required, to escape from local maxima, or to reduce the probability of being stuck in them.

One simple way of modifying Iterative Improvement such that local maxima are dealt with more effectively, is to simply start a new search whenever a local maximum is encountered. Alternatively, one can relax the improvement criterion and, when a local maximum is encountered, randomly select one of the non-improving steps. In either case, it cannot be guaranteed that the search algorithm effectively escapes from arbitrary local maxima, because the nature of a local maximum can be such that after any such "escape step", the only improving step available may lead directly back into the same local maximum.

A more sophisticated method to escape from local maxima is simply to modify the evaluation function when a local maximum is reached using a certain function. This can be done by assigning weights or penalties on certain elements of the solution, and modifying those weights whenever the iterative improvement process gets trapped in a local maximum. This general approach provides the basis for a number of algorithms called *dynamic local search* (DLS). This approach has been applied to the SAT problem, improving the performance of the search algorithm [43].

Another strategy, *tabu search*, combines the simple iterative improvement (hill climbing) which restarts itself when it reaches a local maximum, with a list of candidate forbidden solutions. Typically, those are local maxima recently visited during the search. This ensures that the search will not get stuck in the same local maxima.

2.3.4 Intensification vs. Diversification

The strong randomization of several elements of local search algorithms, i.e. the utilization of stochastic choice as an integral part of the search process, can significantly increase their performance and robustness. However, with this potential comes the need to balance randomized and goal-directed components of the search strategy, a trade-off which Hoos and Stützle characterize as *diversification vs. intensification*. *Intensification* refers to a search strategy which aims to greedily improve solution quality or the chances of finding a solution in the immediate future by exploiting, for instance, the guidance given by the evaluation function. *Diversification* strategies try to prevent search stagnation by making sure that the search process achieves reasonable coverage when exploring the search space, and does not get stuck in relatively confined regions in which at some point no further progress can be made. In this sense, Iterative Improvement is an intensification strategy. A large variety of techniques for combining and balancing

intensification and diversification strategies have been proposed. The successful application of these algorithms is often based on experience rather than on theoretically derived principles. In this context, problem-specific knowledge is often crucial for achieving peak performance and robustness. We address the topic in the context of cryptanalytic problems in Chapter 3.

2.3.5 Large vs. Small Neighborhoods

The performance of any stochastic local search algorithm depends significantly on the underlying neighborhood relation and, in particular, on the size of the neighborhood. One of the most widely used types of neighborhood relations is the so-called *k-exchange neighborhoods*, in which two candidate solutions are neighbors if they differ in k solution components. When using the standard k-exchange neighborhoods it is easy to see that for growing k, the size of the local neighborhoods (i.e. the number of direct neighbors for each given candidate solution), also increases exponentially with k. On the other hand, larger neighborhoods generally contain more and potentially better candidate solutions, and hence they typically offer better chances of facilitating locally improving search steps. This situation illustrates a general tradeoff: Using larger neighborhoods might increase the chance of finding (high quality) solutions of a given problem in fewer local search steps when using stochastic local search algorithms in general and Iterative Improvement in particular; but at the same time, the time complexity for determining improving search steps is much higher in larger neighborhoods. Typically, the time complexity of an individual local search step needs to be polynomial (w.r.t. the size of the given problem instance), where depending on problem size, even quadratic or cubic time per search step might already be prohibitively high.

One way to benefit from the advantages of large neighborhoods without incurring a high time complexity of the search steps is based on the idea of using standard, small neighborhoods until a local optimum is encountered, at which point the search process switches to a different (typically larger) neighborhood, which might allow further search progress. This approach is known as is *variable neighborhood search* and is based on the fact that the notion of a local optimum is defined relative to a neighborhood relation, such that if a candidate solution is locally optimal w.r.t. one neighborhood relation, it need not be a local optimum for a different neighborhood relation, with a larger neighborhood size [44].

2.3.6 Best Improvement vs. First Improvement

The most widely used neighbor selection strategies are the so-called *best improvement* and *first improvement* strategies. Iterative best improvement selects, in each search step, the neighboring candidate solution which results in a maximal improvement in the evaluation function. Best improvement requires a complete evaluation of all neighbors in each search step (which often can be done in parallel). The first improvement neighbor selection strategy tries to avoid the time complexity of evaluating all neighbors by selecting the first improving step encountered during the inspection of the neighborhood. Obviously, the order in which the neighbors are evaluated can have a significant influence on the efficiency of this strategy.

As in the case of large neighborhoods, there is a tradeoff between the number of search steps required for finding a local optimum and the computation time for each search step. Typically, for first improvement search steps can be computed more efficiently than when using best improvement, since only a small part of the local neighborhood is evaluated by first improvement. However, the improvement obtained by each step of first improvement local search is typically

smaller than for best improvement and therefore, more search steps have to be applied to reach a local optimum. On the other hand, best improvement increases the chances for meeting a local optimum earlier in the process.

2.3.7 Probabilistic vs. Deterministic Neighbor Selection

Both, best improvement and first improvement as described in Section 2.3.6 are deterministic in nature, given a specific evaluation function, a current candidate solution, and its set of neighbors. Instead, a probabilistic approach may be employed.

A simple probabilistic approach for neighbor selection (and relevant only for first improvement strategies) consists of reviewing the neighbors in a random order. This introduces an additional level of diversification.

A more sophisticated approach is employed by *simulated annealing* local search algorithms. While improving moves (w.r.t. the evaluation function) are always accepted, moves that decrease the evaluation function may also be accepted, based on a probabilistic function. This probabilistic function depends on the amount of degradation w.r.t. the evaluation function. We describe simulated annealing in more detail in Section 3.4.

2.3.8 Single Candidate vs. Population of Candidates

So far in this chapter, the discussion about search algorithms was limited to a state which contains only one current solution. At each step of the search, the neighbors of this current solution are evaluated, and one of them selected. The search relies on a neighboring function which maps each candidate solution to a set of valid neighboring solutions.

Another approach is to have a *population* with multiple current solutions. This approach is employed by *genetic algorithms*. Genetic algorithms are inspired by models of the natural evolution of species. They transfer the principle of evolution through mutation, recombination, and selection of the fittest, which leads to the development of species which are better adapted for survival in a given environment, to solving computationally hard problems. Genetic algorithms are iterative, population-based approaches: starting with a set of candidate solutions (the initial population), they repeatedly apply a series of three genetic operators, selection, mutation, and recombination. Using these operators, in each iteration of a genetic algorithm, the current population is (totally or partially) replaced by a new set of individuals; in analogy with the biological inspiration, the populations encountered in the individual iterations of the algorithm are often called generations.

The selection operator implements a probabilistic choice of individuals for the next generation and for the application of the mutation and recombination operators. Better individuals have a higher probability of being selected. Mutation is an operation on individuals which introduces small, often random modifications. Recombination is an operation which generates a new individual (called the offspring) by combining information from two parents. The most commonly used types of recombination mechanism are called crossover; these assemble pieces from a linear representation of the parents into a new individual. One major challenge in the design of recombination, is to combine parents in a way that the resulting offspring is likely to share desirable properties of the parents while improving over their fitness.

Intuitively, by using a population of candidate solutions instead of a single candidate solution, a higher search diversification can be achieved, particularly if the initial population is randomly selected. The primary goal of genetic algorithms for combinatorial problems is to evolve the population such that good coverage of promising regions of the search space is achieved. However, genetic algorithms may lack sufficient search intensification.

2.3.9 Smooth vs. Rugged Search Landscape

While most combinatorial problems are multidimensional search problems, with a large number of dimensions, it may be useful to visualize the search space as a "landscape", with (multidimensional) mountains, hills, plateaus and valleys. The "altitude" of a position (a candidate solution) is determined by the evaluation function applied to that candidate solution. The goal of a search algorithm is to find the highest peak in the landscape. The term "hill climbing" originates from that analogy. Ideally, the highest peaks in the landscape should represent optimal solutions. The neighbors of each position in the landscape are determined by the neighboring function. The *distance* between every two positions is the (minimum) number of moves required to reach one position from the other one, using only neighboring moves as defined by the neighboring function. The search landscape therefore depends on the specific problem instance, the specific evaluation function, and the specific neighboring function.

A *smooth* landscape means there is some degree of correlation between the evaluation scores of two neighboring positions (solutions) in the landscape. That correlation usually decreases as the distance between two positions increases. The opposite of the smoothness of a landscape is similarly called the *ruggedness* of the landscape. There are various generic measures of smoothness (or ruggedness), such as the *search landscape correlation function* [45]. A rugged search landscape is (intuitively) more likely to have more local optima than a smoother one. Local search algorithms rely on iterative improvement and progressing from a candidate solution to one of their neighbors, using the evaluation function for guidance.

We illustrate the concepts of landscape smoothness and ruggedness, with several simulated 3D plots. Those plots do not represent any actual problem, as any non-trivial problem would have a search space with more than two dimensions.[2]

In Figure 2.1, we illustrate a very smooth landscape. With such an ideal smooth landscape, an iterative search (hill climbing) algorithm is guaranteed to succeed, regardless of its starting point, as there is only one local maximum. In Figure 2.2, there are multiple local maxima, but the landscape is relatively smooth, and local search with a simple diversification strategy (e.g. multiple restarts) is likely to succeed. With a rugged landscape as in Figure 2.3, a local search with a stronger element of diversification may be required, but could still succeed. A chaotic landscape as in Figure 2.4 would not allow any local search algorithm to succeed, and a smoother evaluation function, other than the one used in this landscape, would be required.

[2] Those illustration plots were obtained using MATLAB, with functions of the form $\sum_i \frac{\sin(x^{a_i})}{x} \cdot \frac{\sin(y^{b_i})}{y}$.

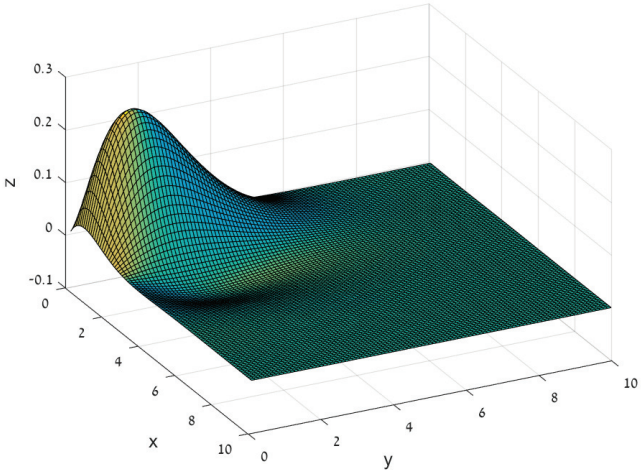


FIGURE 2.1: Local search – illustration of an ideal landscape

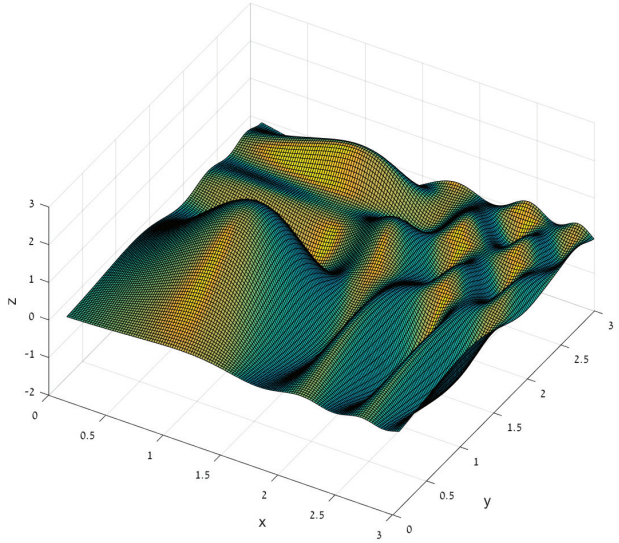


FIGURE 2.2: Local search – illustration of a smooth landscape

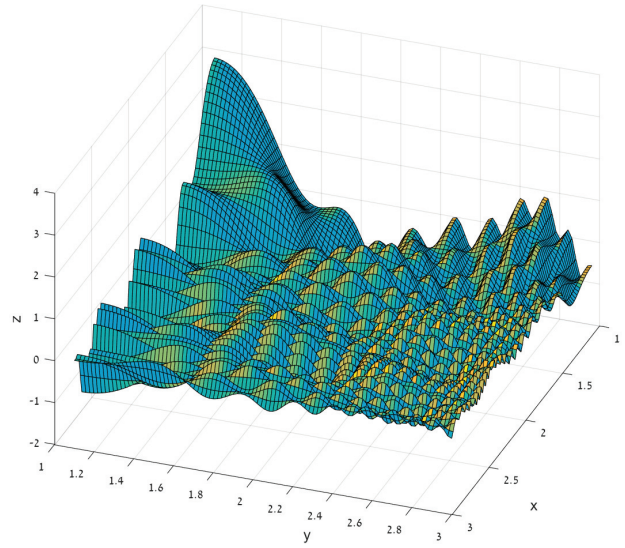


FIGURE 2.3: Local search – illustration of a rugged landscape

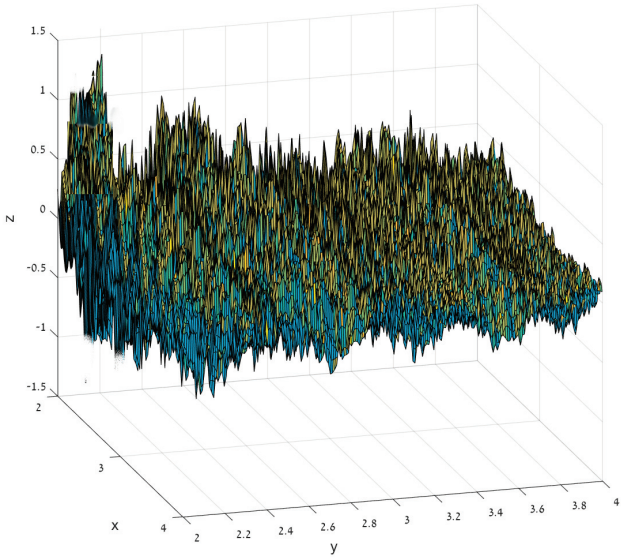


FIGURE 2.4: Local search – illustration of a chaotic landscape

2.3.10 Fitness-Distance Correlation

A key element of the analysis of a search landscape is the correlation between the evaluation score of any position (candidate solution) and its distance to a solution (global optima). The distance is measured as the number of neighboring moves required to reach the solution from the candidate solution. The higher the correlation, the higher the probability a local search algorithm will be able to progress in the right direction. The intuition for a strong correlation is that the evaluation score is more likely to improve as we get closer to the solution (global optimum), and therefore the guidance provided by the evaluation function is more effective.

A scalar metric has been proposed to measure that correlation, the *fitness-distance correlation coefficient* (FDC coefficient) [46]. Instead of relying on a single scalar value, a plot, conveniently called the *fitness-distance plot* may be employed, to visualize the relationship between distance and evaluation scores. An example of such a plot is given in Figure 2.5. It can be seen that fitness (the Y-axis) usually decreases when the distance (the X-axis) increases.

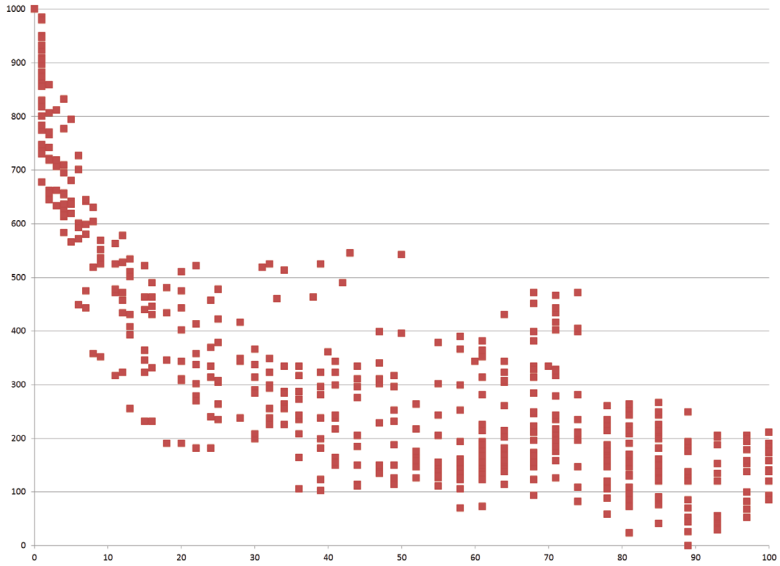


FIGURE 2.5: Local search – fitness (Y-axis) – distance (X-axis) plot

In general, with combinatorial problems, the analysis of fitness-distance correlation is difficult because it is impossible to completely measure its behavior over a large search space. As a substitute, random simulations of problem instances may be used, and samples of fitness-distance pairs of values recorded. In most combinatorial problems with a large problem size, even this partial solution is often not feasible, as we may only simulate problem instances for which the solution is known. For example, we can easily create a complex TSP instance, with a large number of points in the plane, but we most likely do not know the solutions (global optima) to this problem. As a result, FDC analysis might be possible only using simple instances of the problem, and those may not be representative of the more complex cases, which often are the cases of higher interest.

In contrast, for cryptanalysis problems, the analysis of fitness-distance correlation is often easier. We can easily create any number of instances of a cryptanalysis problem for simulation purposes, for which we have the full information about the distance of candidate keys to the correct key, since we "know" the solution, the correct key, which we used to create the ciphertext. FDC analysis not only is feasible for cryptanalysis problems, but it is a powerful tool, as we shall see in Section 3.2.4.

2.3.11 Local Search Metaheuristics vs. Local Search Algorithms

A *local search metaheuristic* is not an algorithm, but it is rather a set of strategies employed when implementing local search algorithms. A metaheuristic is a framework in which specific choices have been made, such as employing probabilistic vs. deterministic neighbor selection, or the use of a single candidate vs. a population of candidates. This framework specifies how the various components of local search shall be implemented, but it does that in a high-level manner, leaving room for problem-specific adaptations.

Commonly used local search metaheuristics include:

- Hill climbing (iterative improvement)
- Simulated annealing
- Tabu search
- Genetic algorithms
- Ant colony optimization

In the next chapter, we describe the application of local search metaheuristics for the cryptanalysis of classical ciphers.

3

Cryptanalysis of Classical Ciphers using Local Search Metaheuristics

In Chapter 2, we introduced local search metaheuristics. In this chapter, we describe how they are applied for the cryptanalysis of classical ciphers, focusing on hill climbing and simulated annealing. We also present related prior work, and briefly describe why the use of the local search metaheuristics is ineffective for the cryptanalysis of modern ciphers.

3.1 Cryptanalysis as a Combinatorial Problem

The cryptanalysis of any cipher is essentially a combinatorial problem (see Section 2.1). A cryptographic key is composed of several discrete elements, and the goal is to find the combination of key elements which results, after decrypting the ciphertext with it, in the most plausible plaintext.

For any given type of cipher, there is usually a finite number of possible keys, or a finite keyspace. Theoretically, a cipher system may be designed in such a way that its key space is not finite. For example, the columnar transposition cipher may have a transposition key of any arbitrary length. In practice, the keyspace is limited by some constraints on the use of the cipher, such as by setting a limit on the length of the transposition key. Historically, columnar transposition keys had usually between 5 to 25 elements. Even the most secure type of cipher, the one-time pad, has a finite number of possible keys, limited by the amount of one-time pad material that can be physically exchanged by the communicating parties. Anyhow, in all known and practical uses of historical ciphers, there have always been some limitations on the key, resulting in a finite keyspace.

If the keyspace is small enough, a frontal attack in the form of a brute-force systematic search (see Section 2.2.3) over the full keyspace may be considered. For example, the size of the DES cipher keyspace is

$$72,057,594,037,927,936 = 2^{56} \tag{3.1}$$

A successful brute-force attack on DES was demonstrated already in the late 1990s, and today may be completed in less than a day on specialized hardware. A brute-force attack on a

columnar transposition cipher with a key of length 13, with a keyspace of size $2^{32.5}$ ($13! = 6,227,020,800 = 2^{32.5}$), is practical today even on a home PC.

Still, most of the classical ciphers used during the 20th Century have a keyspace size too large to allow for a brute-force attack. For example, a simple monoalphabetic substitution cipher for a language with 26 characters has a key space of size $26! = 2^{88.4}$. A columnar transposition cipher with a key of 25 elements has a keyspace of size $25! = 2^{83.6}$. In the following chapters, we present examples of classical ciphers with even larger keyspaces. A large keyspace does not necessarily mean that the cipher is secure, even if it prevents brute-force attacks. For example, despite its large keyspace, the monoalphabetic substitution cipher is a highly insecure cipher.

Even if a large keyspace does not allow a direct brute-force attack based on the systematic survey of all possible keys, it is still possible to solve some classical ciphers using a deterministic combinatorial algorithm. The most obvious case is when the cryptanalyst has knowledge of specific limitations on the key. For example, some of the elements of the key may change for each message (the "message key" or "external key"), while other elements ("the internal key") may stay constant throughout a single day (or month). If the daily key is known, it might be possible to conduct a brute-force search for the other elements, which change from message to message. Another similar situation arises for some cipher systems, if the plaintext message or part of it is known or can be guessed. The Turing-Welchman Bombe developed in WW2 was essentially a combinatorial search (brute-force) device used to recover the Enigma rotor settings [47]. The Bombe required the knowledge (or guessing) of some parts of the plaintext. Such a pairing between ciphertext and corresponding plaintext enables the cryptanalysis to deduce some constraints on the possible keys. Those constraints may be enough to reduce the relevant keyspace to a size that is tractable via brute-force attacks.

Even if a brute-force attack is feasible, either on the full keyspace or on part of it, a method is required to determine whether a given key is the correct key. In the case of a known-plaintext attack, the most straightforward method is to check that after decrypting the ciphertext with a candidate key, the expected plaintext is accurately reproduced. For ciphertext-only attacks, some other measure of correctness or scoring function is needed, as described in Section 3.2.

However, in most cases, the keyspace of classical ciphers is very large, and cannot be reduced to a tractable smaller size. Instead of brute force, we need more efficient search algorithms. This is the motivation for the use of (stochastic) local search metaheuristics (see Chapter 2) for the cryptanalysis of classical ciphers. The most straightforward local search metaheuristic is **hill climbing**, described in Section 3.3, which is the primary focus of this research, and has been successfully applied to a number of hard classical cryptanalysis problems. A second metaheuristic also found to be effective for classical cryptanalysis is **simulated annealing**, also covered in this research, and described in Section 3.4. Other local search metaheuristics have been proposed and applied for the cryptanalysis of classical ciphers, with mixed results. More details are provided in the following sections.

3.2 Scoring Functions for Cryptanalysis Problems

The scoring function (a.k.a. as the evaluation function – see Section 2.3.2) is a critical element of search algorithms, and often, the most critical one. The scoring function allows the search algorithm (either a systematic or local search), to evaluate the quality of a candidate key in the keyspace, and to determine whether one candidate key is better than another key. In this section, we present some scoring functions commonly used for known-plaintext and for ciphertext-only

attacks. We also describe the attributes that make a scoring function effective for the cryptanalysis of classical ciphers.

3.2.1 Introduction

Systematic ("brute-force") search, and (stochastic) local search both require scoring functions to evaluate candidate keys. In Chapter 2 we used the term *evaluation function*, but in this chapter and the next ones, we use the term *scoring function*. The score is a measure we either want to maximize ("fitness function"), or to minimize (a "cost function"). Straightforward examples of fitness and cost functions, used in a known-plaintext attack, is to count the number of characters correctly reproduced, and the number of characters incorrectly reproduced, respectively, after decrypting the ciphertext with a candidate key. Unless mentioned otherwise, all the cryptanalytic search problems described in this thesis are mapped to maximization problems, that is, the goal of the search algorithms is to find the key with the highest score.

In formal terms, a scoring function for a ciphertext-only attack is a function applied on a candidate key K_i and on the ciphertext C, which we denote as $S(K_i, C)$. In the case of a known-plaintext attack, the parameters for the scoring function also include the known plaintext P, that is, $S(K_i, C, P)$. The ciphertext C, as well as the plaintext P in the case of a known-plaintext attack, are constant throughout the search.

In most cases, however, the scoring function for a candidate key K_i is applied on $P_i = E^{-1}(C, K_i)$, the putative plaintext obtained after decrypting C with the candidate key K_i, where E^{-1} is the decryption function, the inverse of E, the encryption function. We denote a scoring function S applied on the putative decrypted plaintext as $S(P_i)$, or simply $S(P)$.

3.2.2 Scoring Functions for Known-Plaintext Attacks

As mentioned before, the most straightforward scoring function for a known-plaintext attack is simply counting the number characters correctly reproduced in the putative decrypted plaintext.

A more subtle scoring method could be to compute the distance of between a character in the putative decrypted plaintext, and its corresponding character in the known plaintext, and to sum all those distances. This alternative method is particularly effective with additive ciphers, where the additive part (added to the plaintext symbol) is composed of several elements, such as for the Hagelin M-209 (see Section 7.4.2.4).

3.2.3 Scoring Functions for Ciphertext-Only Attacks

In the case of a ciphertext-only attack, we do not know the plaintext. But in general, the language of the plaintext is known. Scoring functions for ciphertext-only attacks usually rely on the statistical features of that language. A straightforward measure for a putative decryption, is to count the number of dictionary words which appear in the putative decryption. The main drawback of this method, is the fact that in order for a word to be correctly reproduced, a number of consecutive characters (the letters of a word) must be accurately reproduced in the putative decryption. This will rarely happen if the decrypted text has a large number of errors. It might be said that such a scoring function has a very low **resilience to key errors** (see Section 3.2.4), and every key with more than a few errors will have a zero or close to zero score. On the

other hand, this scoring function is **selective** (see Section 3.2.4). If we decrypt the ciphertext with a candidate key K_i, and the decrypted text contains a large number of dictionary words are reproduced, K_i is likely to have more correct elements than another candidate key K_j, if decrypting the ciphertext with K_j reproduces very few dictionary words.

Another scoring function consists of counting the occurrences of each character (monograms), and compare this count to the frequency of monograms in a large corpus of the target language. For example, it is expected that in a typical English text, there will be more occurrences of E or T than occurrences of the letters Q and Z. This method is more resilient to decryption errors, as it does not require adjacent characters to be correctly reproduced. On the other hand, scoring functions based on monogram statistics are less selective than word-based scoring functions. A candidate key with very few correct elements may nevertheless produce, after decrypting the ciphertext with it, a text which contains a mix of letters with frequencies similar to those of the target language, but mostly reproduced at wrong places. Such a key would obtain a high monogram score, even though it has a large number of errors.

As a compromise between word-based scoring functions, which are selective but less resilient to key errors, and scoring functions based on monogram statistics, which are less selective but more resilient to key errors, **n-gram** statistics may be employed. For the case of bigrams (pairs of letters), with $n = 2$, we count the number of occurrences of each bigram, e.g. TH or QA, and compare their frequencies to the bigram frequencies in the language. For English, we expect TH to appear more frequently than QA. As this method requires at least some of the adjacent characters to be correctly reproduced, so that correct bigrams may be reproduced, this method is less resilient to key errors than with monograms. On the other hand, it is more selective, as it relies on sequences of two characters. Other commonly used n-gram scoring functions include trigrams ($n = 3$), quadgrams ($n = 4$), pentagrams ($n = 5$), and even hexagrams ($n = 6$). The higher the n of the n-gram, the more selective the function is, but the less it is resilient to key errors.

There are two main approaches for computing scoring functions based on n-grams. We denote the probability of the i-th n-gram appearing in the language as G_i. For a putative decryption P, we denote the relative frequency of the i-th n-gram as F_i, with $F_i = \frac{n_i}{N}$, n_i being the count of its occurrences in P, and N the length of P. The first method consists of computing the sum of the squares of the difference between the actual frequency of each n-gram and its expected frequency (in the language), as follows:

$$S(P) = \sum_{i=1}^{c} (F_i - G_i)^2 \tag{3.2}$$

c is the number of possible n-grams. For English and monograms, $c = 26$, and for bigrams, $c = 26^2 = 676$. Similarly, for higher-order English n-grams, $c = 26^n$.

A second approach, proposed by Sinkov in [48], uses logarithmic values of n-gram probabilities (G_i), as follows:

$$S(P) = \sum_{i=1}^{c} F_i \log(G_i) \tag{3.3}$$

Summing up the log of the probabilities of bigrams is equivalent to computing the log of their products. In our research, and in most related works, this second approach is employed, as it allows for more efficient algorithms than when using the first approach.

In general, n-gram or word-based scoring functions are language-specific. They depend on the language statistics, or require a dictionary. Other scoring functions are not language-specific. Instead, they are based on a common characteristic of all natural languages, that is, redundancy. According to Shannon, redundancy is a measure of how much a text in the language can be reduced in length without losing any information [13]. For example, a human may often be able to read an English sentence after all its vowels have been removed. Another manifestation of redundancy in a language is the fact that certain letters (monograms) are more likely to appear in a specific language than others, for example, E and T compared to X or Z in English. Redundancy, however, is not limited to monograms. It is also reflected in bigrams (TH vs. XZ), as well as higher-level n-grams (ING vs. XIS). Redundancy in natural languages is the basis for highly efficient text-compression algorithms. Redundancy is also (inversely) related to randomness. A random stream of characters does not display any redundancy, that is, no character is expected to appear at a higher frequency than another one. For a truly random sequence, this is also true of n-grams of any order (bigrams, trigrams, etc.). Ideally, the ciphertext produced by a well-designed cipher shall be able to hide any type of language redundancy, as well as any other statistical patterns of the language. Except for the one-time pad, this was very difficult if not impossible to achieve with classical ciphers, before the advent of modern cryptography.

The most well-known scoring function which relies on the language redundancy and does not require language-specific statistics, is the **Index of Coincidence**, or IC, introduced by William F. Friedman [3]. The IC is the probability of any two characters in a text being identical. Formally, the IC is defined as

$$IC = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{N(N-1)/c} \tag{3.4}$$

where N is the length of the text, c is the number of letters of the alphabet, and n_i is the count of the i-th letter of the alphabet. The sum of the n_i is necessarily N. The product $n_i(n_i - 1)$ counts the number of combinations of n_i elements taken two at a time. (Actually this counts each pair twice; the extra factors of 2 occur in both numerator and denominator of the formula and thus cancel out.) Each of the n_i occurrences of the i-th letter matches each of the remaining $n_i - 1$ occurrences of the same letter. There are a total of $N(N-1)$ letter pairs in the entire text, and $1/c$ is the probability of a match for each pair, assuming a uniform random distribution of the characters.

While the IC of a text only depends on the contents of that text, and may be computed without any knowledge of the underlying language, it is also possible to compute the value of the expected IC in a text in a certain language. To compute that average language IC, we assume that N, as well as all the individual n_i values, are very large, and therefore, $(n_i - 1)/(N-1) = n_i/N = G_i$, and:

$$IC_{\text{expected}} = \frac{\sum_{i=1}^{c} {G_i}^2}{1/c} \tag{3.5}$$

If all c letters of the alphabet were equally distributed, the expected IC would be 1.0. For comparison, the actual IC for telegraphic English text (without spaces or punctuation) is around 1.73, reflecting the unevenness of natural-language letter distributions, and the redundancy in the language. In most cases, however, IC values are reported without the normalizing denominator:

$$IC_{\text{expected}} = \sum_{i=1}^{c} {G_i}^2 \tag{3.6}$$

For example the expected IC for English is 0.067=1.73/26. Similarly, the IC (without the normalizing denominator) for a uniform (random) distribution of the same alphabet with 26 characters is $0.0385 = 1/26$.

Throughout this thesis, we use the IC without the normalizing denominator:

$$IC = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{N(N-1)} \tag{3.7}$$

An important characteristic of the IC of monograms is that it is invariant to a monoalphabetic substitution. If we apply a substitution, each monogram is replaced in 3.7 by its substitute, but as there is a one-to-one mapping between the original monograms and their substitutes, the sum of the products is the same, albeit in a different order.

The use of the IC is not limited to monograms. It may also be applied to bigrams or to higher level n-grams. Such higher-order n-gram IC scoring functions, the IC of bigrams and quadgrams, were used for the cryptanalysis of the ADFGVX cipher (see Section 6.4).

The IC has been widely used for the cryptanalysis of classical ciphers, such as modern cryptanalysis of Enigma (see Section 3.5.1) and Purple (see Section 3.5.2). In this research, the IC is applied for the cryptanalysis of the ADFGVX (Section 6.4), the Hagelin M-209 (Section 7.3.2.5), and the Chaocipher (Section 8.4.2). One of the strengths of the IC as a scoring function, is that it can be applied even when a monoalphabetic substitution is combined with another encryption method, as the IC values of a text before and after substitution are the same, as explained above. A good example is the ADFGVX cipher, which consists of a monoalphabetic substitution followed by a columnar transposition. It is possible to search for the transposition key, by measuring the IC of n-grams after undoing the transposition, and ignoring the substitution. This allows for an effective divide-and-conquer attack (see Section 6.4). A similar approach was used for Chaocipher (Section 8.4.2). In contrast, n-gram scoring for a text after monoalphabetic substitution is meaningless, as the n-grams have been altered by the substitution, and therefore their counts and frequencies will not match the expected frequencies of n-grams in the language. This holds both for a standalone monoalphabetic substitution, or for cascade ciphers which include a monoalphabetic substitution.

The main strength of the IC as a scoring function, however, is its increased resilience to key errors, when applied to classical ciphers. Most other scoring functions are already ineffective above a small-to-moderate number of key errors. In contrast, the IC is often able to differentiate, between a key with a relatively high number of errors, and a similar key with fewer errors. For that reason, IC is often the scoring function of choice in the early stages of a key search, when initial keys have a large number errors. IC is less effective and often not selective enough in the later stages of the key search, when the current key is close to the correct key, and has very few

of errors. At this stage, scoring functions based on n-grams will be more effective in order to converge towards the correct key, as they are more selective.

Another drawback associated with IC-based scoring functions, is the higher probability for **spurious high scores** to be produced during the key search, that is, a high IC score given to a candidate key which is mostly incorrect. This spurious high IC score may be even higher than the IC of the correct plaintext. This is also relevant for other scoring functions with good resilience to key errors, e.g. monograms. Spurious scores are more likely to occur with short ciphertexts, and less likely with longer ones.

3.2.4 Selectivity vs. Resilience to Key Errors

In Section 2.3.9 we introduced the intuitive concept of the search landscape. An important requirement is that the landscape, which is affected by choice of the scoring (evaluation) function, be smooth to a certain extent. Another important measure is the Fitness-Distance Correlation (FDC), best visualized using fitness-distance plots (see Section 2.3.10). The term fitness refers here to the evaluation score. The term distance refers to the number of neighboring moves needed to reach the global optimum (the correct key) from the candidate key. A count of the number of errors in the key may also be used as an alternative measure of distance. In the FDC analyses performed as part of our research, we use count of key errors as the primary distance measure. A strong FDC may be intuitively associated with the monotonicity of the scoring function.

While we may intuitively assume that a good FDC helps a local search being more effective, a single FDC scalar measure may too coarse to reflect some subtle aspects of local search performance, when applied for the cryptanalysis of classical ciphers. In our analyses, we differentiate between the FDC in two ranges of distances, a longer range of medium-to-long distances, and a shorter range of short-to-medium distances, closer to the solution. As stated in Section 2.3.10, with cryptanalytic problems, we have the advantage of being able to simulate FDC behavior on a large variety of problem instances, and not just the simpler ones, in contrast with most combinatorial search problems.

In the previous sections, we referred to two attributes of a scoring function, selectivity, and resilience to key errors, when used for the cryptanalysis of a classical cipher. We define here those two terms in a (slightly) more formal manner:

- **Selectivity**: For a scoring function to be considered as selective, then given two candidate keys with a **low-to-moderate** number of errors, the one with fewer errors should more likely obtain a better score. Furthermore, the correct key should obtain the highest possible score. Also, for short ciphertexts, a selective scoring method should also prevent or mitigate spurious scores, that is, high scores obtained by decrypting the ciphertext with a key mostly or entirely incorrect. Word-based, or higher-order n-gram scoring functions are usually more selective than IC or lower-order n-grams.
- **Resilience to key errors**: Given two candidate keys with a **moderate-to-high** number of errors, the key with the smaller number of errors should more likely obtain a better score. IC-based scoring functions are more resilient to key errors than monograms. Higher-level n-grams have poor resilience to key errors.

Note that those attributes are relevant only to long key lengths (e.g. transposition ciphers with 15 or more elements), or to cipher systems with a non-trivial number of key elements (e.g. the Hagelin M-209, which has a total of 131 pins and 27 bars) and a large key space.

The selection of the most effective scoring function is most often the result of a trade-off between selectivity and resilience to errors. As a general rule, for a complex classical cipher cryptanalysis problem, with a large key space, it is often preferred to start the search with a scoring function which is more resilient to key errors, such as IC or monograms. Higher-order n-gram functions, which are more selective, are more appropriate after some of the key elements have already been recovered (and thus the number of key errors is limited). In particular, when most of the key elements have been correctly recovered, a selective scoring function is often needed to converge to the (entirely) correct key. When the ciphertext is short, there are fewer choices. With short ciphertexts, scoring functions found to be more resilient to key errors with longer ciphertexts, are often ineffective due to spurious high scores. For short ciphertexts, the use of a more selective scoring function, which does not generate spurious high scores, is mandatory, even though that scoring function may have a low resilience to key errors. Those considerations will be further elaborated in Section 4.5, and in the case studies.

We illustrate the tradeoff between selectivity and resilience using fitness-distance plots in Figures 3.1, 3.2 and 3.3, for various lengths of Hagelin M-209 ciphertexts. In each plot, the Y-axis represents the evaluation score (fitness), and the X-axis the distance (measured as the percentage of key errors). We compare the behavior of two scoring functions, applied to the decryptions of the ciphertexts using a series of putative keys, in which random errors have been incrementally introduced, starting from the correct key settings and no errors (0%), until all the elements of the key are incorrect (100% of errors). The first function is based on (log) bigrams and has a high selectivity (but low resilience to key errors). The second function is based on IC and has better resilience to key errors, but lower selectivity. We compare their behavior in the context of several ciphertext lengths.

In Figure 3.1, the ciphertext is relatively long (3 000 symbols), and the spread for each function is limited. It can be seen that with 40% of errors or more, the bigram-based function is flat, that is, it is not resilient to errors above this percentage. On the other hand, it is highly selective, and will give a score of 200 or more only if there are less than 10% of errors (all scores have been normalized to the range 0-1000). Conversely, the IC-based function is more resilient, has some (negative) non-zero gradient up to 60-70% of errors. On the other hand, it is less selective, and would give a score of 200 even to keys with 30% of errors.

Figure 3.2 represents FDC for a ciphertext of medium length (1 500 symbols). It can be seen that there is a wider spread of scores, and that the IC-based function has lost some of its resilience, and displays a gradient only with less than 55% of errors in the key.

Figure 3.3 represents FDC for a ciphertext of short length (500 symbols). The spread of the scores is much higher than for longer ciphertexts. The IC-based function has still some gradient (on average), but there is now a large number of spurious high scores. Those result from keys which obtain a high score, uncorrelated to the number of errors (distance). As a result, this function is ineffective for short ciphers, as it not only has a lower selectivity, but it also has lost most of its resilience to key errors, due to the spurious scores.

To summarize the fitness-distance analyses of those two functions, the conclusion is that for medium-to-long ciphertexts, it is better to start the search with the more resilient function (IC-based), which allows the search to progress even if there are 60-70% of errors in the initial key.

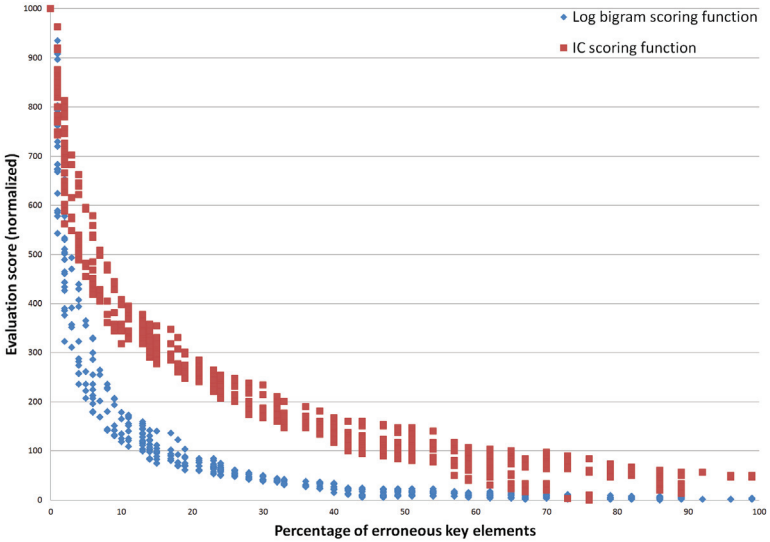


FIGURE 3.1: Scoring functions – fitness-distance plot for long Hagelin M-209 ciphertext (3 000 symbols)

Later, when additional key elements have been recovered, best is to continue with a more selective function (n-grams). For short ciphertexts, due to spurious scores, only a highly selective function can be used, but the search using that scoring function requires higher quality initial keys, with fewer initial errors.

3.2.5 The Unicity Distance and Scoring Functions

In cryptography, the unicity distance is the minimal length of ciphertext needed to recover the key using a brute-force attack, systematically checking all possible keys, so that there is just one key that produces a plausible decipherment. It was introduced by Claude Shannon in 1949 [13]. This minimal length ensures that no spurious keys are found in the process, that is, keys different from the original encryption key, which produce a plausible plaintext when used to decipher the ciphertext. The expected unicity distance can be computed as follows:

$$U = \frac{H(K)}{D} \tag{3.8}$$

where U is the unicity distance, and $H(K)$ is the entropy of the key space K. D is the plaintext redundancy in bits per character, which differs per language, and is defined as:

$$D = \log(L) - H(L) \tag{3.9}$$

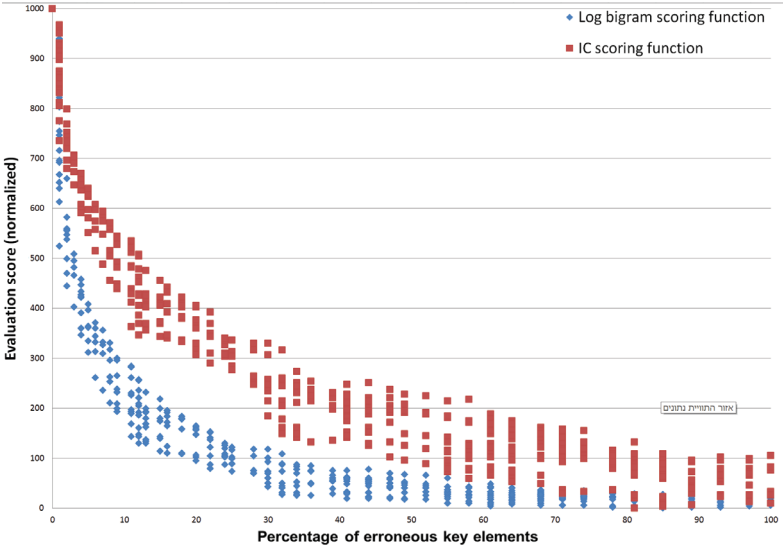


FIGURE 3.2: Scoring functions – fitness-distance plot for medium-length Hagelin M-209 ciphertext (1 500 symbols)

where L is the space of the language alphabet (e.g. 26, A to Z, for English), $\log(L)$ is the maximum amount of information that may be carried per character ($\log_2(L) = 4.7$ bits for an alphabet of 26 characters), and $H(L)$ the actual amount of information carried on average by a single character in a text in that language. For English, $H(L) = 1.5$ and therefore $D = 4.7 - 1.5 = 3.2$ [49].

For a monoalphabetic substitution cipher, the number of possible keys is $26! = 4.0 \cdot 10^{26} = 2^{88.4}$, which is the number of ways the alphabet can be permuted. Assuming all keys are equally likely, then $H(K) = \log_2(26!) = 88.4$ bits. For English texts, $U = \frac{88.4}{3.2} = 28$. Therefore, at least 28 characters of ciphertext are required for an unambiguous decryption of an English plaintext encrypted with a monoalphabetic substitution cipher.

Similarly, a columnar transposition cipher with a key of length 26, also has the same unicity distance. Additional values of U for various classical ciphers may be found in [50]. For example, for Playfair, $U = 24$. In [51], the unicity distance for a 3-rotor Enigma with plugboard is estimated to be around 20. In [52], the unicity distance for the Hagelin M-209 is estimated to be 47 (interestingly, our new known-plaintext algorithm described in Section 7.4 requires at least 50 characters). For the one-time-pad cipher, there is in theory an infinite number of possible keys, and therefore, $U = \infty$, but in practice, the space of the possible keys is bound by the length of the plaintext.

A brute-force attack on ciphertexts shorter than U may result in several keys producing plausible texts, only one of them being the correct one. There is no guarantee that spurious keys will not be found when using local search metaheuristics, and those spurious keys would be an additional source of spurious high scores, i.e. high scores obtained by wrong keys. While some scoring methods may be more selective and/or more resilient to errors than others, no scoring method

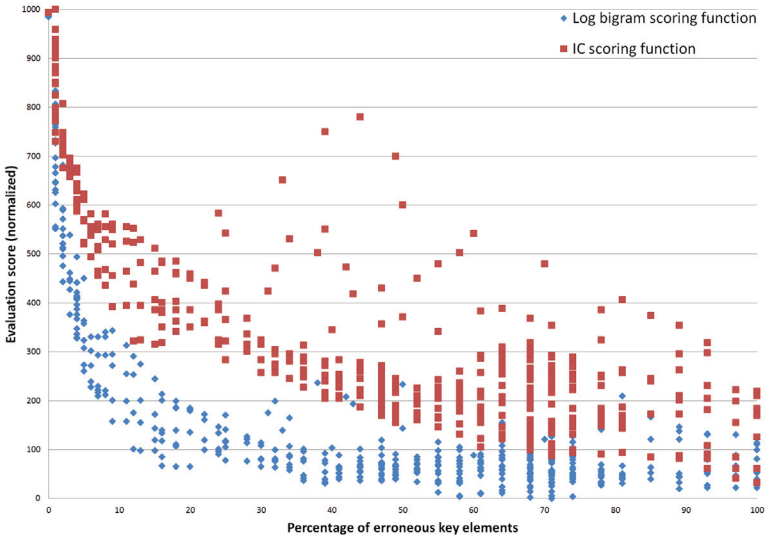


FIGURE 3.3: Scoring functions – fitness-distance plot for short Hagelin M-209 ciphertext (500 symbols)

can be protected from spurious keys and the resulting spurious scores obtained with ciphertexts shorter than U. Furthermore, the fact that a ciphertext is longer than U does not mean that there exists a cryptanalytic attack capable of recovering the key for ciphertexts of that length.

3.2.6 Extended Definitions of the Unicity Distance

In [52], Reeds proposes two extensions of the unicity distance (the first one was also proposed by Deavours in [50]).

With the first extension, instead of using the redundancy of the language, Reeds uses the *redundancy of the statistical measure* employed in a specific cryptanalytic attack. The argument is that the logic and structure of any language are highly complex, and it is difficult to design a cryptanalytic attack that takes advantage of all aspects of the redundancy in a language. Instead, practical cryptanalytic attacks rely on some attribute of the language, such as the statistics of n-grams, or the IC, which express some aspects of the redundancy in the language. For example, some bigrams such as TH and IN are much more frequent than bigrams such as ZK or QT.

Reeds gives an example, for the monoalphabetic substitution cipher, of a cryptanalytic attack based on bigram statistics. Reeds estimates the per-letter entropy of a theoretical language in which the next letter is selected according to the bigrams probabilities of normal English to be 3.57, and therefore the redundancy for such a language is $\log_2 26 - 3.57 = 4.7 - 3.57 = 1.13$. As a result, the "effective" unicity distance for a cryptanalytic attack which relies on bigrams statistics is $U_{eff} = log_2(26!)/1.13 = 78$. Deavours provides a slightly different value, with $U_{eff} = 65$ [50] for bigrams. For trigrams, Deavoirs estimates that $U_{eff} = 55$, and for octagrams (8-grams),

$U_{eff} = 38$. For comparison, Shannon's unicity distance for the monoalphabetic substitution cipher is 28. It might be argued that because higher-order n-grams have shorter U_{eff}, they should be preferred when implementing a cryptanalytic attack. This may be true when the ciphertexts are short, as lower-order n-gram measures are more likely to generate spurious high scores, and only a highly selective scoring method (e.g. high-order n-grams) may be employed. The main drawback of scoring functions based on higher-order n-grams, is their decreased resilience to key errors, which may prevent a local search algorithm from progressing after starting from a random initial key.

For a cryptanalysis attack on the Hagelin M-209 using monograms, Reeds estimates U_{eff} to be 491 [52], compared to 47 using Shannon's original definition. Interestingly, this is close to the limit of our new ciphertext-only algorithm described in Section 7.5, which requires 500 characters, and uses monograms as the scoring function.

Reeds proposes a second extension to Shannon's unicity distance, this time using an *alternative representation of the cipher system* [52]. His argument is that for some cipher systems, it might be possible to design an attack on a superset of the cipher system, whereas an attack on the target cipher system (a subset of the extended system) might be too difficult to implement. The drawback is obviously the fact that the extended cipher system is most likely to have a larger keyspace.

Reeds illustrates this extended unicity distance concept with the Hagelin M-209. The main components of the Hagelin M-209 are a set of pinwheels, and a rotating cage composed of 27 bars (see Section 7.2.1). It is possible to represent the function of the cage, as a vector with 64 entries, each entry having a value from 0 to 25. Due to the nature of the Hagelin cage, not all the possible assignments of this vector are feasible (for example, all values from 0 to 25 must be represented at least once in the vector). Reeds was able to develop an attack, described in Section 7.3.2.6, which is applicable to the case of a theoretical extended Hagelin cipher system, in which the cage has been replaced with a cage, for which all possible assignments are allowed. This system is a superset of the real Hagelin M-209, and obviously, has a larger keyspace. Reeds's attack relies on the Index of Coincidence. For the specific problem, Reeds estimates an upper bound for the redundancy of the IC measure to be $D = 0.357$ [52]. When taking into account the keyspace of this extended system, Reeds concludes that for this extended system and for this attack, $U_{eff} = 1210$. Reeds also states that according to his experiments, this number is very close to the minimum ciphertext length needed by his attack in order to succeed in practice (see Section 7.3.2.6).

Reeds concepts of extending the unicity distance for specific statistical measures and for alternative cipher system representations, and in particular, his results for the Hagelin M-209, are promising. More research is needed to assess whether those concepts can be effectively applied to other cipher systems, in order to analytically compute some bounds for the best performance theoretically achievable when using a certain statistical measure (or scoring function), or a certain model of the cipher system.

3.3 Hill Climbing for Classical Ciphers

In this section we present the primary local search metaheuristic employed for the cryptanalysis of classical ciphers, hill climbing (see Iterative Improvement in Section 2.3.3).

Algorithm 1 describes the application of hill climbing to a generic cryptanalysis problem. We start with a random key, and iteratively try to improve it by checking neighboring keys. A neighbor key is a key obtained via a small change or transformation on the current best key *BestKey*. The algorithm stops when no more improvement can be achieved by applying transformations on the current key.

Algorithm 1 Hillclimbing algorithm for a classical cipher

```
1:  procedure HILLCLIMBING(C)                                   ▷ C = ciphertext
2:      BestKey ← RandomKey()
3:      repeat
4:          Stuck ← true
5:          for CandidateKey ∈ Neighbors(BestKey) do    ▷ Iterate over neighbors of BestKey
6:              if S(CandidateKey,C) > S(BestKey,C) then
7:                  BestKey ← CandidateKey                          ▷ Found a better key
8:                  Stuck ← false
9:                  break
10:     until Stuck = true
11:     return BestKey
```

The main drawback of such an algorithm is that it will often return a local maximum (local best key), and not necessarily the best key over the whole keyspace. To overcome this limitation, hill climbing is often combined with multiple random restarts, also known as "shotgun restart hill climbing". The modified algorithm is listed in Algorithm 2

Algorithm 2 Shotgun restart hill climbing algorithm

```
1:  procedure SHOTGUNHILLCLIMBING(C,N)                  ▷ C = ciphertext, N = rounds
2:      BestGlobalKey ← RandomKey()                                  ▷ Best key overall
3:      for I = 1 to N do
4:          BestKey ← RandomKey()                                    ▷ Best local key
5:          repeat
6:              Stuck ← true
7:              for CandidateKey ∈ Neighbors(BestKey) do          ▷ Iterate over neighbors
8:                  if S(CandidateKey,C) > S(BestKey,C) then
9:                      BestKey ← CandidateKey                   ▷ Found better local key
10:                     if S(BestKey,C) > S(BestGlobalKey,C) then
11:                         BestGlobalKey ← BestKey              ▷ Found better global key
12:                     Stuck ← false
13:                     break
14:         until Stuck = true
15:     return BestGlobalKey
```

3.4 Simulated Annealing for Classical Ciphers

In this section, we present the simulated annealing local search metaheuristic, also found to be effective for the cryptanalysis of classical ciphers (see Probabilistic Neighbor Selection in Section 2.3.7).

Simulated annealing (SA) is a random-search metaheuristic which exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum (or maximum) in a more general system. It forms the basis of an optimization metaheuristics for combinatorial search and other problems. Simulated annealing was developed to deal with highly nonlinear problems. SA approaches a minimization problem similarly to using a bouncing ball that can bounce over mountains from valley to valley. It begins at a high "temperature" which enables the ball to make very high bounces, allowing it to bounce over any mountain to access any valley, given enough bounces. As the temperature declines the ball cannot bounce so high, and it may settle to bounce within a smaller range of valleys, until it finally reaches the bottom of one of the valleys. Note that for cryptanalysis problems, we most often need to solve a maximization problem (and find the highest mountain), unlike the illustration above.

SA's major advantage over other methods is the ability to avoid becoming trapped in local optima. The algorithm employs a random search which not only accepts changes that improve the scoring function S, but also some changes with a worse score. The latter are accepted with a probability of

$$p = e^{-\frac{|d|}{T}} \tag{3.10}$$

where d is the change in the scoring function S. T is a control parameter, which by analogy with the original application is known as the system "temperature". The higher the temperature T, the higher the probability of a new worse state being accepted. At the beginning of SA, as the temperature is high, the process behaves like a random search over a wide range of states, and the majority of worse states are accepted, in addition to new states with a better score. At the end of the process, when the temperature is low, the process is more similar to hill climbing, where only improving changes are accepted.

Applying SA to a cryptanalysis process is very similar to applying hill climbing. The search space is the keyspace. As the scoring function, we may use any of the functions applicable to hill climbing, and the same applies to the types of transformations (changes) applied on the key at each step. A typical SA algorithm applied to cryptanalysis of a classical cipher is described in Algorithm 3:

Algorithm 3 Simulated annealing algorithm

```
1:  procedure SIMULATEDANNEALING(C, N, T_0, α)      ▷ N = SA rounds, α = cooling factor
2:      BestKey ← CurrentKey ← RandomKey()
3:      T ← T_0
4:      for I = 1 to N do
5:          for CandidateKey ∈ Neighbors(CurrentKey) do      ▷ Iterate over neighbors
6:              D ← S(CandidateKey, C) − S(CurrentKey, C)
7:              if D > 0 or Random(0..1) < e^(−|D|/T) then
8:                  CurrentKey ← CandidateKey                ▷ New key accepted
9:                  if S(CurrentKey, C) > S(BestKey, C) then
10:                     BestKey ← CurrentKey                 ▷ Found a better global key
11:                 break
12:         T ← α · T                                        ▷ Reduce temperature
13:     return BestKey
```

Note that in the inner loop of the SA algorithm above, all the neighbors of the current key (*NeighborKeys*(*CurrentKey*)) are tested. This is in contrast with many SA implementations

(such as for the Playfair cryptanalysis described in Section 3.5.3), in which only a random subset of transformations is tested. In the methods developed as part of this thesis and that are based on SA, all neighbors are tested by the inner loop.

A common variation of the SA algorithm consists of maintaining a constant temperature throughout the various rounds, instead of reducing the temperature after each round. Such a constant temperature SA approach was used for the Playfair cryptanalytic method described in Section 3.5.3.

Finally, as with hill climbing, the basic SA algorithm is usually invoked multiple times, each time starting with a different key, to increase the overall probability of success.

3.5 Related Work

In this section, we present a survey of related work in which local search metaheuristics have been applied for the cryptanalysis of classical ciphers, starting with case studies where state-of-the-art performance was achieved, HC for Enigma and Purple and SA for short Playfair cryptograms. We conclude with an overview of why local search metaheuristics are less relevant for the cryptanalysis of modern ciphers, while they are highly relevant for classical ciphers.

3.5.1 Ciphertext-Only Cryptanalysis of Enigma

The Enigma machines were a series of electro-mechanical rotor cipher machines developed and used in first half of the twentieth century to protect commercial, diplomatic and military communication. Enigma was invented by the German engineer Arthur Scherbius at the end of WWI. Early models were used commercially from the early 1920s, and adopted by military and government services of several countries, most notably Nazi Germany. Several Enigma models were produced, but the German military models, equipped with a plugboard, were the most complex.

The Enigma machine consists of 3 or 4 rotors, each implementing a substitution of 26 alphabet letters. The 4-rotor models were introduced by the German Navy in 1942. All the other Enigma models have 3 rotors. The rightmost rotor always steps after a character is encrypted, and the other rotors step only when the rotor on their right reaches a certain position. As a result, the leftmost rotor rarely steps during the encryption of a single message, whose length is usually shorter than 250 letters. When the operator presses a key on the keyboard, electrical current flows from the key, via the plugboard connections, through all the 3 (or 4) rotors, to a reflector, and back again through the rotors, and back again through the plugboard connections. A full description of the machine can be found in [4]. A description of the 3-rotor model can be found in this thesis, in Chapter 10.

In the 1930s, Polish mathematicians developed methods for the ciphertext-only cryptanalysis of Enigma, which relied on the fact that some elements of the message key (the “message indicator”) were transmitted with encryption, but twice [53]. We describe their methods in detail in Chapter 10. After the “double indicator” procedure was discontinued in 1940, those methods were not relevant anymore, and new methods were developed, this time by British codebreakers at Bletchley Park, led by Alan Turing. The primary method relied on the “Turing Bombe”, a mechano-electrical device designed by Turing and Gordon Welchman in 1940. The Turing Bombe implemented a known-plaintext attack, which requires the plaintext or part of it to be

known or correctly guessed [47]. With the Bombe, British codebreakers were able to recover daily keys, thus facilitating the decryption of other Enigma messages encrypted using those same daily keys.

In 1995, Gillogly published a new method for the ciphertext-only cryptanalysis of the Enigma machine [20], based on hill climbing. Gillogly's method was refined by Weierud and Sullivan [21]. Weierud's method implements a divide-and-conquer semi-brute-force attack. All the possible rotor settings are exhaustively tested. The rotor settings include the selection and order of the rotors, the ring settings and the rotor starting positions. For each such rotor settings, hill climbing is applied, to recover the plugboard settings. At first, an empty set (or random set) of plugboard connections is selected, then the plugboard settings are modified, each time testing a new possible connection between a pair of plugs, or the option of disconnecting an existing connection. Special care is required in case one of the plugs, or both of them, are already connected. At each step, all possible swaps of two plugs are tested. Scoring is adaptive (i.e. it varies according to the state of the search). First, the IC, a scoring function with high resilience to errors, is used. When no more improvement can be achieved with IC, log bigrams are used, and finally, log trigrams, those scoring functions being more selective. Weierud and Sullivan applied their method for the successful decryption of hundreds of original German Army messages. The method was found highly effective for messages with around 180 letters, and in general with 100 or more. The same method was also used by Krah for the decryption of original German Navy 4-rotor Enigma messages, using distributed computing [54].

In 2017, Ostwald and Weierud further improved this method to support the cryptanalysis of very short messages, or messages with a large number of garbles [51]. Rather than starting with an initial empty or random set of plugboard connections, their algorithm implements a preliminary phase designed to produce higher quality initial plugboard settings. This preliminary phase exhaustively searches for all possible plugboard connections involving the letters most frequently used in German Enigma messages, either the top six (E, N, R, X, S, or I) or the top four (E, N, R, or X). Those connections which result in the higher IC score (after decryption) are selected for the main hill climbing phase, which is similar to Sullivan and Weierud. With this method, Ostwald and Weierud were able to decipher a set of short and often garbled original Enigma messages from the Second World War, which could not be deciphered using the Weierud and Sullivan method, including messages with about only 30 letters. Ostwald's and Weierud's method applies several adaptations to hill climbing, rather than using a naive implementation of hill climbing, including:

- Hill climbing, with multiple restarts, and a preliminary phase generating high-quality initial plugboard settings (the optimal connections for the most frequent letters).
- Divide and conquer with semi-brute-force search for rotor settings, and hill climbing for plugboard connections.
- Adaptive scoring approach – starting with IC with good resilience to errors, then using bigrams and trigrams with better selectivity.

As we shall see in Chapter 4, those adaptations are in line with our new methodology and its guiding principles.

3.5.2 Ciphertext-Only Cryptanalysis of Purple

Angoki B-gata ("Type B Cipher Machine"), codenamed Purple by the United States, was a diplomatic cryptographic machine used by the Japanese Foreign Office just before and during WWII. Purple, like Enigma, is a substitution machine, each input character being replaced by another character, the substitution function changing after each character has been typed. But unlike the rotor-based Enigma, Purple incorporated stepping switches. The machine consists of the following elements:

- A **typewriter**, with 26 Latin letters (Japanese letters were first mapped into Latin letters), and 26 outputs, one per letter.
- An **input plugboard**, that maps the 26 letters from the keyboard, to the "sixes" and the "twenties".
- The **"sixes"**: A set of six stepping switches, which implement a permutation of six input letters. Each stepping switch has 25 positions, each position implementing a different permutation of the six inputs. All the size switches regularly step after each letter is typed.
- The **"twenties"**: Three sets of twenty stepping switches. The order of the three sets can be changed. Each set of "twenties" performs a permutation of 20 input letters. In each set, the stepping switches have 25 positions. At each position, the set implements a different permutation of the 20 inputs. The permutations of the three sets of "twenties" are applied in series, one after the other. The motion of the three sets of stepping switches is governed by the position of the set of "sixes".
- An **output plugboard**, that maps the 26 letters from the "sixes" and the "twenties" to the printing apparatus.
- A **printing** apparatus.

A complete description may be found in [22].

During WWII, a team of US cryptanalysts were able to successfully reconstruct the machine mechanism, and developed statistical cryptanalytic attacks, which often relied on the insecure use of the machine by Japanese operators. For example, the input and output plugboards were always the same, and there were several constraints on the settings that could be used [22].

In 2003, Freeman, Sullivan, and Weierud published a ciphertext-only computerized method for the cryptanalysis of Purple [22]. Their method is based on a divide-and-conquer approach. The algorithm first recovers the settings for the "sixes". It tests all 25 possible starting positions for the "sixes", and for each position, applies hill climbing to recover the plugboard connections of the inputs to the "sixes", using bigrams. After the settings for "sixes", a similar process is applied for the "twenties". All possible orders of the three "twenties" switches are tested, as well as all their $25 \cdot 25 \cdot 25 = 15\,625$ possible starting positions. For each combination, hill climbing is applied, to detect the mapping of the remaining 20 plugboards connections (the input to the "twenties"), using log-trigrams.

This algorithm was successfully applied to several historical messages. Rather than naively applying a generic hill climbing algorithm, it employs a sophisticated divide-and-conquer multistage hill climbing process, as well as an adaptive scoring scheme, along the lines of our new methodology, which we later present in Chapter 4.

3.5.3 Ciphertext-Only Cryptanalysis of Playfair

Playfair is a manual symmetric encryption cipher invented in 1854 by Charles Wheatstone, however its name and popularity came from the endorsement of his friend Lord Playfair. The Playfair cipher encrypts pairs of letters (bigrams), instead of single letters as in the case of other substitution ciphers such as the Caesar cipher. Frequency analysis is still possible on the Playfair cipher, however it would be against 600 possible pairs of letters instead of 26 different possible letters. For this reason the Playfair cipher is more secure than substitution ciphers applied on single letters, and its use continued up until WWII.

Playfair enciphering and deciphering are based on a key table. The key table is a 5×5 grid of letters. Each of the 25 letters must be unique and one letter of the alphabet (usually J) is omitted from the table (as there are only 25 positions in the table, but 26 letters in the alphabet). It is also possible, and often more convenient, to derive a key table from a keyword or sentence. The first characters (going left to right) in the table will be the phrase, with duplicate letters removed. The rest of the table will be filled with the remaining letters of the alphabet, in order. The key table derived from the keyphrase "Hello World" is therefore:

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

To encrypt a message, the message is split into pairs of two letters. If there is an odd number of letters, a Z is added as the last letter. Assuming we want to encrypt the message "hide the gold", after splitting into bigrams, and adding Z at the end, we obtain:

```
HI DE TH EG OL DZ
```

Next, for each pair, we locate its two letters of the plaintext pair in the square. We replace them according to the following rules:

- If the two letters are corners of a rectangle, take the letter on the horizontal opposite corner of the rectangle. For example, `HI` is encrypted as `LF`.
- If both letters are in the same column, select the letter below it in the square (going back to the top if at the bottom). For example, `DE` is encrypted as `GD`.
- If both letters are in the same row, select the letter to the right of each one (going back to the left if at the farthest right). For example, `OL` is encrypted as `WO`.

Using these rules, we obtain:

```
LF GD MW DP WO CV.
```

Historically, Playfair ciphers were solved using manual methods, mostly based on known plaintext or probable words, and depths (several messages encrypted with the same key). Playfair ciphertexts may also be solved using hill climbing. The best performing computerized solution was developed by Michael Cowan [24]. It employs simulated annealing, with multiple restarts.

For scoring, log-quadgrams are used. Although quadgrams are selective, they have low resilience to key errors. But more resilient scoring methods (e.g. bigrams) cannot be used with short ciphertexts (about 100 letters), because of spurious high scores. Log-bigrams or log-trigrams, which have better resilience, could also have been used for longer messages (200 letters or more), but the goal of Cowan's method was to solve short ones (100 letters or less). The simulated annealing algorithm employs a fixed temperature approach. A fixed temperature was found more effective than a diminishing temperature schedule as in the original SA algorithm. At each step, a random set of transformations is applied to the key, rather than testing all possible transformations. The transformations are comprised of:

- Swapping any two elements in the square
- Swapping any two rows of the square
- Swapping any two columns of the square
- Creating a left-to-right mirror copy of the square
- Creating a top-to-bottom mirror copy of the square
- Creating a top-left to bottom-right mirror copy of the square
- Creating a top-right to bottom-left mirror copy of the square

The swap transformations are applied more frequently than the more complex transformations. The mirroring transformations are more disruptive than the swaps, but they are needed for the algorithm to succeed. According to [24], and also based on experiments we made as part of this thesis, Cowan's method is able to recover the key and plaintexts from short ciphertexts with 75-100 letters, whereas hill climbing usually requires at least 200-300 letters. Hill climbing (when successful) also requires 5-10 times the number of decryptions required with simulated annealing.

Cowan's algorithm for Playfair illustrates the successful application of simulated annealing to the cryptanalysis of a classical cipher with challenging settings (short messages).

3.5.4 Other Related Work

In this section, we briefly survey additional related work about the application of local search metaheuristics for the cryptanalysis of classical ciphers, including some comparative studies. In contrast with the related work on Enigma (Section 3.5.1), Purple (Section 3.5.2), and Playfair (Section 3.5.3), the results of none of the studies listed here is the current state of the art in terms of performance. Those studies primarily demonstrate that specific local search metaheuristics may be applied to specific ciphers, rather than demonstrating superior performance with the cryptanalysis of those ciphers.

In [19], Chen and Rosenthal apply a Markov Chain Monte Carlo algorithm to solve substitution, columnar transposition as well as combined substitution-transposition ciphers. For transposition, bigram scores are used. With their method, to recover a key of length $|K| = 20$, $|C| = 500$ to 1000 characters of ciphertext are required.

In [17], Russell, Clark, and Stepney combine the classical method of multiple anagramming with an ant colony optimization algorithm. As for cost functions, they use both bigram scores

as well as dictionary matches. With this method, keys can be fully recovered for $|K| = 25$ and $|C| = 625$.

In [16], Clark compares three types of attacks on columnar transposition ciphers: (1) genetic algorithms, (2) simulated annealing, and (3) hill climbing/tabu search. For scoring, Clark uses a subset of bigrams and trigrams. The two transformations used in his study for all three types of attacks are (a) swapping two random key elements and (b) random rotations of consecutive key segments. The results for the three methods are similar. For a key of length $|K| = 15$, the recovery of at least 14 of the 15 key elements requires a ciphertext with at least $|C| = 450$ to 600 characters. Clark also compares the three methods for the cryptanalysis of simple substitution ciphers. The results are also very similar, and the three methods require at least 400 characters to correctly recover 90% of the plaintext.

In [14], Dimovski and Gligoroski use a genetic algorithm, simulated annealing, and hill climbing/tabu search to solve columnar transposition ciphers with scoring based on bigrams. The transformation used for simulated annealing and tabu search (and for mutations in the genetic algorithm) consists of the swapping of two random key elements. Results are similar for all three metaheuristics. For $|K| = 30$, a ciphertext of at least $|C| = 1000$ ($r = 33$) letters are required to recover 25 of the 30 key elements.

We can make the following observations about those case studies:

- Most methods apply a rather simplistic version of the local search metaheuristics, such as using a single phase and no restarts or limited restarts.
- The transformations applied on the key by the search algorithms are often only the most trivial ones, consisting mainly of swapping two key elements. Furthermore, the set of transformations is not systematically applied, and only a random subset is tested at each search iteration.
- The scoring methods are often straightforward and non-optimal. Rather than using an adaptive approach, the same simplistic scoring functions are applied uniformly to advantageous cases, such as long ciphertexts, and to the more challenging cases, such as short ciphertexts.
- In those studies, there is no evidence that local search metaheuristics other than HC or SA have better performance than them, when applied to the cryptanalysis of classical ciphers.
- In particular, genetic algorithms were found to be no better than hill climbing or simulated annealing for the cryptanalysis of classical ciphers. Experiments performed at an early stage of our research have confirmed this finding.
- With columnar transposition, the algorithms listed here may only succeed with keys no longer than with 15 to 30 elements. For a key with 30 elements, at least 1000 ciphertext letters are needed. For comparison, with the algorithm we describe in Chapter 5 and in [33], very long keys (120 elements in the worst case) can be recovered. For a key with 30 elements, only 180 ciphertext letters are required.
- The algorithms for columnar transposition fail to take into account the more general but also more challenging case of ICT (see Section 5.1.1).
- The performance in all studies about the simple substitution cipher is significantly worse – at least 300 letters required – than with Olson's DeCrypto ([55]), which can solve substitution cryptograms with no more than 50 letters.

There is a clear discrepancy between the sub-par levels of performance observed with the works listed here, compared to state-of-the-art performance with of the attacks on Enigma (Section 3.5.1), Purple (Section 3.5.2) and Playfair (Section 3.5.3). Moreover, the latter ciphers are usually more secure and cryptanalytically challenging than substitution or transposition ciphers, so that we would have expected the opposite.

This stark discrepancy was one of the main motivations for our research and for the development and formulation of our new methodology, described in Chapter 4. Next, in Chapter 5, we present our new and highly effective attack on the columnar transposition cipher. We start with a baseline algorithm, which represents the majority of the prior work approaches. We then incrementally apply one or more of the methodology principles, each time showing their impact on performance. This side-by-side comparison, not only illustrates the effectiveness of the new methodology, but also helps in understanding the discrepancy observed with successful vs. less successful applications of local search metaheuristics to the cryptanalysis of classical ciphers.

3.5.5 Cryptanalysis of Modern Ciphers using Local Search Metaheuristics

A prerequisite to implementing a cryptanalytic attack based on a local search metaheuristic, is the ability to find a statistical measure which can be applied to a candidate key. This statistical measure forms the basis for the development of effective scoring methods. The design of a strong cipher should thwart efforts by cryptanalysts to find such a statistical measure, by hiding in the ciphertext, all the statistics properties of the plaintext language. Modern ciphers achieve that with a strong element of diffusion, via the use of multiple encryption rounds. A practical effect of a strong diffusion is that a single error in the key, or very few errors, will cause the decrypted text to look like a random sequence of symbols. As a result, all the scoring functions described Section 3.2, such as IC or n-grams, have no resilience to key errors, when applied to modern ciphers. Other scoring functions, e.g. trigrams, may still be selective but without resilience to key errors, local search algorithms cannot use them. Therefore, a partially correct key has little value in the search for the correct key of a modern cipher. In contrast, with classical ciphers, if is often possible to find a selective scoring method which has some resilience to key errors, and also generates a relatively smooth search landscape. Such a smooth search landscape is not feasible for modern ciphers.

Furthermore, multiple rounds and other complex logic elements are easier to implement with modern computing or digital electronics. In contrast, with classical ciphers, the ability to implement diffusion has historically been limited by several factors. With manual ciphers, the tradeoff was mainly between security and ease of use. The more complex the cipher was, possibly more secure and with better diffusion, the more time it took to manually encipher or decipher, and the higher the chance for errors was.

With cipher machines, more complex encryption could be implemented than with pen-and-paper ciphers, but still limited by the technology of the time and the electromechanical nature of those machines. Only limited diffusion could ever be achieved. Some cipher machines implemented diffusion by using the plaintext itself as part of the key (“autokey”), such as the “P5” motor limitation of the Tunny Lorenz SZ42. While achieving increased diffusion and better security, the autokey feature also had a major drawback. A single error could completely disrupt the decryption process. Such errors were common when transmitting over radio channels. As a result, the auto-key feature was rarely used. With modern communication networks and error-correction techniques, the quality of the transmission channel is less of an issue.

As a result, local search metaheuristics as those described in this thesis and in related work are irrelevant for most modern ciphers, with rare exceptions. The most interesting and successful application of a local search metaheuristic to a modern cryptographic problem is an attack based on the solution of the Permuted Perceptron Problem [56], using simulated annealing. The difficulty of solving this problem was the basis for a set of zero-knowledge identification schemes proposed by Pointcheval [57].

4

A New Methodology

In this chapter, we present a new methodology for the cryptanalysis of classical ciphers using local search metaheuristics. We first describe the motivation for the methodology, and present an overview of how it was developed. After that, we provide a detailed description of the primary principles of the methodology.

4.1 Motivation

The modern cryptanalysis of classical ciphers is part of the broader field of the study of the history of cryptography in general, and the study of historical ciphers in particular. The study of historical ciphers may be approached from several perspectives.

The first one is purely historical, focusing on the outline of cryptographic and codebreaking developments, with an emphasis on their impact on historical events, such as the decipherment of the Zimmermann Telegram by the UK in January 1917, which precipitated the entry of the US into WWI.

A second perspective is understanding and documenting the technical details of the historical ciphers methods and cipher machines. Some cipher machines, such as the Enigma, had numerous variations, posing a significant challenge for historians when identifying and documenting those machines. The internal mechanisms of some machines such the German teleprinter encryption devices, as well as later Cold War rotor-based designs, were sometimes highly complex. Furthermore, the details of those machines were often kept confidential for many years, long after their use was discontinued.

The third research perspective, more relevant to our research, is the study of historical codebreaking techniques. Most of the secrets about the codebreaking of Enigma, and of the German teleprinter ciphers, in WWII, have already been published. But several key official documents from WWII are still classified. Other documents have been declassified, but important details have been deleted or redacted. Other documents released by the NSA mention that a certain cipher was successfully cryptanalyzed, but give no detail about how this was achieved. Moreover, some of the historical codebreaking techniques may be very complex, involving advanced statistical analysis, or sophisticated machinery. Because of this combination of complexity and secrecy, the study of historical codebreaking techniques is often a challenging task.

The focus of our research, the study of the cryptanalysis of historical ciphers using modern tools, such as local search metaheuristics, can be of great assistance for a better understanding of historical codebreaking techniques. First, it allows for a more accurate and meaningful assessment of the challenges faced, for example, by WWI or WWII codebreakers. If a cipher is still challenging today, given modern computing and algorithmic tools, it must have been a much greater challenge historically. It might often be difficult for a modern codebreaker to fully understand historical codebreaking techniques, unless he himself also attempts to solve the problem, using either historical techniques, or modern techniques. Often, historical and modern cryptanalysis may rely on the same cryptographic weakness, or similar statistical properties. Finally, the modern cryptanalytic study of historical ciphers may help bridge the "holes" caused by the secrecy maintained on key documents, or by the heavy redaction of declassified material. At a minimum, modern cryptanalysis may help to speculate about how a historical solution was achieved, even though all we may have is a historical document only mentioning a success with a certain cipher machine, but without any details on how this success was achieved.

An additional and significant benefit of modern cryptanalysis of historical ciphers, is that they can help to decipher original ciphertexts, for which the key is not known. The most remarkable case is the decipherment by Weierud and Sullivan in 2005, of hundreds of original WWII German messages encrypted with the Enigma cipher machine [21]. Those messages had significant historical value, as they indicated, for the first time, the fate of a French Resistance leader, as well as the fate of a German general who opposed Hitler. A more recent study, a result of our research (see Chapter 6), is the decipherment of a large collection of German telegrams from WWI, encrypted using the ADFGVX cipher. This collection of encrypted telegrams, now readable, provides historians with a unique insight into events in 1918 which had a major impact on the geo-political and military situation in the Eastern Front, the Balkans and the Caucasus region.

Finally, developing new and more powerful methods may help to solve public crypto challenges, such as the Double Transposition Challenge, considered by Otto Leiberich, the former head of the German "Zentralstelle für das Chiffrierwesen", to be unsolvable [38].

The cryptanalysis of some historical ciphers is still a hard problem today, despite the advent of modern computing. This is due mainly to the complexity of the cipher, combined with a very large key space. As described in Chapter 3, those problems can be mapped into search problems, and therefore they may benefit from the use of optimization techniques, and local search metaheuristics in particular. In related works, the use of HC and SA was shown to be highly effective for modern attacks on Enigma and Purple (using HC), and short Playfair ciphertexts (using SA), and those attacks are today state of the art. In other prior works, the implementation of local search metaheuristics was less effective. Moreover, it was not clear, a priori, why certain case studies were highly successful, but the algorithms developed for other case studies, and which employ similar metaheuristics, did not achieve state-of-the-art-performance.

The primary goal of our new methodology is to provide a set of principles and guidelines, to assist in the design of effective attacks on classical ciphers, based on local search metaheuristics.

4.2 Overview of the Methodology Principles

As part of the research for this thesis, we analyzed the prior work case studies, to identify the factors that characterize the successful applications of local search metaheuristics, and those factors which characterize the less successful ones. Some initial patterns emerged, and we

applied the insights we learned to the columnar transposition cipher (Chapter 5) and the double transposition cipher (Chapter 9). We further refined the methodology, and applied it for the cryptanalysis of other challenging ciphers, such as ADFGVX (Chapter 6), the Hagelin M-209 cipher machine (Chapter 7), Enigma (Chapter 10), and Chaocipher (Chapter 8). The process was iterative, and the performance of one attack could be improved as a result of refinements and insights gained while working on other attacks. Eventually, through analytic and empirical research, we came up with the formulation of five major guiding principles for the effective use of local search metaheuristics for the cryptanalysis of a classical cipher. For a cryptanalytic attack on a classical cipher to be effective, not all five principles need to be applied, but for the more challenging classical cryptanalysis problems, at least some of them are useful or even necessary. The five guiding principles are summarized here:

GP1: Hill climbing or simulated annealing

GP2: Reduction of the search space

GP3: Adaptive scoring

GP4: High-coverage transformations preserving a smooth search landscape

GP5: Multiple restarts with optimal initial keys

Those principles are described in further detail in the following sections. For each principle and guideline, we include one or more references where they have been successfully applied.

4.3 GP1: Hill Climbing or Simulated Annealing

Prior work on Enigma and Purple, as well as case studies as part of this research (columnar transposition, double transposition, ADGVX, Hagelin M-209, and Chaocipher), have clearly established HC as a powerful and highly effective metaheuristic. A major advantage of HC is its simplicity. When developing new attacks, it will often be the first choice.

There are cases, where SA may be effective (and sometimes more powerful than HC), as demonstrated by prior work on short Playfair ciphertexts (see Section 3.5.3). The main drawback of SA is that it requires a significant amount of experimentation to tune its parameters, such as the starting temperature and the cooling schedule.

Other metaheuristics such as genetic algorithms and ant colony optimization are interesting primarily because they mimic some natural phenomena. However, while they may work for less challenging cryptanalytic problems, there is no evidence they perform better than HC or SA, particularly for the more challenging classical ciphers.

Most of our case studies employ HC. We use SA for a ciphertext-only attack on Hagelin M-209 (see Section 7.5). HC and SA may also be combined in the same attack, as described later in the section.

The most straightforward HC-based approach is to implement a single process, for example a single HC process along the lines of the algorithm described in Algorithm 2, with multiple shotgun restarts. A more complex scheme may often be more appropriate, which involves more than one HC or SA processes. The most relevant schemes are listed below, and we describe those with more than one search process in the following sections:

1. One **single** search process, for the whole key.
2. Several **parallel** search processes, each process searching for different parts of the key.
3. **Nested** processes searching for different parts of the key, the inner process being invoked at each step of the outer process.
4. Several **sequential** search processes, for different parts of the key, performed one after the other.
5. Several **sequential** search processes, for the whole key, performed one after the other.

Several schemes from the list above may also be further combined to form an even more complex scheme.

4.3.1 Parallel Search Processes

A parallel search scheme is mostly relevant to HC processes, each applied on a different part of the key K. We illustrate the scheme with an example of two processes. HC_1 is a hill climbing process designed to improve the first part of the key, KP_1, and a second process, HC_2, is designed to improve the second part of the key, KP_2. The parallel search algorithm is described in Algorithm 4.

We start with some initial (e.g. random) KP_1 and KP_2. We first apply HC_1 to improve the initial KP_1, until KP_1 cannot be further improved. Next, we similarly apply HC_2 to improve KP_2, given the possibly improved KP_1, until KP_2 cannot be further improved. We continue to alternate between of HC_1 and HC_2 until neither KP_1 nor KP_2 can be further improved. While both HC_1 and HC_2 modify each only one part of the key, the scoring function they use is the same as the function S used by the main process.

Algorithm 4 Parallel hill climbing algorithm

```
1:  procedure ParallelHillClimbing(C)                          ▷ C = ciphertext
2:      BestKP1 ← ComputeInitialKP1()
3:      BestKP2 ← ComputeInitialKP2()
4:      BestScore ← S(Decrypt(BestKP1, BestKP2, C))
5:      repeat
6:          Stuck ← true
7:          NewKP1 ← HC1(BestKP1, BestKP2)          ▷ Improve first part given BestKP2
8:          NewKP2 ← HC2(NewKP1, BestKP2)           ▷ Improve second part given NewKP1
9:          NewScore ← S(Decrypt(NewKP1, NewKP2, C))
10:         if NewScore > BestScore then
11:             BestScore ← NewScore
12:             BestKP1 ← NewKP1
13:             BestKP2 ← NewKP2
14:             Stuck ← false
15:     until Stuck = true
16:     return BestKP1, BestKP2
```

Actually, HC_1 and HC_2 are not truly parallel processes, but rather they are activated in alternation, each process trying to improve one part of the key, and using the outcome of the other process (the other part of the key), as part of its input, but without modifying it.

A scheme of two parallel (alternating) HC processes was implemented for the known-plaintext cryptanalysis of Hagelin M-209 (see Algorithm 7.4.2.1), the first one improving the lug settings, and the second one improving the pin settings.

4.3.2 Nested Search Processes

This scheme is relevant for both HC and SA processes. With this scheme, an **outer** process searches for some part of the key, and at each step, activates an **inner** search process, using the output of the second process to produce a score used to decide whether or not to accept new key settings. We illustrate the scheme with two processes, an outer hill climbing process HC_{outer} which searches for the first part KP_1, and an inner simulated annealing process SA_{inner} which searches for the second part, KP_2, given KP_1.

The outer process HC_{outer} is described in Algorithm 5. It is usually invoked multiple times, with multiple restarts. Each cycle (or restart) starts with an initial KP_1 (e.g. random), and tries to improve this KP_1 using transformations. For each candidate $CandidateKP_1$, it first invokes a nested inner SA_{inner} process, to find the best KP_2 given $CandidateKP_1$ (SA_{inner} does not modify $CandidateKP_1$). Given the $CandidateKP_1$ obtained from SA_{inner} and the current $CandidateKP_1$, HC_{outer} computes a score, and uses that score to decide whether or not to accept the candidate $CandidateKP_1$.

Algorithm 5 Hillclimbing with nested simulated annealing

```
1:  procedure HC_outer(C)                                   ▷ C = ciphertext
2:      BestKP_1 ← ComputeInitialKP1()
3:      BestKP_2 ← ComputeInitialKP2()
4:      BestScore ← S(Decrypt(BestKP_1, BestKP_2, C))
5:      repeat
6:          Stuck ← true
7:          for CandidateKP_1 ∈ Neighbors(BestKP_1) do      ▷ Neighbors of first key part
8:              CandidateKP_2 ← SA_inner(CandidateKP_1) ▷ Find best KP_2 given CandidateKP_1
9:              NewScore ← S(Decrypt(CandidateKP_1, CandidateKP_2, C))
10:             if NewScore > BestScore then
11:                 BestScore ← NewScore
12:                 BestKP_1 ← CandidateKP_1
13:                 BestKP_2 ← CandidateKP_2
14:                 Stuck ← false
15:                 break
16:     until Stuck = true
17:     return BestKP_1, BestKP_2
```

A scheme of nested HC (outer) and SA (inner) processes was implemented for the ciphertext-only cryptanalysis of Hagelin M-209 (see Section 7.5): the outer HC searching for the lug settings, and an inner SA for the pin settings.

This nested scheme resembles Iterated Local Search (ILS) [58]. ILS uses two types of stochastic local search steps: one for reaching local optima as efficiently as possible, and the other for effectively escaping from local optima. The two types are applied in alternation to perform a walk in the space of local optima w.r.t. a given evaluation function. The main ("outer") process can be initialized in various ways, e.g. by starting from a randomly selected element of the search space. From a candidate solution, a locally optimal solution is obtained by applying a subsidiary

("inner") local search procedure. ILS uses the result of that subsidiary local search to determine whether or not to accept the candidate solution. The main and subsidiary components need to complement each other for achieving a good tradeoff between intensification and diversification of the search process, which is critical for obtaining good performance (see Section 2.3.4). ILS was found to be highly effective for several hard combinatorial problems.

The main difference between ILS and our nested scheme described in this section, is that with ILS both the main (outer) and subsidiary (inner) local search processes may affect **any** part of the key, while in our nested algorithm, the outer and inner processes affect **different** parts of the key. Additional research is needed to evaluate the potential of the original ILS algorithm, for the cryptanalysis of classical ciphers, also comparing it with our nested approach.

4.3.3 Sequential Search Processes – Different Key Parts

With this scheme, two HC or SA processes are applied one after the other, to recover the two parts of the key, KP_1 and KP_2. For example, we may have a simulated annealing process SA_1 to recover KP_1, followed by a hill climbing process HC_2 to recover KP_2, given the value of KP_1 obtained by SA_1 (but not changing it).

This sequential scheme is typically used for a divide-and-conquer attack (see Section 4.4), when it is not possible to brute-force any of the two key parts. This scheme requires a scoring method that can be applied to KP_1 alone, for the first process. A sequential approach with divide-and-conquer was used for the cryptanalysis of Purple, described in the previous chapter (see Section 3.5.2), for the double transposition (see Section 9.4.5.1) and for ADFGVX (see Section 6.4).

4.3.4 Sequential Search Processes – Applied on the Whole Key

With this scheme, two processes are applied one after the other, both trying to recover the whole key K. The output of each process serves as the input for the next one, which further tries to improve it. This scheme is usually composed of hill climbing processes, except for the first one which may also be a simulated annealing process. For example, a simulated annealing process SA_1 can be applied first, using a highly resilient scoring function, such as IC, to recovery some initial correct key elements. The resulting K is then passed to the next process, hill-climbing process HC_2, which uses a more selective scoring function, such as quadgrams, to recover the full and correct key.

This scheme is often employed to generate optimal initial keys for hill climbing, rather than using a random initial key. The first process generates one or more keys, using various techniques (see Section 4.7), and selects the best one for as the initial key for the main hill climbing process. This sequential scheme was used for the cryptanalysis of the columnar transposition cipher, described in Section 5.3.3.

Note that a sequential scheme may also be applied to some part of the key, with two search processes applied to that same part of the key. In the cryptanalysis of the double transposition cipher (see Section 9.4.5.1), two separate HC processes are applied one after the other to recover the second transposition key. Using a divide-and-conquer scheme (see Section 4.3.3), a third process uses this second transposition key in turn to recover the first transposition key. In total, the cryptanalysis of the double transposition has three search processes, combining two sequential schemes.

4.3.5 Summary of GP1

Hill climbing and simulated annealing are the local search metaheuristics of choice for effective cryptanalysis of classical ciphers. They may be applied as a single search process, or as multiple processes – parallel, nested or sequential.

4.4 GP2: Reduction of the Search Space

In most problems of classical cipher cryptanalysis, the size of the keyspace not only prohibits the use of brute-force search techniques, but also poses a major challenge for local search algorithms, including HC and SA. To mitigate the problem, methods for reducing the size of the search keyspace may be employed for some ciphers, and they are described here and illustrated in the case studies.

One way to reduce the search keyspace, is to try and identify cipher settings which are cryptographically equivalent. If they exist, redundant settings should be ignored, and only those that are unique from the cryptological point of view should be considered in a search. For example, the lug settings of the Hagelin M-209 machine are highly redundant, with a significant proportion of equivalent settings. As described in Section 7.2.4.2, it is possible to restrict the search to 2^{43} cryptographically unique and non-redundant lug settings, instead of 2^{118}.

The primary method to reduce the size of the search space is, however, to apply a divide-and-conquer approach. When using such an approach, the goal is to first recover certain elements of the key settings, usually ignoring the remaining elements. After one part of the key has been recovered, it is possible to recover the remaining parts more effectively using a search algorithm, as the search space is now much smaller than the combined keyspace of all key elements.

Alternatively, a divide-and-conquer method may be based on a systematic survey of all possible values for some part of the key, while employing a local search or another algorithm (e.g. combinatorial) to recover the remaining parts of the key. We denote such an approach as partial brute-force divide-and-conquer, as brute force is applied only to some part of the key.

In some cases, it might not be possible to divide the key into parts, so that one part may be "brute-forced" in a computationally practical manner. An alternative approach could be to survey and iterate over only a limited subset of the search space of a certain key part. This subset should be well distributed in the space of that key part, so that any possible value of the key part (in the superset) should be "close enough" to at least one member of the subset (e.g. they should share a minimum number of elements, or their Hamming distance limited). This approach was used for ciphertext-only cryptanalysis of the Hagelin M-209 (see Section 7.3.2.5). The key settings of the Hagelin M-209 machine consist of lug settings and pin settings. The spaces of both of the lug settings or the pin settings are individually too large for an exhaustive survey. Instead, a limited subset of selected "representative" lug settings is surveyed, and this subset is designed so that it is well distributed over the complete (superset) lug setting space [36].

The best known historical example of a successful divide-and-conquer approach is the Turing Bombe method, used in WWII for known-plaintext cryptanalysis of Enigma. The problem is divided into two parts, the recovery of the rotor settings (and of some plugboard settings), followed by the recovery of the full plugboard settings. The Turing Bombe is an electromechanical device which implements an exhaustive search over a large number of possible rotor settings,

and for each one, applies a logic (using an electrical circuit) to rule out non-matching rotor settings. When matching rotor settings are found, the Bombe also produces some of the plugboard settings, the remaining being recovered using a manual process. Other examples of successful divide-and-conquer approaches include the modern attacks on Enigma and Purple, described in Section 3.5.1 and in Section 3.5.2. Other cases studies in the following chapters (double transposition, ADGVX, Hagelin M-209, and Chaocipher) all illustrate the successful use of various space reduction techniques, mainly divide-and-conquer.

Summary of GP2:

- **Ignoring equivalent settings**, if redundancy exists in key settings.
- **Divide-and-conquer** approach, if feasible. Either using **partial brute force**, or sequential search processes.
- Reducing the search space using a **representative set**, if applicable.

4.5 GP3: Adaptive Scoring

The scoring method is often the most critical factor for successful HC or SA. In the previous chapter (see Section 3.2.3), we introduced two primary attributes, for a scoring method to be effective, namely **selectivity**, and **resilience to key errors**. Those goals are often contradictory. At the extremes, IC often has the best resilience to key errors, but low-to-moderate selectivity, and particularly low selectivity for short ciphertexts (due to spurious high scores). Conversely, high n-grams (e.g. quadgrams) have very good selectivity, but less resilience to key errors. It is often required to perform extensive tests and simulations, under various key settings and key error scenarios, to assess fitness-distance correlation (see Section 2.3.10) in general, and selectivity and resilience in particular, for a candidate scoring method. Such an analysis may be found in Section 7.4.2.4 for the ADE score (used in known-plaintext attack for the Hagelin M-209), and in Section 9.4.5.3 for the IDP score (ciphertext attack on the double transposition).

The scoring method must be carefully adapted to the type of cipher and cipher problem. Furthermore, an effective attack may require the use of several scoring functions, used at different stages of the attack. Most often, the most challenging part of an attack is the recovery of the first correct key elements, especially for HC starting with random initial keys. Random keys, given a large keyspace, almost always contain a very large number of errors. For the initial phase of the search, a scoring method with better resilience to key errors is preferred, as it is more likely to detect improvements in such a key with many errors. When some of the correct key elements have been recovered, a more selective scoring function (such as higher-order n-grams) is more appropriate, so that the search may converge quicker and more accurately towards the correct key. This adaptive approach shares some elements with Dynamic Local Search approaches (see Section 2.3.3)

If the ciphertexts are short, however, only scoring functions with good selectivity may be effective, because of spurious high scores (see Section 3.2.3).

In Section 3.2.3, we presented a number of generic default scoring functions, n-grams and IC, which are based on the analysis of a decrypted plaintext. It is, however, often necessary to develop a specialized scoring function, in case a good tradeoff between selectivity and resilience cannot be achieved with the standard scoring functions. The *ADE*, used for the known-plaintext

analysis of Hagelin M-209 (see Section 7.4.2.4) is a good example of such a specialized, highly effective scoring function.

In some cases, a specialized scoring function may be designed so that it applies only to some parts of the key, to serve as the basis for a powerful divide-and-conquer attack. Such divide-and-conquer specialized scoring functions were successfully used for the cryptanalysis of ADFGVX (see Section 6.4), the double transposition cipher (the *IDP* – Section 9.4.5.2), Chaocipher (see Section 8.4.2), and Enigma with double indicators (Chapter 10).

The challenge of recovering the first correct elements of the key may also be addressed using optimized restarts (see Section 4.7). To obtain good quality initial keys for restart rounds, a search process (HC or SA) using a scoring function with good resilience may be employed, as illustrated in several of the cases studies in this thesis. A good initial key is a key of better quality than a random key, that is, a key that is likely to have fewer errors and a better score than a random initial key.

Summary of GP3:

- **Assessing the selectivity and resilience** of candidate scoring methods.
- **Adaptive scoring** – start with better resilience, then better selectivity.
- **Specialized** scoring when default methods are not effective.
- Specialized scoring for **divide-and-conquer** attacks, if applicable.

4.6 GP4: High-Coverage Transformations Preserving a Smooth Landscape

As described in Section 2.3.5, a large neighboring size helps to achieve diversification, as more types of transformations provide local search with more opportunities to progress. But this comes at a cost, in two main aspects: computational cost, and a potentially negative impact on the search landscape smoothness. The first one is obvious, the more transformations to check, the more computing is needed at each step of local search. The second effect is more subtle.

As described in Section 2.3.9, a smooth landscape is beneficial to the performance of iterative/increment improvement search algorithms, such as stochastic local search. The search landscape for a certain problem instance depends on two factors, the evaluation (scoring) function, and the neighboring function. The neighboring function depends on the type of transformations that may be applied to a key, to obtain its neighbors. Similarly, the number of neighbors depends on the number of possible transformations. The more transformations, the higher the probability that they include *disruptive transformations*. Disruptive transformations are transformations that generate neighbors with poor correlation (see Section 2.3.9), i.e. moving to this neighbor results in a "big jump" in the score – up or down. Therefore, a larger neighborhood usually results in a more rugged landscape.

To achieve a proper balance between the conflicting goals of achieving a large neighborhood and of preserving a smooth landscape, several strategies have been found effective, for cryptanalytic problems with classical ciphers. First, as a general rule, as many as possible non-disruptive types transformations should be implemented. Those are often simple transformations, such as modifying a single key element, or exchanging two key elements ("swaps of two").

To achieve more diversity, additional types of possibly more complex transformations, which increase the size of the neighborhood, may be required. If additional and more complex transformations are either disruptive, or there are many of them to compute at each step, or the cost of evaluating each candidate neighbor is high (for example, in a nested scheme – see Section 4.3.2), the more complex types of transformations should be used sparingly and only when needed. That is, only after a local maximum has been reached, which cannot be escaped using the simpler types of transformations. After the search has found a (complex) transformation which allows the search to escape the local maximum, it returns to using only the simpler ones. This approach is a Variable Neighborhood Search approach (see Section 2.3.5).

We illustrate a variable choice of transformations with the cryptanalysis of a columnar transposition cipher. In this problem instance, the transposition key has 20 elements (essentially, this key is a permutation of the numbers 1 to 20). We employ HC to recover the key. The simplest type of transformation consists of swapping two elements in the key. There are $\frac{20 \cdot 19}{2} = 190$ possible swaps of two elements, exhaustively tested at each step of the search. To enlarge the neighborhood and increase diversification, we may also want to test more complex transformation, e.g. swaps of three elements.

There are $\frac{20 \cdot 19 \cdot 18}{3!}$ possibilities to select three elements (unordered) from twenty. While there are 3! possibilities to reorder those three selected elements, only two of those reordering options result in all the three elements being removed from their original place. There are therefore $\frac{20 \cdot 19 \cdot 18 \cdot 2}{3!} = 2280$ relevant options for swaps of three elements. While we could always test all $190 + 2280$ transformations at each step of HC, we do not need to do so. Instead, we can try first to progress with simple swaps (190 options), and only when no more improvement can be achieved with swaps of two, we start testing swaps of three (2280 options) until we find one that produces an improvement, after which we can go back testing swaps of two elements.

Swaps of two or three elements might not be enough for HC to converge, for the more challenging cases of incomplete transposition rectangles (see Section 5.1.1) or for short ciphertexts. While we could consider swaps of more elements, e.g. four or five, this would require each step of the search to process a significantly higher number of transformations, and some of them could be too disruptive. For the cryptanalysis of columnar transposition cipher, we instead implemented new types of transformations, *segment transformations*, that are applied on consecutive elements (segments) of the key. With columnar transposition, a swap or rotate transformation applied on n consecutive elements is less disruptive than changing n randomly chosen elements. On the other hand, those segment transformations create additional and often critical opportunities for HC to progress and converge. An analysis of the impact of using segment transformations, which better preserve the landscape smoothness, showing them to be significant, appears in Figure 5.4.

Many of the HC or SA algorithms used in prior works (see Section 3.5.4), apply a set of transformations *randomly*, rather than *systematically*. As a result, only some of the candidate neighbors are surveyed, and the search loses opportunities to progress. In our case studies, we found that the most effective approach consists of applying transformations systematically at each step of the search (possibly, with a Variable Neighborhood Search approach), rather than randomly. In addition, it is important to test the possible transformations in *random order* (see Section 2.3.7). Furthermore, and this applies mainly to hill climbing, a First Improvement approach shall be employed, rather than Best Improvement (see Section 2.3.6).

Summary of GP4:

- **High-coverage** set of transformations.

- **Systematically** applied (rather than randomly selected) but in random order.
- **Preserve the smoothness** of the landscape.
- **Variable Neighborhood Search** approach to increase diversification.
- (for hill climbing) **First Improvement** approach rather than Best Improvement.

4.7 GP5: Multiple Restarts with Optimal Initial Keys

HC and SA are statistical search methods, which succeed only with a certain probability. The probability of success depends on many factors, such as the length of the ciphertext, the complexity of the key/settings, and the design of the specific method, including the scoring function and the transformations used. Obviously, running an attack multiple times increases the overall probability of success. While some of the prior work used only a single run of HC or SA, or a relatively small number of restarts, in our research we always allow the attack to be repeated until it is successful or until a certain time limit has expired.

An alternative approach is to run multiple and independent instances of the HC or SA search, in parallel. Those may execute on the same machine with multithreading or multitasking, or on several platforms using distributed computing. A key search algorithm employing multiple restarts is essentially an Embarrassingly Parallel search problem, as those multiple restarts are fully independent.

A less straightforward aspect of multiple restarts is the choice of the initial key. The simplest approach is to use a random key. The random keys generated for that purpose should be uniformly distributed over the full keyspace, to provide for diversity. For SA, a simple random initial key is usually sufficient, as SA relies on the acceptance of some downward moves, to allow the search to explore diverse areas of the keyspace.

For HC especially when applied to a challenging cryptanalysis search problem, a random initial key may not be enough to allow the search to "take off" and converge. The main challenge of HC is often to recover the first correct key elements, in addition to those which may be correct by chance in a randomly-generated initial key. For that purpose, HC needs an initial key better than just a random key.

One simple way to increase the quality of the initial key, is to generate a large number of random keys, to score them, and to select the key with the highest score, as the initial key for HC. As discussed in Section 4.5, a scoring method with good resilience should be used to select the best of several random keys. This simple approach has been successfully used for the ciphertext-only cryptanalysis of Hagelin M-209 (see Section 7.5), and for ADFGVX (see Algorithm 6).

A more sophisticated and powerful approach consists of adding a separate preliminary search phase, using HC, SA, or other metaheuristics. This might be a modified version of a main HC with some limitations on the number of restarts or on the types of transformations, or another greedy algorithm. Most often, this will require a scoring method different from the one used in the main HC process, and usually a more resilient, or even a specialized scoring function. For example, the specialized *Adjacency Score* and the *Alignment Score* were used in the preliminary phase of the cryptanalysis of the columnar transposition cipher (see Chapter 5.3.4). This approach bears some resemblance to Greedy Randomized Adaptive Search Procedures (GRASP) algorithms [59]. GRASP algorithms compute high-quality initial candidate solutions (for the

main search algorithm) using a greedy constructive search, possibly some limitations preventing it from selecting optimal components (w.r.t. the evaluation function) [60]. Empirical results indicate that the additional local search phase improves the performance of the algorithm considerably for several combinatorial problems [59].

The preliminary phase may also be applied several times, and the best key selected as the initial key for the main HC process. In Section 9.4.6, we describe how we implemented a reduced version of HC ("left-to-right optimization"), to generate high-quality initial keys for the main HC process, for the double transposition cipher.

Finally, as the method to obtain higher quality and optimal initial keys gets more sophisticated, there might be a tradeoff between the quality of initial keys and the computation effort needed to produce them. A longer preliminary phase increases the overall time needed for a complete round (the preliminary HC and the main HC). In some of our case studies, we simply limited the amount of time allocated to the preliminary phase.

Summary of GP5:

- **Multiple restarts**, for both HC and SA.
- **Parallel processing** of independent instances, if computational resources are available.
- Where applicable and needed, a **preliminary phase** to produce high quality optimized initial keys, possibly using a greedy algorithm and specialized scoring.

4.8 Conclusion

In this chapter, we presented a new methodology for effective cryptanalysis of classical ciphers using local search metaheuristics, and its guiding principles. This new methodology, developed as a result of empirical research and the analysis of successful and less successful prior works, has been applied and validated with a series of case studies, described in detail in the following chapters. As a result, new state-of-the-art techniques have been developed for a number of challenging classical cryptography problems, solutions have been found to public cryptographic challenges, and some of the methods also enabled the decryption of historical documents, such as a collection of original WWI ADFGVX messages.

In the next chapters, we review the cases studies, in which we applied the principles of the methodology.

5

Case Study – The Columnar Transposition Cipher

In cryptography, a transposition cipher is a cipher in which the order of the letters is modified, rather than replacing the letters with other symbols as in substitution ciphers. Historically, transposition ciphers took many forms, such as Route ciphers, Rail Fence ciphers, or Grilles ciphers [1]. However, the most popular transposition cipher was the Columnar Transposition cipher, due to its simplicity [2]. For example, the Irish Republican Army (IRA) used it as its main cipher during the 1920s [61]. Columnar transposition was also used as a building block in composite ciphers, such as the ADFGVX cipher, or the Double Transposition cipher. Historically, the Columnar Transposition cipher was solved using manual pen-and-paper methods [5]. More recently, modern computerized approaches, based on local search techniques, have been applied for its cryptanalysis [62] [63] [16]. The scope and the performance of those methods, however, are rather limited.

In this chapter, we describe how we applied the methodology presented in Chapter 4, to implement a new ciphertext-only attack on the columnar transposition cipher. This attack is designed to be effective in challenging settings, such as the use of long keys, or of incomplete transposition rectangles. The attack is based on a two-phase hill climbing algorithm, specialized scoring methods, and a rich set of non-disruptive key transformations, mostly applied on key segments. This new ciphertext-only method allows for the recovery of transposition keys with up to 1 000 elements, and up to 120 elements for worst cases of incomplete columnar transposition rectangles

This chapter is structured as follows: In Section 5.1 we describe the columnar transposition cipher and analyze its keyspace size. In Section 5.2, we provide an overview of the classical manual cryptanalysis methods, as well as modern computerized methods. In Section 5.3, we incrementally present the algorithm, starting with a baseline, and each time focusing on a specific improvement. Each such improvement implements one or more of the thesis methodology guidelines from Chapter 4. In Section 5.4, we summarize the results.

The results presented in this chapter have also been published in Cryptologia [33].

[0]The evaluation is based by applying the method to ciphertexts generated from English plaintexts.

5.1 Description of the Columnar Transposition Cipher

In this section, we present the working principle of the columnar transposition cipher, the notation we use throughout this chapter, and an analysis of the size of the cipher keyspace.

5.1.1 Working Principle of the Columnar Transposition Cipher

The working principle of the columnar transposition cipher is simple. An example of encryption is illustrated in Figure 5.1. First, a transposition key must be selected, and must be known by both the transmitting side who needs to encrypt the message, and the receiving side who needs to decrypt it. The transposition key consists of a series of numbers, specifying how the columns of the plaintext should be transposed or permuted. This key may be derived from a keyword, usually for short keys, or for longer keys, from key phrases, as those are easier to memorize than numerical keys. In case a keyword (or key phrase) is used, the equivalent numerical key is extracted by assigning each letter of the keyword a numerical value which reflects the relative position of the letter in the "A..Z" alphabet. In our example, the keyword is "KEYWORD". D is the first of the keyword letters to appear in the alphabet, so it is assigned a numerical value of 1. E is the next letter and it is assigned the numerical value 2, and so on, until we obtain the full numerical key (3,2,7,6,4,5,1). In case a letter appears more than once in a keyword, successive numerical values are used. For example, the numerical key for the keyword "S**E**CR**E**T" would be (5,**2**,1,4,**3**,6), with the successive values 2 and 3 used to represent the letter E which appears twice.

To encrypt a plaintext, we first copy the plaintext, line by line, into a rectangle. The width of the rectangle is equal to the length of the key. On top of the rectangle, we inscribe the keyword, and on top of the keyword, we inscribe the equivalent numerical key. This is illustrated in part (1) of Figure 5.1. Note that the last row of the rectangle is incomplete, and therefore the first 3 columns of the transposition rectangle, before transposition, are longer (by one row) than the other 4 columns. This case is referred as an incomplete transposition rectangle or **Irregular Columnar Transposition (ICT)**. The case where all columns are of the same length, and all rows are complete, is referred to as **Complete Columnar Transposition (CCT)**.

Next, we transpose or reposition the columns according to the transposition key to form the ciphertext rectangle, as shown in part (2) of Figure 5.1. Plaintext column 1 is copied to column 3 in the ciphertext rectangle, plaintext column 2 to column 2, plaintext column 3 to column 7, and so on. The resulting ciphertext rectangle also has 3 columns longer than the others, but those are not necessarily the first columns on the left, as with the plaintext rectangle.

Finally, after transposing the columns, we extract the text column by column from the ciphertext rectangle, to obtain the final ciphertext, as shown in part (3) of Figure 5.1.

The decryption process is similar, but those steps are performed in reverse order. First, the ciphertext is copied into a rectangle, column by column, as shown in part (2) of Figure 5.1. Special care is required for the case of an incomplete transposition rectangles, as we first need to determine, according to the key, which columns are long and which are short. In our example, the ciphertext columns 2, 3 and 7 are long columns, as they correspond to the first 3 plaintext columns, 2, 1 and 3 respectively. After filling the ciphertext rectangle, taking into account the length of the columns, we reposition the columns by applying the inverse transposition key: ciphertext column 1 is copied back to column 7 in the plaintext, ciphertext column 2 back to

column 2, ciphertext column 3 back to column 1, and so on. Finally, we read the text from the rectangle row by row to obtain the decrypted plaintext.

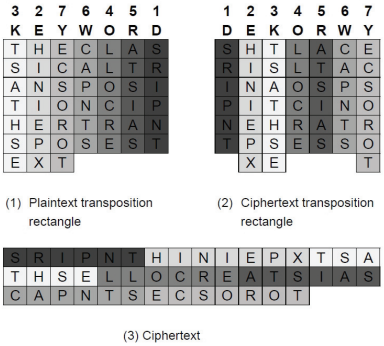


FIGURE 5.1: The columnar transposition cipher

5.1.2 Notation

In this section, we introduce the formal notation we use throughout this chapter. We denote as K the transposition key in its numerical form, e.g. (3,2,7,6,4,5,1), and its length as $|K|$. We denote the plaintext as P and its length as $|P|$. The ciphertext after encryption is denoted as C, and its length as $|C|$. Note that the length of the ciphertext is always equal to the length of the plaintext, i.e. $|C| = |P|$.

We define the number of full or complete rows in the transposition rectangle, as r. Since the width of the transposition rectangle is equal to the key length, $|K|$, we compute the number of full rows as follows: $r = \lfloor \frac{|C|}{|K|} \rfloor$. With ICT, the last row is not complete, and some of the columns are longer, by one row, than the others columns. We define the number of long columns as u, which we compute as follows: $u = |C| \bmod |K|$. Based on our definition of ICT and CCT it follows that $u = 0$ for CCT, and that $u > 0$ for ICT. We present a summary of all notations in Table 5.1.

Name	Symbol	Notes
Transposition key	K	
Length of transposition key	$\|K\|$	
Ciphertext	C	
Plaintext	P	
Length of ciphertext/plaintext	$\|C\|$	$\|C\| = \|P\|$
Number of full rows	r	$r = \lfloor \frac{\|C\|}{\|K\|} \rfloor$
Number of long columns	u	$u = \|C\| \bmod \|K\|$

TABLE 5.1: Columnar transposition cipher – notation

5.1.3 Size of the Keyspace

For a key of length $|K|$, the number of possible and unique keys is $|K|!$. In Table 5.2 we listed several key lengths, up to 1 000, and the corresponding size of the keyspace. It can be seen that for keys with up to 15 elements, a brute-force search for the key is feasible, while for keys with 25 elements or more, brute-force search is not an option.

Key length ($|K|$)	**Size**	**Size**
15	$1.3 \cdot 10^{12}$	2^{41}
25	$1.6 \cdot 10^{25}$	2^{84}
50	$3.0 \cdot 10^{64}$	2^{215}
100	$9.3 \cdot 10^{157}$	2^{525}
250	$3.2 \cdot 10^{492}$	2^{1637}
500	$1.2 \cdot 10^{1134}$	2^{3768}
1000	$4.0 \cdot 10^{2567}$	2^{8530}

TABLE 5.2: Columnar transposition cipher – size of keyspace

5.2 Related Work – Prior Cryptanalysis

In this section we present the prior methods for the cryptanalysis of the columnar transposition cipher. First, we describe the classical and manual methods, used during the first half of the 20$^{\text{th}}$ century. Then, we present modern computerized methods, based on local search techniques, as well as their limitations.

5.2.1 Historical Cryptanalysis

Historically, several manual methods were developed to cryptanalyze columnar transposition ciphers, including for the CCT and the ICT cases. The best-known example is the "strips method" described by Friedman in [5], which can be applied to CCT cases with short keys. At first, the cryptanalyst arranges the ciphertext in columns, on paper. He then cuts the text into strips, each column into one strip. Next, he manually tries to match the strips against each other using those arrangements which create the most probable bigrams, or pairs of successive letters. There are some bigrams which are easy to recognize, such as Q always followed by U. After that he extends the process to the reconstruction of trigrams, quadgrams, etc. The cryptanalyst repeats this process until the full key has been recovered. For ICT, the analyst may use "hat diagrams" to apply this process to all possible starting and end positions of the columns [5]. Those methods tend to be cumbersome for keys longer than 20, and in particular for the case ICT.

Another example of a manual method is "multiple anagramming" [64]. This method can be applied to the special case of two plaintexts having the same length and encrypted with the same key. The permutation of letters as a result of transposition is identical for both plaintexts. Therefore, any rearrangement (anagramming) of some of the ciphertext letters which produces a valid plaintext when applied to the first ciphertext, would also produce a valid plaintext when applied to the second ciphertext. A good description of multiple anagramming can be found in [65], page 467.

There are other solutions for the special case of two messages encrypted using the same key, and for which the plaintexts have either the same beginning or the same ending. Those methods are applicable if the identical portion of the text is longer than several times the length of the key. We first copy each ciphertext into a transposition rectangle. Next, we look at sequences of letters which appear in one of the columns of the first ciphertext rectangle, and also in one of the columns of the second ciphertext rectangle. By analyzing the relationships between those columns that share common sequences, as well as the exact positions of those matching sequences, it is possible to recover the original transposition key. The solution is described in detail in [5], Section V, paragraphs 23 and 24.

Finally there are solutions for the special cases of a known beginning or ending. For example, if the beginning of the plaintext is known (and that known portion of the text is longer than the key length), it is possible to identify the letters which will appear at the beginning of the columns. By matching those letters to the letters of the plaintext, it is possible to recover the transposition key. A similar method can be used if the plaintext ending is known. Those solutions are described in detail in [5], Section V, paragraph 20.

5.2.2 Modern Cryptanalysis

Starting from the 1990s, computerized solutions based on local search techniques have been proposed for the cryptanalysis of the Columnar Transposition cipher.

One of the earliest works using a genetic algorithm for columnar transposition ciphers was by Matthews in 1993 [62]. In addition to the case of CCT, his solution also covers the more complex ICT case, but only for relatively short keys. For scoring, Matthews uses a small subset of the most common bigrams and trigrams and assigns a weight to each one of them. With a ciphertext of length $|C| = 184$, Matthews's method is able to recover a key of length $|K| = 9$ which corresponds to an ICT case with $r = 20$ full rows and $u = 4$ long columns.

In [63], Giddy and Safavi-Naini use simulated annealing for the cryptanalysis of columnar transposition ciphers. They use bigram statistics for the cost function, as well as random transformations called G-Transpositions, which are equivalent to the segment slides described in our work (see Section 5.3.2). Their algorithm is able to reproduce at least 80% of the key elements in the following CCT cases:

- $|K| = 15$ and $|C| = 255$ $(r = 17)$
- $|K| = 20$ and $|C| = 500$ $(r = 25)$
- $|K| = 25$ and $|C| = 500$ $(r = 20)$

In [16], Clark compares three types of attacks on columnar transposition ciphers: (1) genetic algorithm, (2) simulated annealing, and (3) hill climbing/tabu search. For scoring, Clark uses a slightly modified version of Matthew's subset of bigrams and trigrams. The two transformations used in his study for all three types of attacks are (a) swapping two random key elements and (b) random rotations of consecutive key segments. The results for the three methods are similar. For a key of length $|K| = 15$, the recovery of at least 14 of the 15 key elements requires a ciphertext with at least $|C| = 450$ to 600 characters which is equivalent to $r = 30$ to 40.

In [17], Russell, Clark and Stepney combine the classical method of multiple anagramming with an ant colony optimization algorithm. As cost functions, they use both bigram scores as well as

dictionary matches. With this method, keys can be fully recovered for the following cases (all of which are CCT):

- $|K| = 15$ and $|C| = 300$ ($r = 20$)
- $|K| = 20$ and $|C| = 400$ ($r = 20$)
- $|K| = 25$ and $|C| = 625$ ($r = 25$)

In [14], Dimovski and Gligoroski use a genetic algorithm, simulated annealing, and tabu search to solve columnar transposition ciphers with scoring based on bigrams. The transformation used for simulated annealing and tabu search (and for mutations in the genetic algorithm) consists of the swapping of two random key elements. Results are similar for all three techniques, as follows:

- For a key of length $|K| = 15$, at least $|C| = 800$ ($r = 53$) letters are required to recover at least 12 of the 15 key elements.
- For $|K| = 30$, a ciphertext of at least $|C| = 1000$ ($r = 33$) letters are required to recover 25 of the 30 key elements.

In [19], Chen and Rosenthal apply a Markov Chain Monte Carlo algorithm to solve columnar transposition ciphers, using bigram scoring. They apply transformations consisting of (a) swaps of single key elements and (b) slides of key segments. Their results are as follows:

- To solve a key of length $|K| = 20$, between $|C| = 500$ to 1000 characters of ciphertext are required (r from 25 to 50).
- Keys with $|K| = 30$ can also be recovered (with $> 80\%$ probability of success) with $|C| = 2000$ ($r = 66$).

Finally, in *Decoding the IRA* [61], Jim Gillogly describes how he solved hundreds of messages encrypted with columnar transposition, with keys no longer than 15 elements, using hill climbing. For most messages, the transposition rectangles were incomplete.

The results from prior works are summarized in the Table 5.3. For several key lengths, we show the minimum length of ciphertext required by each method to recover at least 80% of the elements of the correct key. We also show the shortest length of ciphertext required by any of those methods.

From those results, we can see that the scope of those prior algorithms is rather limited, as well as their performance, in term of maximum key length, the minimal ciphertext lengths required for successful cryptanalysis, and their ability to handle ICT cases. Except for Matthews which is not shown here as it relates only to very short keys ($|K| < 10$) [62], and Gillogly [61], no prior published work addresses the more challenging case of ICT. Gillogly successfully applied his methods to ciphertexts shorter than 100 letters, with keys of length of up to 15 elements, but no performance data is available for longer keys. Historically, the operational use of columnar transposition almost always involved ICT. Prior methods are also limited to short keys, with typically 25 or fewer elements, and they require long ciphertexts, in order to successfully recover the key.

|K|	**Giddy** [63]	**Clark** [16]	**Russel** [17]	**Dimovski** [14]	**Chen** [19]	**Shortest length**
CCT						
15	255	450	300	990		**255**
20	500		400	1 000	500	**400**
25	750	1 000	625	1 000		**625**
30				990	2 000	**990**
> 50	Not covered in prior work					
ICT	Limited to short keys (up to 10-15 elements)					

TABLE 5.3: Columnar transposition – prior work performance

5.3 A New Ciphertext-only Attack

The new attack was developed along the lines of our new methodology, described in Chapter 4. It is based on a sequential two-phase hill climbing algorithm (see Section 4.3) with multiple restarts designed to produce high-quality initial keys for the main search (see Section 4.7). It implements adaptive scoring, using new specialized scoring methods, to achieve better resilience to errors in the first phase (see Section 4.5), as well as new types of transformations designed to improve the search coverage (see Section 4.6).

We start in Section 5.3.1 with the description of a baseline hill climbing algorithm, which is similar to several of the algorithms used in prior works. The purpose of this baseline algorithm is to serve as a basis for further improvements, and also as a reference for performance evaluation. We then incrementally improve this baseline algorithm.

In Section 5.3.2, we introduce new types of non-disruptive key transformations, *segment slides* and *segment swaps*, tailored for the columnar transposition cipher. Those new transformations significantly increase the search coverage. This version of the algorithm is effective with mid-length keys (CCT), and also with short keys in the case of ICT.

In Section 5.3.3, we add an initial phase, to produce high-quality initial keys for the main phase. This initial phase employs a new specialized scoring function – the *adjacency score*. This version of the algorithm is able to solve CCT cases with long keys.

In Section 5.3.4, we present the most complex variant of the algorithm, intended to solve Incomplete Columnar Transposition (ICT) cases with long keys. This hill climbing algorithm also has two phases, the initial phase using a new specialized scoring function – the *alignment score*, in addition to a modified adjacency score.

In each section, we analyze the performance of the relevant algorithms and their improvements, for several CCT and ICT scenarios, covering various values of $|K|$ and r (as well as u for ICT). Performance includes the success rate (the probability of fully recovering the entire key), as well as the work factor.

In Table 5.4, we summarize the various enhancements to the baseline hill climbing algorithm, described in detail in the following sections. For each algorithm, we specify the scoring methods and the transformations used, and the purpose of the enhancements.

Section	Methodology principles	Enhancement	Results
5.3.1	GP1	Swaps of single key elements Log quadgrams Single phase	Baseline implementation, comparable to prior work CCT: short keys, up to 30 No support for ICT
5.3.2	GP1,4	Segment swaps and slides	CCT: Mid-length keys, up to 100, and shorter ciphertexts (180 vs. 990 for key length 30) ICT: Short keys, up to 30
5.3.3	GP1,3,4,5	Preliminary phase with adjacency score	CCT: Very long keys, up to 1 000
5.3.4	GP1,3,4,5	Preliminary phase with adjacency score and alignment score	ICT: Long keys, up to 120

TABLE 5.4: Columnar transposition – algorithm enhancements

For all the methods described in this chapter, it is assumed that the length of the transposition key, $|K|$, is known. If it is not known, the algorithms must be run multiple times, for each possible key length.

5.3.1 Baseline Hill Climbing Algorithm

We implemented a baseline hill climbing algorithm, which we use as a reference for performance comparison, and as the basis for improvements described later in this section. The baseline hill climbing implementation consists of the following steps:

1. Generate an initial random transposition key.
2. For every possible transformation on the current key:
   (a) Apply the transformation to the current key, and obtain a new candidate key.
   (b) Decrypt the ciphertext using the candidate key, to obtain the putative plaintext.
   (c) Score the putative plaintext according to the scoring function.
   (d) If the score is higher than the score before the transformation, keep the candidate key as the current key.
3. If an improvement in score cannot be achieved anymore by transformations on the current key, either we have found the correct key (and we stop), or reached a local maximum, in which case we restart the whole process from Step 1.

The transformations on the key consist of swapping two key elements. This type of transformation is also used in most of the prior work publications mentioned in the references. All possible swaps of two elements of the key are checked at each iteration of hill climbing, with a total of

$|K| \cdot (|K|-1) \div 2$ options. As the scoring function, we use the sum of the log-frequencies of quadgrams (4 consecutive characters) in the putative plaintext.

In Table 5.5 we summarize the results with this baseline algorithm, using simple swaps and a single phase. The performance of this baseline implementation is similar to the results with methods developed in prior works. In all cases, the transposition rectangles are complete (CCT case).

Key length $\lvert K\rvert$	**Ciphertext length** $\lvert L\rvert$	**Average % of key elements recovered**
15	255 ($r = 17$)	76%
20	400 ($r = 20$)	85%
25	625 ($r = 25$)	88%
30	990 ($r = 33$)	87%

TABLE 5.5: Columnar transposition – performance of baseline algorithm for CCT

5.3.2 Improved Algorithm for Mid-Length Keys

The first improvement is intended to allow for the algorithm to work with shorter ciphertexts, to handle mid-length keys (up to $|K| = 100$) for the case of CCT, and short keys (up to $|K| = 30$) for the case of ICT. This is achieved by introducing new types of transformations, as described here.

Initially, we experimented with the use of a new type of transformation, the *Segment Swap*. This transformation consists of swapping two segments of consecutive key elements. During hill climbing, we check all possible key Segment Swap transformations, at all possible starting positions p_1 and p_2 of the two segments, and for every segment size l, provided the segments do not overlap. In the following example, the key is ABCDEFGHIJ. The underlined letters represent the two segments of length $l = 2$ we wish to swap, "CD" and "HI", which start at positions $p_1 = 3$ and $p_2 = 8$, respectively. After swapping the segments, we obtain the following key: ABHIEFGCDJ.

We modified the baseline algorithm to use segment swaps instead of single swaps, but the improvement was marginal and almost negligible. Next, we tried a new type of transformation, the *Segment Slide* (could also be described as a cyclic rotation or shift). With this type of transformation, we select a segment of length l, starting at position p, and we slide (or shift) that segment to the right, s times. In the following example, the key is ABCDEFGHIJ, and the underlined letters represent the segment we want to slide, 3 times to the right ($l = 3,\ p = 3,\ s = 3$). We obtain the following key, after the Segment Slide transformation: ABFGHCDEIJ. The underlined letters are the letters affected by the transformation. Note that while sliding the segment "CDE", the segment "FGH" has also been shifted in the process.

An important characteristic of segment-wise transformations (i.e. segment slides and segment swaps) is that, as their name implies, they operate on contiguous segments of key elements. Therefore, most adjacency relations, i.e. which column j is on the immediate right of another column i, are preserved even though a large number of key elements may change their position as a result of such a transformation on the key.

We tested the performance of the modified algorithm using both types of transformations, segment slides and segment swaps. In all the scenarios shown in Table 5.5, the algorithm recovered 100% of the correct key elements. We then tested the improved algorithm with both segment slides and segment swaps as transformations, on more challenging cases, including shorter ciphertexts, longer keys and ICT, that neither prior methods, nor the equivalent baseline algorithm, were able to solve successfully. The results are presented below.

Performance with Shorter Messages

Table 5.6 shows the minimal ciphertext length (CCT cases) required to achieve at least 90% probability of full key recovery. We compare this to the length of ciphertext required by the best prior method (or the baseline algorithm). For example, the length of ciphertext required for $|K| = 30$, is only 180, compared to 990 in prior work, i.e. less than 20%.

	Hill climbing using segment slides	**Prior work and baseline algorithm**
Key length $\|K\|$	**Minimal length of ciphertext $\|C\|$**	**Minimal length of ciphertext $\|C\|$**
15	75 ($r = 5$)	255 ($r = 17$)
20	120 ($r = 6$)	400 ($r = 20$)
25	150 ($r = 6$)	625 ($r = 25$)
30	180 ($r = 6$)	990 ($r = 33$)

TABLE 5.6: Columnar transposition – performance with segment slides for CCT and short keys

Performance with Mid-Length Keys (CCT)

We can see that this improved algorithm is also able to fully recover CCT keys with lengths of up to least 100 elements. Prior work and the baseline algorithm were unable to recovery keys longer than 30.

	Hill climbing using segment slides	**Prior work and baseline algorithm**
Key length $\|K\|$	**Minimal length of ciphertext $\|C\|$**	**Minimal length of ciphertext $\|C\|$**
50	450 ($r = 9$)	No solution, regardless of length
75	675 ($r = 9$)	No solution, regardless of length
100	1100 ($r = 11$)	No solution, regardless of length

TABLE 5.7: Columnar transposition – performance with segment slides for CCT and long keys

Sporadic tests were successfully run on even longer keys, up to 200 long. For keys longer than 100, the algorithm may succeed, but the time required for a solution is very long, since the number of transformations to check is proportional to $|K|^3$. For example, at $|K| = 100$, at each step of the hill climbing process, $1\,413\,751 \approx 2^{21}$ unique transformations, according to our simulations, must be checked. In the next section (5.3.3) we introduce further optimizations, intended not only to increase the probability of success, but also to increase the **speed** of key recovery, for the case of CCT with very long keys.

Performance in case of ICT

To evaluate the performance of the improved algorithm on ICT cases, we tested it on the worst case ICT scenario, which is when the number of long columns is equal or approximately equal to the number of short columns, i.e. $u = \frac{|K|}{2}$. The results are shown in Table 5.8.

Key length $\|K\|$	**Minimal length of ciphertext $\|C\|$**
15	97 ($r = 6, u = 7$)
20	210 ($r = 10, u = 10$)
25	387 ($r = 15, u = 12$)
30	615 ($r = 20, u = 15$)
50	2525 ($r = 50, u = 25$) 70% success rate
100	Consistently fails regardless of length

TABLE 5.8: Columnar transposition – performance with segment slides for ICT

It can be seen that for ICT, this algorithm is effective on short keys (length 15-20), less effective on keys of length 25-30 as it requires a very long ciphertext, rather ineffective for keys of length 50, and completely ineffective on longer keys. In Section 5.3.4, we introduce an enhanced algorithm, capable of recovering keys for ICT cases with much shorter ciphertexts, as well as longer ICT keys, up to $|K| = 120$.

From now on, all further evaluations will be done using both segment slides and segment swaps as transformations. We later come back and evaluate their respective contribution, in the context of the final algorithms.

5.3.3 Two-Phase Algorithm for CCT and Very Long Keys

The purpose of the next improvement, described here, is to allow for successful cryptanalysis in the case of CCT used with very long keys, and to speed up the algorithm for CCT in general. One of the main characteristics (and weaknesses) of the columnar transposition cipher is that columns of the original plaintext are kept intact and appear as continuous segments of text in the ciphertext (see Figure 5.1). The algorithm described in this section takes advantage of this characteristic. It relies on the relationships between the columns of the ciphertext, which are also the columns of the original plaintext, but transposed. More specifically, it tries to identify which columns in the ciphertext transposition rectangle, are columns that originally were *adjacent* in the plaintext rectangle, before transposition.

We first examine how adjacent columns are reflected in transposition keys. For that purpose, it is more convenient to look at numerical keys. In our example in Figure 5.1, the numerical key is (3,2,7,6,4,5,1). This numerical transposition key indicates that the first column of the plaintext rectangle should be transposed to be the 3$^{\text{rd}}$ column in the ciphertext rectangle. Similarly, it specifies that the second column in the plaintext will be the second column in the ciphertext, that the 3$^{\text{rd}}$ column in the plaintext will be the 7$^{\text{th}}$ column in the ciphertext, and so on for the remaining columns. This also means that in the original plaintext, the plaintext column which is now ciphertext column 2 was originally on the right of the plaintext column which is now ciphertext column 3. Furthermore, the plaintext column which is now ciphertext column 7 was adjacent to, and on the right of the plaintext column which is now ciphertext column 2, and

so on. Therefore, from the transposition key, (3,2,7,6,4,5,1), we can easily deduce all adjacent pairs, namely (3,2), (2,7), (7,6), (6,4), (4,5) and (5,1).

In a ciphertext-only attack, however, we do not know the key. Still, we can try to evaluate the likelihood, for any given pair of columns i and j, that columns i and j were adjacent in the original plaintext (before transposition), with column j being on the immediate right of column i. For that purpose, we introduce a new scoring function, the *adjacency score*. This score is computed by summing up the log-frequencies, according to language statistics, of all the bigrams resulting from the juxtaposition of column j to the right of another column i. Furthermore, we generalize this concept of adjacency score, to a candidate key, which as we saw before, specifies putative pairs of adjacent columns. The adjacency score for a candidate key is computed by simply summing up the adjacency scores of all those (putatively) adjacent pairs of columns.

In our algorithm, we implemented an even more precise adjacency score and therefore a better "adjacency prediction". For each pair of columns i and j, we check the triplet of columns (i,j,k), for each possible third column k, and sum up the log-frequencies of the trigrams generated by the juxtaposition of columns i, j and k. We keep the best value, for any possible k, as the adjacency score for the pair i and j. We use this improved method for keys with up to 250 elements. For longer keys, however, the number of possible i, j and k combinations to check is too high. For example, for $|K| = 500$, the number of triplets to check is $500 \cdot 499 \cdot 498 = 124\,251\,000$. Therefore, for keys longer than 250 elements, we use the simpler implementation of the adjacency score, using bigrams statistics.

We now introduce an improved hill climbing algorithm, for the case of CCT with very long keys. We add a new phase (Phase 1), which relies on the adjacency score. This improved two-phase algorithm is described here:

1. Phase 1:
   (a) Pre-compute the adjacency score for every possible pair of columns i and j.
   (b) Generate a random initial candidate key, and compute its adjacency score, by summing up the adjacency scores of all the resulting (putative) pairs of adjacent columns.
   (c) Iteratively perform the following, for every possible Segment Slide and Segment Swap transformation on the current key.
      i. Apply the transformation to the current key to obtain a candidate key.
      ii. Compute the adjacency score for the candidate key.
      iii. If the score is higher than the score before the transformation, keep the candidate key as the current key.
   (d) If no additional improvement in score can be achieved by transformations on the current key, go to Phase 2, using the current key as the initial key for Phase 2.
2. Phase 2:
   (a) Iteratively perform the following, for every possible Segment Slide and Segment Swap transformation on the current key.
      i. Apply the transformation to the current key to obtain a candidate key.
      ii. Decrypt the ciphertext using the candidate key to obtain the putative plaintext.
      iii. Score the putative plaintext using quadgram log-frequencies.
      iv. If the score is higher than the score before the transformation, keep the candidate key as the current key.

(b) If no improvement in score can be achieved by transformations on the current key, stop.

Phase 1 produces a much higher quality initial key than a random initial key, and allows Phase 2 to converge much faster. Often, Phase 1 may produce an almost correct key, and in some cases, even a fully correct key, in which case there is no need for Phase 2. The performance for CCT and keys with up to 75 elements is illustrated in Figure 5.2. For each key length, several ciphertext lengths were tested. The X-axis shows the key length, the Y-axis shows the number of rows $r = \frac{|C|}{|K|}$, and the Z-axis the percentage of tests with full key recovery.

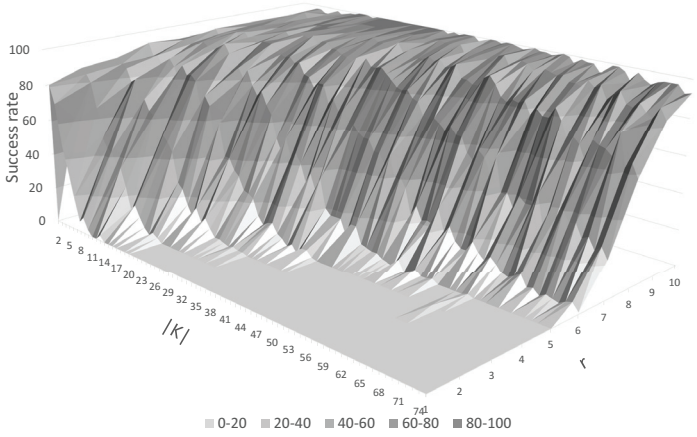


FIGURE 5.2: Columnar transposition – performance with two phases for CCT

This faster algorithm also allowed for the cryptanalysis of CCT cases with very long keys, with up to 1 000 elements. Such long keys are not realistic from the operational perspective. We evaluated the performance of the algorithm with those very long keys, only for the purpose of finding out its upper limits. In Figure 5.3, the percentage of key elements recovered is displayed, for various numbers of rows r, and keys of lengths $|K|$ from 100 to 1 000. As can be observed, complete key recovery can be achieved for keys of length $|K| = 100$ with only $r = 10$ rows, for length 150 with 15 rows, and for length 200 with 20 rows. For keys of length 250, Phase 2 of the algorithm takes a very long time, because of the very high number of transformations to check at each iteration. Therefore, we simplified the algorithm for very long keys (longer than 250), to include only Phase 1. Not running Phase 2 on the one hand reduces the probability of full key recovery, but Phase 1 alone is often enough to recover almost all the elements of very long keys, given enough ciphertext. In other cases, it will produce an almost correct key. About 20-25 rows

are needed for a key of length 250, and about 25-30 rows for keys of length 1000, for full key recovery.

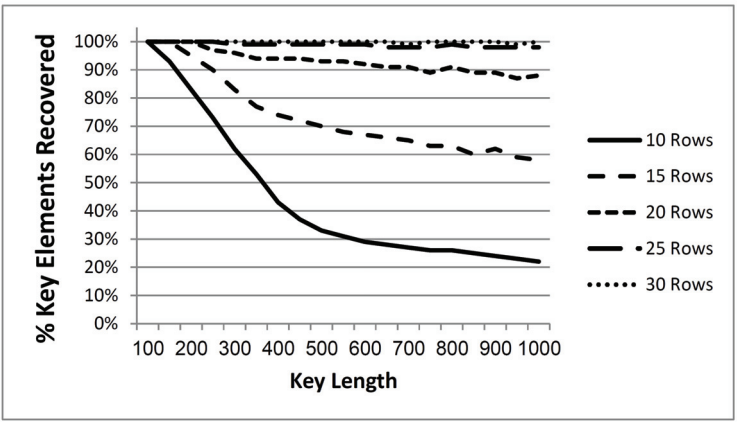


FIGURE 5.3: Columnar transposition – percentage of key elements recovered with two phases for CCT and long keys

We also evaluated the usefulness of the two types of transformations, namely the segment swaps, and the segment slides. In Figure 5.4, the percentage of cases with full key recovery is shown for keys of length $|K| = 50$ (CCT), with various numbers of rows ($r = \frac{|C|}{|K|}$). It can be seen that segment swaps alone are significantly less effective compared to segment slides alone. A slight improvement, however, can be achieved by using both types of transformations, compared to using segment slides alone.

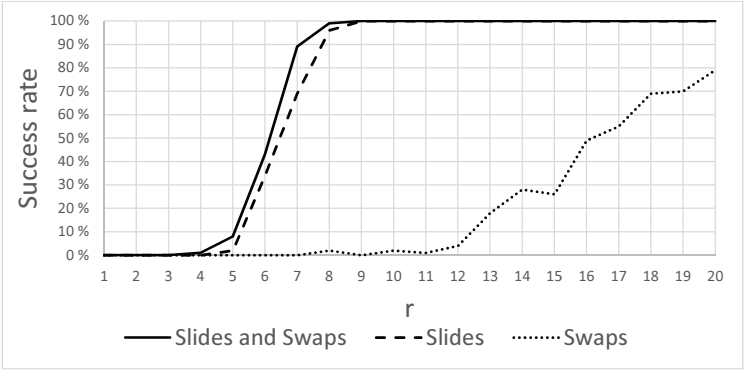


FIGURE 5.4: Columnar transposition – performance with segment slides vs. segment swaps

Next we evaluated the work factor for the algorithm, when applied to CCT. In Table 5.9, we show the average number of decryptions required for full key recovery, for various CCT key length

scenarios. For each length, we include an extreme case, with the minimum number of rows required for full key recovery, as well as a moderate case, with more rows. For comparison, the size of the keyspace is $|K|!$. For $|K| = 25$, it is about 10^{25} (or 2^{84}), for $|K| = 50$ it is about 10^{64} (2^{215}), and for $|K| = 100$ it is about 10^{158} (2^{525}). It can be seen that the number of decryptions required for successful cryptanalysis is significantly smaller than the size of the key space.

Key length $\|K\|$	**# of rows** r	**# of decryptions**
25	6	0.336 M
25	10	0.093 M
50	8	2.152 M
50	15	0.164 M
100	10	15.43 M
100	20	2.708 M
200	20	39.43 M
200	40	7.251 M

TABLE 5.9: Columnar transposition – work factor for CCT

5.3.4 Two-Phase Algorithm for ICT

The purpose of this final improvement is to allow the algorithm to handle the more complex case of ICT, including for long keys. The algorithm for CCT and long keys described in Section 5.3.3, is not applicable to the case of ICT. In the case of CCT, the ciphertext transposition rectangle is complete, and all columns have exactly r elements. Therefore, the starting positions of all ciphertext columns are known. In the case of CCT, ciphertext column 1 starts at position 1, column 2 starts at position $r+1$, column 3 starts at position $2r+1$, and so on.

With ICT, however, each column in the ciphertext may be either a long column, with $r+1$ elements, or a short column, with only r elements. The starting position of each column in the ciphertext, therefore, depends on each of the previous columns being either short or long. Unless the key is known, there are several possible starting positions for each column. To illustrate this, let's assume we have a ciphertext C and a key K, so that half of the columns are long and the rest are short. For convenience, we assume in this example that the length of the key, $|K|$, is even. The number of long columns u and the number of full rows r are as follows:

$$u = |C| \ mod \ |K| = \frac{|K|}{2} \tag{5.1}$$

$$r = \lfloor \frac{|C|}{|K|} \rfloor \tag{5.2}$$

The first ciphertext column may start only at the beginning of the ciphertext, at position 1. The second column may either start at position $r+1$ or $r+2$, depending on whether the first column was a short column (with r elements), or a long column (with $r+1$ elements). The third column may start either at position $2r+1$, at $2r+2$, or at $2r+3$, and so on for the following columns. For the column in the middle of the ciphertext, there may be up to approximately $\frac{|K|}{2}$ starting positions.

In order to compute the adjacency score of any pair of columns i and j, we first need to know the starting positions of both i and j, or at least the relative difference or offset *modulo r* between those starting positions, so that we may correctly align the two columns. But we can't achieve that unless we know the key. Because of this ambiguity in the starting positions of columns, we need to adapt the computation of the adjacency score, for the case of ICT.

While we do not know the starting positions of the columns in the ciphertext, we still can try to estimate, for a candidate key, to what extent the starting positions of columns in adjacent pairs are well aligned. For that purpose, we introduce a new type of scoring score, the *alignment score*.

We present here the modified adjacency score, as well as the new alignment score.

Modified Adjacency Score

To adapt the adjacency score to ICT, we must now compute the adjacency score for columns (i, j), as described in the previous section for CCT, for every combination of the possible starting positions, p_i and p_j, for those columns. We retain the best adjacency score obtained with the best combination of p_i and p_j, as the final adjacency score for (i, j). In addition, we compute the difference between those optimal p_i and p_j starting positions, and keep their difference *modulo r* in an *offset matrix* $O[i, j]$, with $O[i, j] = (p_j - p_i) \bmod r$, with r being the number of full rows. Note that we need to compute this offset matrix only for the case of ICT. In the case of CCT, all columns have the same length, and all offsets *modulo r* are equal to 0, therefore this offset matrix is irrelevant.

The Alignment Score

To understand what this new score is intended to measure, we look again at the transposition example in Figure 5.1. The key length $|K|$ is 7, and the number of full rows is $r = \lfloor \frac{|C|}{|K|} \rfloor = \lfloor 45/7 \rfloor = 6$. We analyze here the alignment of columns 2 and 7 in the ciphertext. Those two columns were adjacent in the original plaintext rectangle, before the transposition. Both are long columns, with $r+1 = 7$ elements. Column 2 consists of the sequence of letters *HINIEPX* and columns 7 of the sequence *ECSOROT*. Column 2 starts at position 7 in the ciphertext, and column 7 at position 33. When juxtaposed and correctly aligned they generate the bigrams *HE*, *IC*, *NS*, *IO*, *ER*, *PO* and *XT*, seven bigrams in total, which originally appeared in the plaintext. This is shown in part (1) of Figure 5.5.

If we do not know the key, all we know is that column 2 could have started either at position 7 or 8 depending on whether column 1 was short or long. Column 7 is the last column, it could have started at position 33 or 34, depending on whether it was a long or short column.

If we are able, however, to guess the correct starting positions of columns 2 and 7, as shown in part (1) of Figure 5.5, all the seven original bigrams can be reconstituted. In this case, the difference *modulo r* between those correct starting positions is $(33-7) \bmod 6 = 2$.

In case we wrongly guess one of the starting positions and correctly guess the second one, then none of the original correct bigrams can be reconstituted. In the example illustrated in part (2) of Figure 5.5, we correctly guessed the starting position (33) of column 7, but the starting position (8) of column 2 was wrong. In this case, the difference *modulo r* between the starting positions of the columns is $(33-8) \bmod 6 = 1$, and all reconstituted bigrams are wrong.

However, if we are wrong for both columns, so that we guess column 2 to start at position 8 and column 7 to start at position 34, 6 out of the correct 7 can still be reconstituted (*IC*, *NS*, *IO*, *ER*,

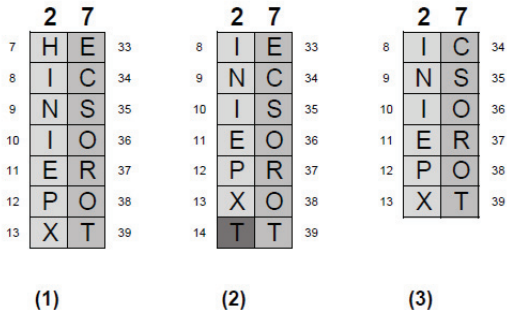


FIGURE 5.5: Columnar transposition – alignment of adjacent columns

PO and *XT*), as shown in part (3) of Figure 5.5. While the starting positions of the columns are wrong, their *alignment relative to each other* is correct. The difference *modulo r* between the starting positions of the columns is also $(34-8) \bmod 6 = 2$, as for the case of the correct starting positions.

From this example, we can see that to reconstitute all or almost all of the correct original bigrams, we must be able to align the two columns so that the difference *modulo* 6 between of their starting positions is correct, i.e. 2. In part (2) of Figure 5.1 which shows the ciphertext transposition rectangle, it can be seen that between the start of column 2 and the start of column 7, there are exactly two long columns, those two being column 2 itself, as well as column 3. **The key observation here is that those two long columns account for the offset *modulo r*, which value is 2, between the starting positions of those columns.** Accordingly, with any candidate key in which columns 2 and 7 are adjacent, we may be able to reconstitute all (7) or almost all (6) of the original bigrams obtained by juxtaposing columns 2 and 7, only if the number of long columns between the starting positions of the two columns (2 and 7), according to the candidate key, is correct and equal to 2.

We generalize this idea and define an *alignment score for a candidate key*. This score is intended to measure how well the text of adjacent columns is aligned, as defined above, this time looking at all pairs of adjacent columns, as reflected by the candidate key. As mentioned before, when computing the (modified) adjacency score for the ICT case, we also retained the offset matrix $O[i, j]$ for each pair of columns i and j. This offset $O[i, j]$ is the difference *modulo r* between the starting positions of the two columns, for those starting positions that generated the highest adjacency score for the pair (and also the best bigrams). We use this offset matrix $O[i, j]$ to compute the alignment score. We compute the alignment score, for a candidate key, by processing all pairs of adjacent columns i and j, as follows:

1. Count the number of long columns, according to the candidate key, between the two columns i and j, including i itself, but excluding j.

2. Compare this number to the value of $O[i, j]$ (obtained when computing the adjacency score).
   (a) If they are equal, add 2 points to the alignment score of the candidate key.
   (b) If they are equal, and we had previously also added points for the previous pair of adjacent columns k and i, k being the column on the left of i, then add an additional point to the alignment score. This is intended to assign a higher score to consecutive sequences, in the candidate key, of well-aligned adjacent pairs of columns.

Next, we show how we use both the adjacency score and the alignment score to generate, in Phase 1, an optimal starting key for Phase 2. Following is the improved Phase 1 algorithm, for the case of ICT:

1. Phase 1:
   (a) Pre-compute an adjacency score matrix for all possible pair of columns i and j such as $1 \leq i \leq |K|, 1 \leq j \leq |K|, i \neq j$, using the modified adjacency score method described earlier in this section. As a by-product, the offset matrix $O[i, j]$ is also computed.
   (b) Generate a random initial candidate key.
   (c) Set an initial minimal threshold for the adjacency score, and an initial minimal threshold for the alignment score.
   (d) Iterate as follows, up to 15 times (or stop sooner if no more improvement could be achieved):
      i. Iteratively perform the following, for every possible Segment Slide and Segment Swap transformation on the current key:
         A. Apply the transformation to the current key to obtain a candidate key.
         B. Compute the primary score, the adjacency score, for the candidate key. If it is lower than the score before the transformation, discard the candidate key. Otherwise, also compute the secondary score, the alignment score, for the candidate key. If it is below the current alignment score threshold, discard the candidate key. Otherwise, keep the candidate key as the new current key.
      ii. Iteratively perform the following, for every possible Segment Slide and Segment Swap transformation on the current key.
         A. Apply the transformation to the current key to obtain a candidate key.
         B. Compute the primary score, this time the alignment score, for the candidate key. If it is lower than the score before the transformation, discard the candidate key. Otherwise, also compute the secondary score, the adjacency score, for the candidate key. If it is below the current adjacency score threshold, discard the candidate key. Otherwise, keep the candidate key as the new current key.
      iii. Increase the thresholds for both the adjacency score and for the alignment score.
   (e) Use the last current key as the initial key for Phase 2, and start Phase 2. Phase 2 algorithm is the same as in Section 5.3.3.

The goal of this Phase 1 algorithm is to maximize both the adjacency and the alignment scores. We could have instead used a linear combination of those scores as a single target function, but experiments in that direction proved fruitless. Instead we developed this two-dimensional search. At each main iteration, we alternate the primary scoring method, using either the adjacency score or the alignment score, as the score we wish to maximize. We also compute a secondary score (the other score) but only verify that it is above a minimum threshold. Therefore, the secondary score might often decrease while the primary score is increasing, generating a "zigzag" pattern. However, the overall tendency is for both scores to increase, as we raise the thresholds for the secondary scores after each iteration. In practice, after a few iterations, the zigzag phenomenon tends to stop, and both scores increase together. This is illustrated in Figure 5.6.

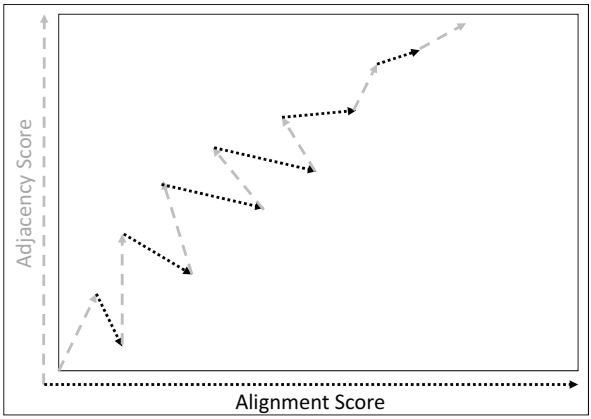


FIGURE 5.6: Columnar transposition – Phase 1 hill climbing with adjacency and alignment scores

We evaluated the performance of this improved algorithm, for keys up to 120 long, for the worst case of ICT. The worst case for ICT is when the number of long columns u is equal to the number of short columns, or approximately equal to $\frac{|K|}{2}$. In Table 5.10, we show the minimum ciphertext length required for full key recovery (and 100% success rate), for several key lengths. For comparison, for keys of length $|K| = 50$, the minimum required ciphertext length is only 825 compared to 2525 with the algorithm described in Section 5.3.2 (Table 5.8), and the success rate is 100% vs. 70%. In addition, this algorithm performs well with relatively long keys, up to $|K| = 120$, as shown in Table 5.10. Sporadic tests showed that it may work for even longer keys.

We also analyzed the performance of the algorithm in relation to the number of long columns u (Figure 5.7). As can be seen, the worst case, requiring the highest number of rows r (and therefore the longest ciphertext) for its solution, is indeed the case where the number of long columns is $u \approx \frac{|K|}{2}$. Note that performance for $u \approx |K| \cdot \frac{3}{4}$, while not shown here, is similar to the case of $u \approx \frac{|K|}{4}$.

Finally, we evaluated the work factor for the improved algorithm for ICT, for the worst case of ICT ($u \approx \frac{|K|}{2}$). In Table 5.11, we show the average number of decryptions required for full key recovery, for various ICT key length scenarios. For each length, we include an extreme case,

Key length $\|K\|$	Ciphertext length $\|C\|$
15	97 ($r = 6, u = 7$)
20	190 ($r = 9, u = 10$)
25	287 ($r = 11, u = 12$)
30	375 ($r = 12, u = 15$)
50	825 ($r = 16, u = 25$)
75	1 687 ($r = 22, u = 37$)
100	2 850 ($r = 28, u = 50$)
120	4 260 ($r = 35, u = 60$)

TABLE 5.10: Columnar transposition – performance for worst case ICT with two phases

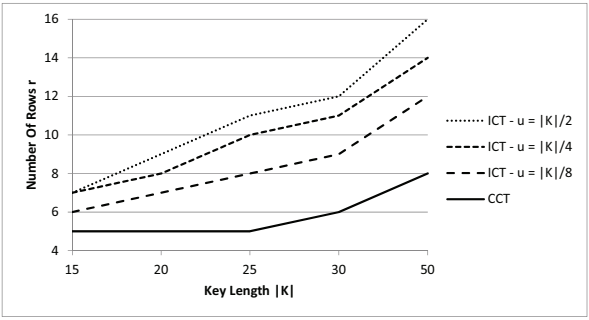


FIGURE 5.7: Columnar transposition – performance for ICT by number of long columns u

with the minimum number of rows required for full key recovery, as well as a moderate case, with more rows.

Key length $\|K\|$	Number of rows r	Work factor (number of decryptions)
25	11	3.872 M
25	20	0.116 M
50	16	36.253 M
50	25	5.590 M
100	28	53.603 M
100	40	6.782 M

TABLE 5.11: Columnar transposition – work factor for ICT

5.4 Summary

In prior work on the computerized cryptanalysis of the classical columnar transposition cipher, the approach was to directly apply one of several generic algorithms, such as hill climbing,

genetic algorithms or simulated annealing. This naive approach, while easy to implement, had severe limitations. It worked only for short keys, long ciphertexts, and could not effectively address the more challenging case of ICT.

The work presented here demonstrates the effectiveness of the methodology for the cryptanalysis of cryptanalysis of columnar transposition ciphers with challenging settings, as summarized in Table 5.12. We significantly enhanced a baseline algorithm, using specialized transformations (segment slides), specialized scoring functions (adjacency and alignment scores), as well as the addition of a preliminary phase which relies on a deep analysis of the relationships between the columns of the ciphertext.

Principle	**Application of the methodology principle**
GP1	Hill climbing, sequential (2-phase) search
GP2	
GP3	In the first phase, adjacency and alignment scores, with high resilience In the second phase, using the more selective log-quadgrams
GP4	Non-disruptive transformations applied to key segments Variable neighborhood search
GP5	Multiple restarts, with first phase to generate optimal initial keys

TABLE 5.12: Columnar transposition – applying the methodology

Our method is highly effective for the cryptanalysis of columnar transposition ciphers when encrypted with very long keys, up to $|K| = 1\,000$ elements for the case of CCT, and up to $|K| = 120$ elements for the more complex case of ICT. It also requires much less ciphertext material than prior methods. For example, in the case of CCT, for a key with $|K| = 30$ elements, a ciphertext with only $|C| = 180$ letters can be cryptanalyzed with this new algorithm, compared to at least $|C| = 990$, or 5 times more, for the most efficient of the other prior methods.

6

Case Study – The ADFGVX Cipher

In the last months of the WWI, the German Army and diplomatic services used the ADFGVX hand-cipher system to encrypt radio messages between Germany and its outposts and stations in the Balkans, the Black Sea and in the Middle East. Hundreds of cryptograms were intercepted from July to December 1918 by British and US military intelligence, who were able to recover most of the keys, and decipher most of the cryptograms using manual cryptanalysis methods. Fortunately, the original cryptograms have been preserved by James Rives Childs, the US officer assigned to G.2 A.6, the SIGINT section of American Expeditionary Forces (AEF) in Paris, and they appear in his book, "'General Solution of the ADFGVX Cipher System"', published by Aegean Press Park in 2000.

In this chapter, we present the results of an effort towards the complete cryptanalysis of the messages, and an analysis of their contents. We present a new computerized method for the ciphertext-only cryptanalysis of ADFGVX messages which we developed. We also provide details on how all the keys were recovered and almost all of the messages decrypted, despite the low quality of significant parts of the intercepted material. To develop an efficient ciphertext-only attack on ADFGVX, we applied the guidelines of the new methodology described in Chapter 4. The new attack is based on a divide-and-conquer attack, in two phases. In the first phase, we recover the transposition key using hill climbing with multiple and optimized restarts, specialized scoring methods and a rich set of non-disruptive transformations. The second phase which recovers the substitution key also implements hill climbing.

The ADFGVX case study not only demonstrates the effectiveness of our new methodology, but also shows how the study of historical ciphers and research on new cryptanalytical methods may assist in revealing important historical material which would not have been accessible otherwise. The analysis of the messages in their historical context provides a unique insight into key events, such as the withdrawal of the German troops from Romania, and the impact of the Kiel Mutiny on communications. Both events had major political and military consequences for Germany in the East Front.

This chapter is organized as follows. In Section 6.1, we provide some historical background information about the ADFGVX cipher and intelligence in WWI in general. In Section 6.2, we provide a description of the ADFGVX cipher. In Section 6.3, we provide a review of historical cryptanalytic methods against ADFGVX, as well as modern methods. In Section 6.4, we present a new computerized method we developed for this research, and in Section 6.5, we describe the process of how most of the messages were decrypted. In Section 6.6, we present an analysis of

German and Allies' cryptographic and cryptanalytic capabilities, as well as a profile of James Rives Childs, and the analysis of several key historical events reflected in the deciphered radiograms. Finally, in Section 6.7, we conclude our findings from the technical and historical perspectives.

The results presented in this chapter have also been published in Cryptologia [34].

The research for the historical analysis in Section 6.6 was conducted by Dr. Ingo Niebel. Ingo Niebel is a historian and freelance journalist living and working in Germany. He received his PhD from the University of Cologne in 2012. Since 1996, he is also a member of Eusko Ikaskuntza, the Society of Basque Studies. He has published several books in German and Spanish about the Basque Country and intelligence during the Spanish Civil War (1936–1939). During his research he uncovered original Civil War telegrams encrypted using the Spanish Strip Cipher, that could not be read as the encryption keys had been lost. He was part of a team led by Professor Christof Paar, from the Ruhr University Bochum, Germany. The team successfully decrypted those historical messages [66].

6.1 Background

The ADFGVX cipher was designed by Fritz Nebel and introduced by the German Army in June 1918, as an extension to the ADFGX cipher, introduced three months earlier. The ADFGVX cipher was the most advanced field cipher of its time, as it combined fractionation, substitution, and transposition. The story of its successful codebreaking by the French cryptanalyst Georges Painvin (1886–1980) is well-known [1, 67]. The decipherment of one of the messages, known as "Le Radiogramme de la Victoire", provided the Entente allies with critical details about an upcoming major German offensive at the beginning of June 1918.

Less known, however, is the use of the ADFGVX cipher in the Eastern Front, for strategic communications between Berlin and several outposts in the East such as Romania, the Turkish Empire, and the German Military Mission in the Caucase. This ADFGVX traffic was intercepted by British and American listening posts from July to December 1918. It was partially decrypted by James Rives Childs (1893-1987), an officer at the American Expeditionary Force G.2 A.6 section, who later wrote several reports on his work [68–71]. He also preserved logs of ADFGVX traffic intercepted between September and December 1918. The cryptograms were published by Aegean Press Park in 2000, after Childs's death, as an appendix to Childs's book about the ADFGVX cipher [68]. The book contains 460 intercepted German radio messages, some sent as multipart cryptograms, with a total of 668 cryptograms. We were able to decrypt 618 (93%) of those 668 cryptograms with our new methods, and transcribed them into a readable German version.

From the historical point of view, this collection of German ADFGVX cryptograms provides a unique insight into Allied Signals Intelligence (SIGINT) and cryptanalytic capabilities towards the end of the war [68]. According to David Kahn, the five US interception posts alone "snatched 72,688 German messages from the airwaves" from the fall of 1917 until the end of the conflict [1, p. 334]. Unfortunately, most of the intercepted material has not been preserved. In his study on the British Army and Signals Intelligence during WWI, the Canadian historian John Ferris mentions that the British Army destroyed "all but a handful of its original intelligence files" [72, p. 24]. For instance, only 25 files recording the activities of British codebreakers operating in France have survived, out of 3330. Similarly, the German Imperial Navy destroyed nearly all

of its original documents for that period. German Army records are also very scarce, due to extensive destruction during and after WWII.

The analysis of the East Front ADFGVX messages poses major challenges for modern researchers, in addition to the technical cryptanalytic challenges. In his study Ferris quotes the British intelligence branch in Iraq complaining that "the mass of information received by any G.H.Q. almost came in disconnected fragments of very varying value ... fragments were the general rule, and the bulk of intelligence work at any G.H.Q really consists in putting together a gigantic jigsaw puzzle from innumerable little bits, good, bad and indifferent" [72, p. 24]. Wartime intelligence work – cryptanalysis and analysis – is usually done under high time pressure, and its outcome may have serious consequences, which is not the case for modern research. But despite extensive prior research about WWI(1914-1918) and the availability of recently declassified documents, we still needed to face several major and often similar challenges. Most of WWI historical research does not cover the period after November 11, 1918, after the German delegation signed an armistice with the Entente in Compiègne, while a large part of the ADFGVX traffic is from that period. Furthermore, most of the prior historical research focuses on the Western Front, as developments on this front eventually decided the outcome of the war. Historians often refer to the Eastern Front as the "forgotten front".

Childs's collection of ADFGVX messages is just a small part of a another "gigantic jigsaw puzzle". The interception logs preserved by Childs are organized chronically by their transmission or interception time, rather than by topic. From the geographical perspective they cover a vast area from the Baltic Sea to the Ottoman Empire, including Palestine and Northern Iraq, Eastern Europe (especially Romania), the Black Sea area, and Georgia. The messages deal with tactical movements of German troops in the Caucasus, Romania and the Ottoman Empire, with technical, logistical, and communications matters, as well as with the political and strategic situation in Germany and in its outposts. Each topic requires intensive research, including the review of historical archives, to establish the historical context and reconstruct the meaning and implications of the various messages.

After we deciphered the messages, our next task was to reconstruct their structure and order, and to collate multipart cryptograms into complete messages. After that, we tried to reconstruct a readable German text for each message, cleaning up garbles. Next, we made an effort to identify the callsigns, acronyms, persons, and entities involved in the communications. This was an iterative process as the interpretation of callsigns often changed as more messages were analyzed. The last step was to bring the contents of the messages within their historical context. This required a thorough review of the history of the war on the Eastern Front, of the history of Germany's communications systems and procedures, and of German intelligence organizations. The latter topics are still limited to experts and mainly focused on one unit, Abteilung IIIb [73, p. 25–54]. For all those reasons the historical part of this work cannot claim completeness.

6.2 Description of the ADFGVX Cipher

The cipher is named after the six Morse symbols, A, D, F, G, V, and X, the only symbols used when transmitting ADFGVX messages. Those symbols were deliberately chosen in order to reduce reception errors, as they sound different when transmitted in Morse (see Figure 6.1 for the Morse symbols). An earlier version of the cipher, the ADFGX, only used five symbols. We describe here the working principle of the cipher and an analysis of the size of its keyspace.

A:	.-	D:	-..	F:	..-.
G:	--.	V:	...-	X:	-..-

FIGURE 6.1: ADFGVX – Morse symbols

	A	D	F	G	V	X
A	P	R	M	Y	U	N
D	3	L	Z	G	E	S
F	8	C	7	1	Q	O
G	V	2	9	I	T	B
V	4	0	6	K	X	H
X	5	A	J	N	D	F

FIGURE 6.2: ADFGVX – Polybius square for Eastern Front key of November 7-9, 1918

6.2.1 Working Principle of the ADFGVX Cipher

Encryption consists of two phases, the substitution phase, and the transposition phase. We illustrate the process using an original ADFGVX message from November 1918, with the following plaintext:

```
FUERxLEGATIONxALLEMANDExKONSTANTINOPELx
```

In the substitution phase, we replace each letter or digit of the plaintext with a pair of ADFGVX symbols, according to a Polybius square. In the early version of the cipher, ADFGX, the size of the square was $5 \cdot 5 = 25$, in which all the letters of the alphabet appeared, except for the letter J which was replaced by the letter I. The main drawback of this early version was the need to spell out numbers in words, resulting in longer messages and more frequent repetitions. This issue was addressed in the ADFGVX cipher, by adding V as the 6th symbol. The Polybius $6 \cdot 6 = 36$ square for ADFGVX allows the encryption of all 26 letters of the alphabet, as well as the ten digits, 0 to 9.

In our example, we use the key in effect in the Eastern Front from November 7 to November 9, 1918. The Polybius square for this key is shown in Figure 6.2.

We start with the first plaintext letter, F. We first locate F in the square and it is at the intersection of the X row and of the X column. We, therefore, replace it with the pair XX. Similarly, the second plaintext letter U is replaced by AV, the third letter E by DV, and so on. Note that in each pair, the first ADFGVX symbol represents the row, and the second one represents the column. We obtain the following interim text:

```
XX AV DV AD VV DD DV DG XD GV GG FX XG VV XD DD DD DV AF XD XG XV DV VV VG FX XG
DX GV XD XG GV GG XG FX AA DV DD VV
```

Next we apply a columnar transposition to the resulting interim text. In our example, the transposition key in effect during November 7-9, 1918, is as follows:

```
6, 12, 7, 15, 1, 11, 16, 5, 8, 14, 3, 18, 9, 13, 2, 17, 20, 10, 19, 4
```

6	12	7	15	1	11	16	5	8	14	3	18	9	13	2	17	20	10	19	4
X	X	A	V	D	V	A	D	V	V	D	D	D	V	D	G	X	D	G	V
G	G	F	X	X	G	V	V	X	D	D	D	D	D	D	V	A	F	X	D
X	G	X	V	D	V	V	V	V	G	F	X	X	G	D	X	G	V	X	D
X	G	G	V	G	G	X	G	F	X	A	A	D	V	D	F	V	V		

FIGURE 6.3: ADFGVX – interim transposition rectangle

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
D	D	D	V	D	X	A	V	D	D	V	X	V	V	V	A	G	D	G	X
X	D	D	D	V	G	F	X	D	F	G	G	D	D	X	V	V	D	X	A
D	D	F	D	V	X	X	V	X	V	V	G	G	G	V	V	X	X	X	G
G	D	A		G	X	G	F	D	V	G	G	V	X	V	X	F	A		V

FIGURE 6.4: ADFGVX – ciphertext transposition rectangle

This key has 20 elements. We first inscribe the interim text, obtained after substitution, into an *interim (transposition) rectangle*, row by row. On top of the interim rectangle, we inscribe the transposition key. We obtain the interim rectangle shown in Figure 6.3. Note that the last row is incomplete, and has only 18 elements.

We now reposition or transpose the columns, according to the transposition key. The first column in the rectangle is repositioned at column 6, the second column to column 12, and so on. After transposing all the columns, we obtain the *ciphertext rectangle* shown in Figure 6.4.

Next, we extract the ciphertext from the ciphertext rectangle, this time column-by-column, starting with DXDG, then DDDD, then DDFA, then VDD, then DVVG, and so on. Note that column 4 is incomplete, so we extracted only 3 symbols. Finally, we obtain the following ciphertext:

```
DX DG DD DD DD FA VD DD VV GX GX XA FX GV XV FD DX DD FV VV GV GX GG GV DG VV DG
XV XV VA VV XG VX DD DX AG XX XA GV
```

After transposition, the two ADFGVX symbols which represent a plaintext letter or digit will most likely not appear next to each other in the ciphertext. This effect is called *fractionation*. Fractionation significantly increases the security of the cipher.

To decrypt a ciphertext, we apply those steps in reverse. We prepare a blank interim rectangle, only marking the transposition key on top of it. Since the length of the ciphertext is known, we can identify which of the cells in the last row, on the right side or that row, will remain empty. We now create a blank ciphertext rectangle and mark the corresponding empty cells. Next, we copy the ciphertext into the ciphertext rectangle, column-by-column. We then apply the inverse transposition on the columns of the ciphertext rectangle, to obtain the interim rectangle. For example, column 1 in the ciphertext rectangle is repositioned to be column 5 in the interim rectangle. Next, we extract the interim text from the interim rectangle, row-by-row. Finally, we divide that interim text into pairs of ADFGVX symbols, and replace each pair with the corresponding plaintext letter or digit, according to the substitution Polybius square.

6.2.2 Analysis of the Keyspace Size

An ADFGVX key consists of two elements, the substitution Polybius square, and the transposition key. The number of possible unique substitution keys is 36!, and the number of unique transposition key varies according to the length of the key. Based on an analysis of the keys recovered, the shortest transposition key had 16 elements, and the longest had 23. Therefore, the size of the ADFGVX keyspace is as follows:

$$36! \cdot \sum_{n=16}^{23} n! \approx 10^{64} \approx 2^{213} \tag{6.1}$$

For ADFGX, the size of the keyspace is as follows:

$$25! \cdot \sum_{n=16}^{23} n! \approx 10^{48} \approx 2^{158} \tag{6.2}$$

6.3 Related Work – Prior Cryptanalysis

In this section, we provide an overview of the historical cryptanalytic methods, as well as modern approaches. The first break into the early version ADFGVX cipher, the ADFGX cipher, was achieved in the spring of 1918 by Georges Painvin, his method addressing only the special cases of messages with similar beginnings or similar endings [1, 67]. Shortly after the war, towards the end of 1918, James Rives Childs from the US codebreaking unit G.2 A.6 was able to develop a more general solution [68–71]. Marcel Givierge, the head of the French Bureau du Chiffre, independently developed a very similar method in the early 1920s. In recent years, additional methods have been proposed for the cryptanalysis of ADFGVX/ADFGX. We present here an overview of the historical methods, as well as of the modern approaches.

6.3.1 Painvin's Methods – Spring 1918

Georges Painvin worked under great pressure, while the German Army launched their 1918 Spring Offensive, to break into the new ADFGX cipher, and into the ADFGVX cipher introduced shortly after. His first achievement was the identification of the details of the system and of its mechanism. On April 6, 1918, Painvin's was able for the first time to recover a daily key and decipher messages encrypted with this key. He achieved that by taking advantage of stereotyped messages with similar endings or similar beginnings, encrypted using the same key.

For illustration purposes, we shall assume two messages have the same ending, and have been encrypted with the same key. Since the two messages share the same ending, their interim text, after the substitution phase (see Section 6.2) will also share the same ending segment of the interim text. We now look at the interim transposition rectangles, for each one of the two messages. The similar ending segment occupies the last rows of each rectangle. The two rectangles have the same number of columns (also equal to the key length). Therefore, the last fragment of each one of the columns of the first rectangle will be identical to the last fragment of one of the columns in the second rectangle. Transposition preserves the continuity of columns, and those similar fragments at the end of the columns are also preserved in the ciphertext rectangles,

after transposition. For each column in the first ciphertext rectangle, there will therefore be one column in the second ciphertext rectangle, which ends with the same fragment. When extracting the ciphertexts, column-by-column, from the ciphertext rectangles, those identical fragments are preserved and will appear in the final ciphertexts.

A cryptanalyst looking at the two ciphertexts, and suspecting that the original plaintexts have the same ending, will be able to identify those identical fragments of text which appear in both ciphertexts. By analyzing the positions of the matching fragments in the two ciphertext rectangles, the analyst is able to deduce the transposition key. As soon as the transposition key is recovered, and the transposition undone, he just needs to solve a simple substitution cipher, using both messages. The method is described in detail in [74]. A very similar method, for the case of similar beginnings, is also described.

With this method, Painvin was able to recover nine daily keys used in the Western Front in 1918 and a tenth key was recovered by British codebreakers [68–71]. While those ten keys cover only a fraction of the daily keys used in the Western Front in the spring and summer of 1918, those ten keys were used to encrypt 50% of all the traffic during that period [68–71]. This can be explained by the fact that the availability of large numbers of messages on a given day provided Painvin and his team for more opportunities to find pairs of messages with similar endings or beginnings. In addition, increased traffic by itself was often a sign of a upcoming German offensive planned for a few days later.

6.3.2 Childs's Method – End of 1918

When working on the decryption of Eastern Front traffic, Childs initially applied Painvin's methods for the special cases of messages with similar endings or similar beginnings. Only after the end of the war Childs was able to develop a more general solution for the recovery of ADFGVX keys. This method was originally presented by Childs in [68], and reproduced with more details by Friedman, Kullback, Rowlett, and Sinkov in [74] and [75].

At the heart of the method is the fact that the frequencies of the symbols A, D, F, G, V and X, when used as the left component of a pair (see Section 6.2) at the substitution phase, are very different from their frequencies when appearing as right components of a pair. Friedman provides an example of a substitution square, with the relative frequencies of each letter or digit (relative to a total of 1 000), as shown in Figure 6.5. In the rightmost column (see (1) in Figure 6.5), the frequencies of the 6 symbols when used as the left component of a pair are shown. In the bottom row (see (2) in Figure 6.5, the frequencies of the 6 symbols when used as the right component of a pair are shown. With this substitution square, A is much more likely to appear as the left component of a pair than as the right component. Similarly, X is much more likely to appear as the right component of a pair than as the left component.

Childs's method is applied to a series of messages encrypted with the same key. In the example given by Childs and Friedman, the method is illustrated on a series of twelve cryptograms of various lengths, with 108 to 254 ADFGVX symbols each, 2 312 in total. The transposition key has 17 elements.

The process is iterative. At each step, the analyst analyzes some of the cryptograms, tries to separate symbols from the ciphertexts into two distinct classes or groups, either as left/first components of pairs, or as right/second components of pairs. For each group, he computes the relative frequencies of the A, D, F, G, V, and X symbols, and tries to classify sequences of additional symbols, appearing in the same cryptograms or in new cryptograms.

	A	D	F	G	V	X	(1)
A	T 92	H 34		E 130		F 29	284
D		L 36	O 75	W 16	R 76	S 61	264
F	A 74		B 10		M 25	I 74	183
G		N 79	P 27	G 16		C 31	153
V		D 42		J 2		K 3	47
X	Q 3	U 26	V 15	X 5	Y 19	Z 1	69
(2)	169	217	127	169	120	198	1000

FIGURE 6.5: ADFGVX – frequencies of symbols as right(1) or left(2) in a pair

At first, the analyst determines whether the length of the encryption key is odd or even, by classifying the initial symbols of all the cryptograms. After that, he uses cryptograms having the same length, to classify additional symbols in those cryptograms as left or right components. Next, he uses cryptograms having the same number of elements in the last row of their transposition rectangles, to classify more symbols in those cryptograms. After having classified enough symbols in several cryptograms, the analyst is able to draw conclusions about the original positions, in the interim transposition rectangles, of columns – after transposition – in the ciphertext transposition rectangles. Using an iterative process, he is finally able to recover the full transposition key.

While termed as a "general solution", this method relies on several factors, that may not be present in all cases. It requires a certain amount of cryptograms all encrypted with the same key (probably 12 or more) as well as several cryptograms having the same length. It also relies on the frequency statistics of the ADFGVX symbols being different for the first (left) elements and the right(second) elements of the pairs, as in Figure 6.5. In [74], an example of a substitution key engineered to hide those differences is presented. This was achieved by moving around the letters in the substitution Polybius square so that the sums of all columns and all rows are similar. In such a case, the method described here would fail.

Interestingly enough, a very similar method is described by Marcel Givierge, the head of the French Bureau du Chiffre [76]. Childs claims to have developed his method independently of Givierge's and that he did not have access at the time to the Givierge's work [68].

In [74], a solution for the much simpler special case of a complete transposition rectangle is also described. A complete transposition rectangle is obtained when the length of the ciphertext is an exact multiple of the key length.

6.3.3 Konheim – 1985

Konheim's method is presented in an extended abstract [77] as well as in an appendix to Childs's book [68]. It also relies on uneven frequencies of letters in any language, also reflected in the substitution Polybius Square (see Section 6.2). First, using a statistical analysis of potential bigrams generated by juxtaposing columns of ciphertext symbols, Konheim detects whether any

two columns were adjacent in the original (interim) transposition rectangle. Such bigrams or pairs of ADFGVX symbols are likely to display the frequency characteristics of a monoalphabetic substitution. Next, the substitution must be solved using "standard techniques", which are not detailed in the paper. Last, after solving the substitution, the transposition key may be recovered.

This algorithm is demonstrated on a single and rather long message, with 1 672 plaintext characters. The ciphertext length has therefore 3 344 ADFGVX symbols. The transposition key is rather short and has only 6 elements. This means that the rows of the transposition rectangle have 3 344/6 = 509 elements. For comparison, transposition keys used in 1918 had between 15 to 23 elements, and messages had on average 215 ADFGVX symbols, and each row in the transposition rectangle usually had no more than 5 to 15 elements.

Furthermore, the statistical methods used for the first step of Konheim's algorithm are rather complex, with many special subcases. Those methods rely on having exceptionally long rows in order to produce statistically significant results. Another caveat of Konheim's method is that he does not specify which "standard techniques" may be applied to solve the substitution before the transposition key has been recovered. The techniques commonly used for solving a monoalphabetic substitution cipher are applicable only if the symbols are in place and not transposed.

6.3.4 Bauer – 2013

Bauer [2] demonstrates a method based on the analysis of the probable locations of high-frequency short words such as "THE". This method is demonstrated on an ADFGX message with 680 symbols, and a transposition key of length 20. As 680 is a multiple of 20, this is a special case of a complete transposition rectangle.

First, columns are matched into pairs of adjacent columns, so that one column contains the left symbols of the original interim text pairs of ADFGX symbols, and the second column contains the corresponding right symbols. Next, the order of those pairs of columns is reconstructed, by guessing the pairs of symbols most likely to represent the high-frequency letters "E" and "T", and trying to identify possible occurrences of the word "THE". This allows for the recovery of the transposition key. Finally, the substitution is solved.

6.4 New Ciphertext-Only Attack

The algorithm we developed is based on hill climbing, and processes a batch of cryptograms believed to have been encrypted using the same key, rather than trying to decipher single messages. Using a divide-and-conquer approach, the algorithm first tries to recover the transposition key. When the transposition key has been recovered, we are left with a simple substitution cipher, which can be solved using a variety of methods [64, 74, 78–84]. We also used a hill climbing algorithm to solve the substitution key. We describe here in more details the algorithm to recover the transposition key.

As mentioned in Section 6.2, the transposition part of ADFGVX is implemented using the classical columnar transposition cipher method (see Section 5.1.1). However, we cannot simply apply the algorithm described in Chapter 5, since ADFGVX also includes a substitution phase (each plaintext letter or digit being converted into a pair of ADFGVX symbols). Furthermore,

after transposition, the two symbols which composed an original plaintext letter (or digit) are most likely to appear apart from each other in the ciphertext, after transposition, and recovering the transposition key is even more challenging. This effect is also known as fractionation, and was deliberately introduced into the ADFGX and ADFGVX cipher methods. The inventors of the ADFGVX cipher were aware of methods to solve columnar transposition ciphers and simple substitution ciphers, but believed that the combination of a transposition cipher and a substitution cipher, with fractionation, would create a much more secure type of hand cipher.

To recover the transposition cipher, we implemented an algorithm along the lines of the methodology described in Chapter 4. First, we assume the length of the transposition key, K_{tr}, is known. If we don't, we just test all possible lengths from 15 to 25. The algorithm implements hill climbing. It is listed in Algorithm 6.

Algorithm 6 ADFGVX – hill climbing algorithm to recover the transposition key

```
1:  procedure HillClimbingTransposition(C, N)          ▷ C = ciphertexts, N = rounds
2:      BestGlobalTr ← null                            ▷ Best global transposition key
3:      for I = 1 to N do
4:          BestTr ← BestRandomTr(10000)               ▷ Best tr. key for round
5:          repeat
6:              Stuck ← true
7:              for CandidateTr ∈ Neighbors(BestTr) do  ▷ Iterate over all neighbors
8:                  if IC_pairs(TrInv(CandidateTr,C)) > IC_pairs(TrInv(BestTr,C)) then
9:                      BestTr ← CandidateTr           ▷ Better key for round
10:                     if IC_pairs(TrInv(BestTr,C)) > IC_pairs(TrInv(BestGlobalTr,C)) then
11:                         BestGlobalTr ← BestTr      ▷ Better global key
12:                     Stuck ← false
13:                     break
14:         until Stuck = true
15:     return BestGlobalTr
```

Each round starts with a preliminary phase, $BestRandomTr(10000)$, which select the best key from 10 000 random keys.

For the preliminary phase, and for hill climbing, the algorithm uses a specialized scoring method IC_{pairs}, which is based on the Index of Coincidence (see Section 3.2.3), but unlike the standard IC, this is not computed from monograms. Instead, for each candidate transposition key $CandidateTr$, we apply the inverse of that transposition key ("undo the transposition") to the ciphertext C, that is, $TrInv(CandidateTr,C)$. We then divide the resulting sequence of ADFGVX symbols into pairs (of adjacent symbols), and compute their index of coincidence, that is, $IC_{pairs}()$. Note that in case the candidate key is equal to the correct original encryption key ($CandidateTr = K_{tr}$), each pair in the sequence after undoing the transposition represents a single original plaintext symbol (A-Z, 0–9) **after substitution**. As the (monogram) IC of a plaintext P encrypted with a monoalphabetic substitution is always equal to the IC of the plaintext, in case of a correct candidate transposition key ($CandidateTr = K_{tr}$), we have:

$$IC_{pairs}(TrInv(CandidateTr,C)) = IC(P) \tag{6.3}$$

This relationship is the key to the implementation of this divide-and-conquer attack, using IC_{pairs}, which allows for the recovery of the transposition key while ignoring the substitution key. During encryption, the substitution phase of ADFGVX maps plaintext symbols into pairs

of ADFGVX symbols. The transposition phase tears apart those pairs of symbols, creating more random combinations. The more the algorithm is able to reposition the elements of those original pairs, using a partially correct candidate transposition key the higher the value IC_{pairs} will be. The highest value of IC_{pairs} is expected when the candidate transposition key is correct.

We implemented a set of transformations used at each iteration, and applied on the current transposition key ($NeighborTrKeys(BestTr)$). Those transformations consist of:

1. Simple swaps of any two elements in the transposition key.
2. Swaps of any two segments (or various lengths) of consecutive elements in the key.
3. Rotating (cyclically) a segment of consecutive elements in the key (at any position, and of various length).
4. Reversing the transposition key, so that the last element becomes the first, and so on.
5. Swapping the elements of every pair in the key. In the case of a key with an odd length, we just skip the last element of the key.

The first three types – single swaps, and segment transformations, are non-disruptive. As for the case of the columnar transposition cipher (Section 5.3.2), the segment transformations are mandatory, and our algorithm does not succeed if we remove them. The last two types were empirically discovered. Before we introduced them, hill climbing would often get stuck on keys with a high score, which were similar to the correct key but with the key elements reversed, or with every two consecutive elements reversed.

This algorithm performs well for transposition keys with an odd length. In the case of transposition keys with an even length, it will often fail. To illustrate the problem, we consider the following transposition key with 8 elements (even length): $K_{tr} = (\underline{4,6},\underline{2,3},\underline{7,5},\underline{1,8})$, and some substitution key, K_{sub}. We encrypt a plaintext P, first applying K_{sub} and obtaining a sequence of ADFGVX symbols, where each pair represents an original plaintext symbol. We then apply K_{tr} and obtain the ciphertext C. We now apply our hill climbing algorithm on that ciphertext, adn assume we know the transposition key length, $|K| = 8$. At some stage of hill climbing, we may obtain the following candidate key: $CandidateTr = (\underline{7,5},\underline{4,6},\underline{1,8},\underline{2,3})$. Note that in this candidate key, the original **pairs** of key elements, e.g. (4, 6) or (2, 3), have been correctly reconstructed, but the relative **order of the pairs** is wrong.

Due to the nature of columnar transposition, if we apply the inverse of $CandidateTr$, that is $TrInv(CandidateTr, C)$, we will obtain a sequence of ADFGVX symbols in which the two elements of each one of the original pairs of ADFGVX (after substitution, but before transposition), are now next to each other. As a result, $IC_{pairs}(TrInv(CandidateTr, C)) = IC(P)$, although $CandidateTr! = K_{tr}$. In other words, we have a spurious high IC_{pairs}. As a result of this phenomena, we cannot rely only on IC_{pairs} as the scoring function, to recover a transposition key with an even length. When the length of the key is odd, this phenomenon does not occur.

To cope with transposition keys with an even length, we adapted the scoring function, so that it combines the previous measure (IC_{pairs}) with the IC of quadruplets (IC_{quads}). We divide the sequence $TrInv(CandidateTr, C)$(obtained after undoing the transposition) into **quadruplets** of ADFGVX symbols, and compute their IC. Note that if the candidate key is the correct key, that is $CandidateTr = K_{tr}$, each quadruplet of ADFGVX symbols (after undoing the transposition)

Transposition key length	Minimum number of cryptograms (n)	Total number of ADFGVX symbols	Average cryptogram length
16	2	486	243.0
16	2	388	194.0
16	2	402	201.0
17	3	560	186.7
18	5	1000	200.0
19	4	676	169.0
19	2	500	250.0
20	5	770	154.0
20	2	534	267.0
20	4	686	171.5
22	4	926	231.5
22	4	766	191.5
22	3	848	282.7
23	2	566	283.0

TABLE 6.1: ADFGVX – performance

therefore represents a bigram of plaintext symbols (before substitution). After modifying the algorithm with this improved scoring method, the algorithm performed well also for transposition keys with an even length.

To evaluate the performance of the algorithm, we performed the following experiment. After dividing the cryptograms into groups according to the keys they were encrypted with, we ran the algorithm on the first n cryptograms of each group, starting from $n = 10$, and each time reducing n until the algorithm failed to recover the key. Rather than selecting the longest or cleanest cryptograms of each group, we just selected the first n cryptograms of each group. Therefore, the n cryptograms selected from each group (key) may vary in terms of length or number of transcription errors. In Table 6.1, we show the minimum number of n cryptograms required by the algorithm to allow for full key recovery. It can be seen that for 6 keys out of 14, only 2 cryptograms with a total of 388 to 566 ADFGVX symbols were enough to recover transposition keys of length varying from 16 to 23. The worst case was with a key of length 18 which required 5 messages, with a total of 1 000 ADFGVX symbols. In contrast, Childs's method was demonstrated on a set of 12 cryptograms with a total of 2 312 ADFGVX symbols, for a transposition key of length 17. Konheim's method was illustrated on a single cryptogram with 3 344 symbols and a very short key with only 6 elements.

6.5 Deciphering Eastern Front ADFGVX Messages

In this section, we describe the structure and format of the cryptograms. We also describe the step-by-step process used to recover all the encryption keys and to decrypt the messages.

6.5.1 The Messages

The messages were published by Aegean Park Press in 2000, and appear in Appendix B of Childs's General Solution of the ADFGVX Cipher System [68]. Appendix C of the book also contains some of the keys for the messages. According to the editor of the book, the messages were provided by Childs twenty years before the publication of the book. The messages appear in the form of interception logs. At the beginning of each page, a header like following is shown:

```
GENERAL HEADQUARTERS, AMERICAN EXPEDITIONARY FORCES
  GENERAL STAFF, SECOND SECTION (G.2 A.6) (alp)

      ''RICHI'' ADFGVX CIPHER

  (Employed between Berlin and the Black Sea)

   Messages Intercepted by British Stations.
```

Cryptograms from the same date usually appear consecutively. ADFGVX keys for the Eastern Front were in effect for three days. We found, however, a large number of messages included in a day's log but encrypted with unrelated keys. Accordingly to German communications procedures, the maximum length of a single cryptogram should not exceed 250 ADFGVX symbols. Longer messages were to be split into shorter parts – "Teile" in German, often abbreviated as *TL*. Note that this rule was not always followed, and some of the cryptograms are actually longer, some longer than 400 symbols. Each message, composed of one or more parts (Teile), had a header sent in cleartext, as in the following example:

```
POT, COS v NKJ (3.10 a.m., October 7th)
FÜR COS DEUTSCHE DELEGATO WIRTCHI ABTLG 2305 (6) 2 TLE
```

The first line contains the callsigns of the receiving stations – POT and COS in this example – followed by a "v" which abbreviates the German word "von" (from) indicating the sender, in this case NKJ. On this line the operators of the Allied interception station usually added some information, such as the date and time of reception. The second line usually contains (in cleartext) the name or location of the recipient of the message. In this example the recipient is the Economic Department of the German Delegation at Tiflis (COS), the capital of Georgia. "2305" refers to the transmission hour (11:05 p.m.) and "(6)" to the transmission day. "2 TLE" indicates that the message consists of two parts (Teile).

Each part of a multipart message consists of a separate cryptogram, with its own header. An example of such a header is "1 TL RICHI-322" which indicates that this is the first part of the message, it is encrypted using the "RICHI" (codename for ADFGVX as used in the Eastern Front) keys, and this first part consists of 322 ADFGVX symbols. Next, the cryptogram is written in groups of 5 ADFGVX symbols. The second part starts with the header "2 TL RICHI-256", and accordingly, has 256 symbols. From the cryptographic perspective, the most important items are the cryptogram length and the transmission time, which are both sent in clear, as well as the interception time, logged by the operators of the listening station. Also, the fact that the ADFGVX symbols are transcribed in groups of 5 symbols is often useful, in trying to isolate reception or transcription problems.

In total, there are approximately 460 ADFGVX messages, many of which are composed of multiple ADFGVX cryptograms. In total, there are about 668 ADFGVX cryptograms, if we exclude repetitions of identical messages. The earliest message is from September 19, 1918, and the last message from December 7, 1918. The average length of a cryptogram is 215. About 7% are shorter than 100 symbols, 80% between 100 and 300, 13% longer than 300. The shortest cryptogram has 36 symbols and the longest has 500 (13 have more than 400 symbols).

About one third of the cryptograms have either an incomplete transcription or have transcription errors. This does not include garbled letters which can be observed only after decryption. In fact, after decrypting the cryptograms, we found that almost all of them had garbled letters. The most frequent transcription errors, which are visible even before trying to decrypt a cryptogram, include:

- Symbols transcribed as a hyphen ("-"). The interception operators were instructed not to guess Morse symbols they were not sure about, so they marked them instead with hyphen signs. In case of severe interference, whole groups of 5 symbols each were marked as hyphens.
- In certain groups, only 3 or 4 of the symbols are transcribed, instead of the expected 5 symbols. In other groups, 6 symbols are transcribed.
- In some cases, the transcription includes wrong ADFGVX symbols, such as U or B.
- In some cryptograms, major segments, usually the beginning or ending of a cryptogram, are missing, and instead the operator wrote "jammed" or "interference".
- The number of symbols transcribed does not match the length information sent in clear.
- Sometimes the length information sent in clear is an odd number. This is clearly an error, as such a case may not happen in a properly encrypted ADFGVX message.

In Section 6.5.2, we describe how we addressed those errors.

6.5.2 Recovering the Keys

The first step in the process was to scan the cryptograms from the book. Next, we applied OCR, using the open-source OCR package Tesseract [85], and limiting the set of valid characters to A,D,F,G,V,X, and the hyphen sign ("-"). We then compared OCR results with the original cryptograms, corrected OCR errors, and removed portions of text which were not part of the cryptograms, such as clear text.

The next step was to divide the cryptograms into groups of cryptograms likely to have been intercepted on the same day. At this stage we simply ignored all the cryptograms with transcription errors, and used the two thirds of the cryptograms without such errors. For each daily group, we applied our algorithm described in Section 6.4, testing all key lengths from 15 to 25. With this method, we were able to recover the keys for most of the days. As expected, the same keys were in effect for 3 consecutive days. With those keys, most of the cryptograms from a given day could be decrypted, but some could not. We suspected that some cryptograms intercepted on the same day did not belong to the same key. Other cryptograms had severe errors, either incorrectly encrypted, incorrectly transmitted, or wrongly intercepted or transcribed. However,

this did not prevent our algorithm from recovering most of the keys, as it performs well even if there is a large number of garbled symbols. It is also robust enough to perform even if up to 20% of the cryptograms do not belong to the same key.

Messages for November 1, 2, and 3 constitute a special case. When trying to apply cryptanalysis on all messages from those days, the key could not be successfully recovered. We suspected that there might be more than one key involved. We applied the algorithm again on smaller subsets of cryptograms from those days, trying various combinations of cryptograms. We were able to recover two different keys for those days, with some cryptograms decrypting on one key, and others on the second key.

Next, we created a database of all those keys, and tried again to decrypt each one of the cryptograms, trying each one of the keys. We were able to obtain correct decryptions for almost all of the cryptograms which had no transcription errors, about 400 in total. It turned out that a large number of cryptograms were wrongly placed in the logs, and could be decrypted using the key from another day.

6.5.3 Handling Errors

Next, we processed the cryptograms with transcription errors, or those that could not yet be decrypted with any of the keys, about 250 in total. For those messages which had hyphens instead of symbols, we used the procedure described here. In ADFGVX, every plaintext character is replaced by a pair of ADFGVX symbols, before transposition. If only one of the symbols has been correctly transcribed, and the other marked as a hyphen, there are only 6 options for that missing symbol. Therefore, there are only 6 possible options for the original plaintext character, out of the 36 possible options (a to z, 0 to 9). Following is an example of a message transcribed with two hyphens. The two plaintext letters affected are marked with the "!" sign. For both positions, all the 6 possible original plaintext characters are shown. By looking at the missing letter in the context of the surrounding words, the correct characters, S and G, can be recovered.

```
              !                 !
NACHRCWEFxCONSTANTZATEITHEUTEx400AUSSERBETRIEBFESETZTxMILMISS
              C                 U
              M                 z
              S                 y
              4                 G
              N                 4
```

About 110 cryptograms had minor transcription errors. Those with few hyphen signs could be easily corrected in most cases using the procedure described above. A similar procedure was also applied to cryptograms having groups with fewer than 5 symbols. For example, if a 5-symbol group was transcribed as DGVX, then we applied the same procedure by checking 5 cases, -DGVX, D-GVX, DG-VX, DGV-X and DGVX-. For a group with more than 5 symbols transcribed, e.g. GDFAXV, we simply tested 6 cases, each time removing one of the 6 letters and selecting the most probable resulting plaintext.

In addition, about 34 cryptograms had a correct length but a high number of hyphens. In most cases, only some of the missing plaintext characters could be recovered using this procedure. Still, large parts of the cryptograms could be read.

For those cryptograms for which the number of symbols transcribed exceeded the length specified in clear, we simply removed an equivalent number of extra symbols, from the beginning of the cryptogram, or from its end. If this did not work, we tested every possible location in the cryptogram. For those cryptograms for which not enough symbols were transcribed, compared to the length sent in clear, the procedure was more complex. We added groups of hyphens at various positions in the cryptogram, and selected the position which resulted in the best plaintext. Then, we tried to recover some of the symbols marked as a hyphen, using the method described above. With those tedious and mostly manual procedures, we were able to recover 27 messages with wrong lengths. While usually the result of interception problems, such cryptograms with wrong lengths may also have been the result of erroneous encryption. In such cases, neither the German receiving side, nor the British who intercepted those messages, were able to decrypt them.

50 cryptograms could not be recovered, with any of the procedures described above. Out of those 50 cryptograms, 36 had too many groups missing, mostly due to interference, as also indicated by comments such as "jammed" or "interference" in the interception logs. The remaining 14 had a correct length but could not be decrypted using any of the keys. We suspect that most of those 14 cryptograms were probably wrongly encrypted.

6.5.4 The Final Keys

Next, we ran again the cryptanalysis program, on the 618 cryptograms we were able to recover, after dividing them according to their keys. We used the transpositions keys we had already obtained, and ran only the substitution solver part. The purpose was to obtain a more accurate substitution key, based on a larger number of cryptograms per key.

Finally, we compared the keys we recovered with the keys provided by Childs in the last part of his book [68]. The results are as follows:

- Eight of the fourteen transposition keys we recovered are identical to Childs's keys, with some minor differences in the substitution keys. Those differences are mainly due to the fact our algorithm is unable to determine the exact position of the digits 0 to 9 in the substitution Polybius square. This may be achieved via traffic analysis, rather than cryptanalysis. Traffic analysis, however, requires additional sources of intelligence, for example, to recognize unit numbers. In addition, for some of the substitution keys, Childs left as blanks the symbols he could not accurately identify.
- Two more keys were provided by Childs, but there were errors so that messages could not be decrypted with those keys. In one case, the error was in the transposition key (an element missing), and in the other case, the error was in the substitution square (two rows of the square swapped). Our program was able to recover the correct keys.
- We recovered four additional keys which do not appear in Childs's book. They refer to the period from November 19 to the beginning of December 1918.

The keys are shown Table 6.2.

Period	Transposition key (length) Substitution key	Recovered cryptograms
Sep 19 – 21	2,2,7,20,10,19,1,13,9,18,3,17,21,8,14,4,6,16,11,22,5,15 (22) “D5613Q9KBNO0HY8EISJUTZFCW7VPML2ARG4X”	15
Oct 4 – 6	4,13,3,14,1,16,9,15,5,19,10,18,6,17,7,20,11,21,8,12,22,2 (22) “YN87PJ3WRUCIEO1SKLZX0DFBH6MT9A2QV54G” *Note: Substitution key in Childs’s book is erroneous*	24
Oct 28 – 31	6,15,12,16,5,7,14,4,13,8,11,1,17,2,10,3,18,9 (18) “HI20SXRUWQY8EK7O619CBJAP453FDZTGLMVN”	33
Nov 1 – 3	3,16,4,15,7,12,18,6,17,8,19,1,13,10,2,14,11,9,5 (19) “UILOF9RCZVSX02G7QTD8WNB5JMHEKPY41A36”	100
Nov 4 – 6	7,10,8,14,3,11,16,1,6,13,4,9,15,5,12,17,2 (17) “17WHFLJ5D2UPEXKVZ9O0Q3Y6R8ABGITCMS4N” *Note: Transposition key in Childs’s book is erroneous*	106
Nov 7 – 9	6,12,7,15,1,11,16,5,8,14,3,18,9,13,2,17,20,10,19,4 (20) “ PRMYUW3LZGES8C71QOV29ITB40 KXH AJNDF”	93
Nov 10 – 12	9,12,7,11,3,8,16,6,14,2,10,15,5,13,1,4 (16) “4ARUT1OIFSKN3 BZPVLD JMXCWHQ2E G0 Y ”	46
Nov 13 – 15	13,8,6,16,7,18,1,14,9,20,10,15,17,2,3,11,5,19,4,12 (20) “JZLH R S T MKDWU V B P FAO GIX CNE”	13
Nov 13 – 15	4,11,5,14,9,7,16,1,12,15,6,10,3,13,8,2 (16) “H BMUF15PX0DJLR S6VONKZ AWITEGC ”	52
Nov 16 – 18	7,12,1,14,8,16,13,9,19,3,15,4,10,18,6,2,11,17,5 (19) “WG EITNHUB2R FDZJS PY VQL 1OAXMKC”	36
Nov 19 – 21	13,20,3,16,7,14,4,12,8,11,5,15,2,18,17,10,19,6,1,9 (20) “LC58QH7VI2YB9EURO60GX3MTFAKP1D4NJZSW” *Note: New key, does not appear in Childs’s book*	12
Nov 22 – 24	6,12,16,7,14,22,11,18,1,15,8,10,20,2,13,21,3,17,19,5,9,4 (22) “QNZ72XS4C0IJY3RBEKL9FD6GMTHUVWA5O8P1” *Note: New key, does not appear in Childs’s book*	26
Nov 25 – 28	21,9,6,14,10,20,1,16,18,7,15,4,11,22,5,17,23,2,12,8,19,3,13 (23) “HQ05DKZAOYM6BEIWTJ7PSCFLV94132NGURX8” *Note: New key, does not appear in Childs’s book*	33
Nov 28 – Dec 1	9,3,14,10,2,8,15,4,16,11,5,13,6,12,1,7 (16) “782GPY5OQHF91UDNI364TLVXEAR0JZBKMCSW” *Note: New key, does not appear in Childs’s book*	29
	Total:	618

TABLE 6.2: ADFGVX – list of keys used in the Eastern Front from September to December 1918

6.6 Historical Analysis

In this section, we present an overview of German cryptographic and cryptanalytic capabilities at the outset of the war, as a context for the conception of the ADFGX/ADFGVX ciphers, the most sophisticated of the field ciphers developed by Germany during WWI. We also provide a short profile of James Rives Childs, the US officer in charge of decrypting East Front ADFGVX traffic. Additionally, we focus on several topics covered by the decrypted messages:

- The political and military situation in Romania in 1918
- The effect of the Kiel mutiny in November 1918 on German communications
- Cryptic messages with a code within a code
- A message related to Karl Wilhelm Hagelin, the father of the cryptographer Boris Hagelin.

6.6.1 The German Military and Signals Intelligence

In "The Codebreakers", David Kahn writes: "German cryptology goose-stepped toward war with a top-heavy cryptography and no cryptanalysis" [1, p. 263]. In 1914 Germany's Army and Navy were well equipped in terms of wireless equipment, but less aware of the advantages and disadvantages of the very new radio technology. Their autocratic and rigid nature did not encourage critical thinking and initiative at the lower ranks. In its maneuvers before the war the Army had forbidden the radio personnel from intercepting wireless communications of the opposite forces. The Navy even went as so far as to punish a seaman who had informed his superiors that he had intercepted and solved an enciphered message sent between two German radio posts. The Kaiser's generals believed that they would win the next conflict with infantry, artillery, and cavalry as they did in the Franco-Prussian War of 1870-71. Communications security and codebreaking were not considered as important.

In addition, early-on in the war Germany suffered a major setback on the SIGINT front, with consequences that would prove to be decisive in the years to come. In August 1914, the Royal Navy cut the German telegraph undersea cables. Germany had to reorganize its communications and to rely mostly on wireless communications, exposing those communications to interception by the enemy. Furthermore, the French cryptographers had an advantage over their German counterparts, with a team of highly experienced cryptographers, who had been monitoring German Army traffic during peacetime German Army maneuvers. Their success in deciphering the main German Army code, the ÜBCHI code, played a critical role in the Western Front during the initial phase of the war.

When the war started, the German Army had no signals security discipline so that fast advancing troops often communicated in plaintext. Their strategic-level intelligence on the French and Russian armies was poor. The German Great General Staff used its military intelligence organization, the subsection Abteilung IIIb, only for the purpose of gathering tactical information. Germany had limited information on the British Army since only the Navy collected intelligence on the United Kingdom, focusing mainly on the Royal Navy. The Kriegsmarine had its own naval intelligence service, the Marinenachrichtendienst, which used direction finding and limited codebreaking to locate and identify the enemy's warships. The Foreign Office ran a separate intelligence network in its embassies and consulates, usually working from open sources. It had its own cipher bureau which did not cooperate with its military counterparts, and until 1918

the Foreign Office still considered its own diplomatic codebooks as unbreakable. While the War Ministry in Berlin was responsible for developing codes and ciphers for the German Army, there was no centralized cryptological institution responsible for analyzing their cryptographic security.

At the battle of Tannenberg against a more powerful Russian Army in 1914, general Paul von Hindenburg and his general quartermaster Ernst Ludendorff owed much of their victory to two amateur codebreakers, lieutenant Alexander Bauermeister and professor of philosophy Ludwig Deubner. Deubner and Bauermeister were attached to the High Army East Command and they collaborated with the cipher bureau of the Austro-Hungarian MILINT, the Evidenzbüro. Early organized Signals Intelligence (SIGINT) in the German Army remained confined to the East Front. Only later, when Hindenburg and Ludendorff headed the Supreme Army Command, the Oberste Heeresleitung (OHL), they started to integrate SIGINT at the strategic planning level. The Navy followed in February 1916 with its own signal intelligence service and opened an interception and codebreaking center in Neumünster. Until that time most codebreaking successes by the Germans were the result of either individual initiatives or of errors by their enemies.

A major reorganization occurred in the spring of 1917. The "Chief of Field Telegraphy", attached to the General Quarter Master, was renamed "Nachrichtenchef", which Childs translated as "Chief Signal Officer" [71, p. 41]. The Nachrichtenchef was responsible for all communications via phone, telegraph, wireless, and also for the use of codes and ciphers.[1]

At the strategic level, the General Chief of Staff of the Field Army, responsible for operational planning, retained its control over Abteilung IIIb. This department, in addition to classical military intelligence from SIGINT and other sources, also acted as a political police inside Germany and as a press censor. The evaluation of intelligence gathered from SIGINT about the enemy's armies went directly to the Nachrichtenabteilung (intelligence department), renamed Abteilung Fremde Heere (Foreign Armies Department) in 1917, and which had two sections, West and East.

According to former cryptologist Hermann Stützel the E-Stelle (or Entzifferungs-Stelle – Deciphering Station) at the OHL (Supreme Army Command) had three sections [86, p. 543]. The Department of Foreign Armies evaluated the intelligence collected by interception posts. They focused on the enemy's trench codes, direction finding, and reconstructing the enemy's order of battle. The second section dealt with developing field codes for the German Army. A third section monitored diplomatic traffic and worked on cracking diplomatic codes and ciphers. The latter was headed by Hermann Stützel. In 1918, Stützel took the initiative of monitoring traffic between the Foreign Ministry and its embassy at Madrid, and exposed the vulnerability of German diplomatic codes. Despite the scandal following the publication of the Zimmermann telegram that brought the USA into the war against the Central Powers in 1917, and despite Stützel's work, the Foreign Ministry kept its conviction that its codes were unbreakable. The Foreign Ministry believed that the Zimmermann message could only have been read by the enemy because they were helped by a traitor [1, p. 294].

Such confidence about the security of their codes was also shared by the German Army cryptographers. One of them was lieutenant Fritz Nebel who served in the Great General Staff at Spa. Nebel achieved the Abitur (German university entrance qualification), specializing in Greek and

[1] The German term "Nachrichten" is ambiguous. "Nachrichten" is used either in the context of intelligence, news, and information, or in the context of signals and communications. The intelligence officers of Abteilung IIIb, attached to the Army Corps, called themselves "Nachrichten-Offiziere", but often so did signal officers, who dealt mainly with military communication systems.

Latin, but when he joined a telegraph battalion, he had no prior cryptography training. Nevertheless, he was asked in early 1918 to create a new and highly secure field cipher, which Germany needed for its coming Spring Offensive. Nebel designed the ADFGX cipher, which he later extended by adding one more symbol (ADFGVX).

In the late 1960s Fritz Nebel met for the first time with Georges Painvin, his former French opponent, at a historical meeting of former cryptologists in Paris. On that occasion, Nebel was shocked when the French cryptanalyst described to him how he had cracked the ADFGVX cipher, including an intercepted German message in May of 1918 containing critical strategic information about German plans, the famous "Radiogramme de la Victoire". Painvin's success had enabled the Allies to stop the German offensive and to take back the initiative on the Western Front. Nebel later confessed that he was "neither a mathematician nor an engineer" when he developed the codes [67, p. 20]. His confession confirms Kahn's harsh verdict on German cryptology in WWI.

6.6.2 James Rives Childs and Allied Cryptanalysis

The French successes with German codes in 1914 were not a coincidence. Since their defeat in the 1870-71 Franco-Prussian War, the French military intelligence had worked hard on cracking German codes. This knowledge enabled the French Deuxième Bureau to intercept and decipher German encrypted messages from the beginning of the war. The French high command was able to obtain the enemy's order of battle and to learn of his plans to attack Paris, and they were able to stop the German offensive in the West, at the famous "miracle of the Marne".

After this success the French cryptanalysts joined forces with their British allies and from 1917 with their American counterparts. They openly exchanged intercepts, decrypts, knowledge, and methods on a continuous basis. On the US side the Radio Intelligence Division, under the command of Lieutenant Colonel Frank Moorman, was attached to the General Staff of the American Expeditionary Forces (AEF). It included the Radio Interception Section and the Code Solving Section, headed by William F. Friedman.

One of Friedman's officers was James Rives Childs. Childs was born on February 6, 1893 in Lynchburg, Virginia. During 1909-1911 he studied at the Virginia Military Institute and from 1912 at the Randolph-Macon College where he obtained his bachelor's degree. After receiving his master's degree from Harvard in 1915, he joined the US Army, and served in the American Expeditionary Force. He was assigned to G.2 A.6, the radio interception service, as the liaison officer with the French and British codebreaking services. During this period, he worked closely with George Painvin, learned from his methods which he successfully applied to East Front ADFGVX traffic. After the war he developed a general solution for the ADFGVX cipher. Childs received the Medal of Freedom for his work cracking German codes [87].

After the war, Childs entered the U.S. Foreign Service. During WWII, he was the chargé d'affaires at the US Legation in Tangier, Spanish Morocco, where he helped saving 1200 Hungarian Jews from the Holocaust by securing Spanish visas for them. During his 30-year career as a US diplomat he held several ambassadorship posts in the Middle East. Childs passed away on July 15, 1987, in Richmond, Virginia. He wrote numerous books and articles, about his service as an ambassador, about Casanova, and about his work on deciphering German ciphers. His papers are held in a special collection at the University of Virginia.

In 1919, while still in Paris, he wrote "The History and Principles of German Military Ciphers, 1914-1918" [70]. This manuscript was not published, and copies are available at the Randolph-Macon College – the University of Virginia Library as well as at the Kahn Collection at the National Cryptologic Museum in Fort Meade. Interestingly, a German version of the manuscript, translated in 1969 at Bad Godesberg, is also available at the Library of the German Armed Forces [69]. The manuscript contains an introduction by David Kahn. In 1935 the US War Department published a report named "German Military Ciphers from February to November 1918". This report was declassified by NSA in 2014 [71]. Though the document contains reports from several authors, the document underlines Childs' authorship.

In 1934, the US War Plans and Training Division published the "General Solution of the ADFGVX Cipher System", also recently declassified and available on the NSA site [75]. In 2000, the report was published by Aegean Park Press [68]. This later edition includes the 460 enciphered radiograms, most of which we have solved and analyzed. The editor states that those WWI intercepted German messages were furnished by Childs "some 20 years ago" (i.e. around 1980) [68].

6.6.3 Reading German Communications from Romania

In this section we present some of the key intercepts and decrypts of ADFGVX traffic related to Romania. We start with an overview of the political and military situation towards the end of the war, as well as Germany's communications infrastructure in that country. We also present the callsigns used by the relevant radio stations. While some of the callsigns were identified by Childs, an analysis of the traffic (similar to traditional Traffic Analysis) as well as external sources were required to recover the remaining callsigns.

On May 7th, 1918, Romania signed the "Peace of Bucharest" treaty, following its defeat against Germany. Under the clauses of this treaty, most of the Romanian army had to be demobilized, and the country was to be occupied by German forces. Field Marshall August von Mackensen was assigned as the head of the German occupation army. Its HQ, the Oberkommando Mackensen (OKM), was initially in Bucharest, with callsign UKS. The 11th Army (AOK 11, callsign ZÖN[2]) was also assigned to this area. Mackensen later transferred his personal headquarter to the palace of Pelesch, 120 kilometers north of Bucharest. Mackensen seemed to appreciate modern communications technology – telephone and telegraph. "The war has perfected all these facilities. We can talk with headquarters as two people sitting in front of each other", he remembered [88, p. 149].

A vast railroad network connected occupied Romania to the German Empire, and to the port of Constanza (callsign COS) on the Black Sea. Another port of strategic and logistical importance for Germany was the Ukrainian port of Nikolajev (callsign NKJ). Every day 100 wagons with petroleum left for Germany which badly needed it for its war machinery.

The outcome of the war, however, depended mainly on developments on the West Front. The 1918 German spring and summer offensives eventually failed, partly due to the Allies cryptanalytic successes. On September 29, 1918 the Kaiser's Supreme Army Command asked its government to seek an armistice with the Entente, to avoid a complete military defeat and the occupation of Germany. In October, the Austria-Hungary empire started to disintegrate into smaller nations. Some of them, like Bulgaria, left the alliance with the Central Powers (Germany and Austria-Hungary), and signed an armistice with the Entente. On October 31, 1918 the

[2] In Childs' transcriptions there are also the versions ZO'N and ZON.

Hungarian Government terminated the union with Austria, and on November 3, Vienna signed an armistice with the Entente. That same day, the social and political protest in Germany against the war culminated in a mutiny of the German Navy at Kiel, and spread all over the German Empire and the occupied countries.

On November 3, 1918, a 13-part message was intercepted by the British, and decrypted by Childs. Two of the parts are missing and some of the letters are garbled. It contains "the entire plans by which General Mackensen proposed to retreat from Romania" [71, p. 14].

LP v NKJ (7.06 p.m., November 4th)
NKJ AN LP VON UKS AN OHL (3) 13 TLE

Part	Reconstructed German text	Translation	Cryptogram starting with
1	BEURTEILUNG DER LAGE BISHER MUSSTE DAMIT GERECHNET WERDEN DASS FEIND MIT DEN BEI VID IN LOEPALANKA UND I5 GEGEN DRUTSCHUK IN VERSAMMLUNG BEGRIFFENEN KRAEFTEN EINEN DONAUUEBERGANG VERSUCHEN WIRD MIT DEM ZIEL DIE BAHN ORSOVA CRAIOVA ZU UNTERBRECHEN UND AUF BUKAREST VORZUSTOSSEN	Evaluation of the situation Until now it had to be expected that the enemy would try to cross the Danube with the troops assembling at Vid in Lompalanka and near Drutschuk with the goal to cut off the railways between Orsova and Craiova and to push forward to Bucharest	GXAAG XVGFF
2	[Original cryptogram missing. Translation from Childs[89, p. 212]]	Since November 1, 1918, it appears that the Serbian armies, together with three French divisions, are engaged in an advance toward Belgrade-Semendria, and the intended attack at Vidi and Lompalanka seems to have been abandoned.	
3	MIT DEM AUFMARSCH STAERKERER KRAEFTE AN DER DONAU SUEDLICH SVISTOV RUSTSCHUK MUSS BESONDERS NACH FRIEDENSSCHLUSS DER TUERKEI WEITER GERECHNET WERDEN [68, p. 113 A]	It has to be expected the deployment of stronger forces at the Danube south of Svistov Rustschuk specially after the peace agreement of Turkey.	DDAGA AXGGG
4	ES IST SOMIT DURCHAUS WAHRSCHEINLICH DASS DIE DURCH FRANZOSEN VERSTAERKTEN SERBISCHEN ARMEEN DEN UEBERGANG UEBER DIE [68, p. 114 A]	Hence it is most likely that the Serbian armies, reinforced by the French, intend to cross the	VXFXD GVDGX
5	DONAU BEI BELGRAD SEMENDIA UND DEN EINMARSCH IN SUE SUE UNGARN BEABSICHTIGEN WAEHREND DIE AUFGABE DER SUEDL	Danube near Belgrade Semendia and the into southern [repeat: southern] Hungary. While the task of the	VGGGV XDVGD

Part	Reconstructed German text	Translation	Cryptogram starting with
6	SVISTOV RUSTSCHUK AUF-MARSCHIERENDEN FRANZOES ARMEE OFFENSIVE RICHTUNG BUKAREST BESTEHEN BLEIBT IM ZUSAMMENHANG MIT DIESER	French Army marching south of Svistov Rutschuk remains the offensive toward Bucharest. In conjunction with	GADAG GXDGV
7	OPERATION IST ES NICHT AUSGESCHLOSSEN DASS RUMAEN KRAEFTE AUS DER MOLDAU DURCH DEN TOELGYES GIMES UND OITOS PASS IN	this operation it cannot be ruled out that the Romanian forces coming from Moldavia over the passes of Toelgyes, Gimes and Oitos [will invade Transylvania]	XAVGG AGFFA
8	SIEBENBUERGEN EINRUECKEN DADURCH WERDEN DIE RUECKWAERTIGEN VERBINDUNGEN DES BESATZUNGSHEERES DIE BISHER NUR IN FOLGE	Thus the lines of communications in the rear of the Occupation Army, which have up to now as a result of	GAXVA FAVFG
9	(9th part missing) [68, p. 114–115]		
10	ANGRIFF BEDROHT UND DIE WEITERE BESETZUNG DER WALACHEI WIE IN O K 2 RM1 A NR 11161 AUSGEFUEHRT ZWEIFELLOS UND IM HINBLICK	is threatened with attack and the further occupation of Wallachia as laid down in order from OK (Headquarters) 2 IA NR 11161. Without doubt and with regard to	VAGFD DFXGA
11	AUF MUNITIONS VERPFLEGUNGS UND KOHLEN VORRAETE UNDURCHFUEHRBAR FALLS DER ALLGEMEINE WAFFENSTILLSTAND NICHT IN ABSEH-BARER	the ammunition, food, and the coal stocks, it is unfeasible. If the general armistice does not become effective in the foreseeable future	GDVGA VFVGG
12	ZEIT EINTRITT WIRD DESHALB VORGESCHLAGEN DAS BESATZUNGSHEER SOFORT AUS RUMAENIEN HERAUS ZU ZIEHEN UND GEMEINSAM MIT DEN DEUTSCHEN	it is suggested that the Occupation Army be withdrawn at once from Romania and together with the German	A-VDA GGXGG
13	[Many symbols missing. Some of the translation from Childs [68, p. 213]]	units of the first Army to start the march to Upper Silesia through Hungary. Approval is requested. (Signed) K.M. I A GR-OP	GVVFG -GFGD [68, p. 116]

TABLE 6.3: ADFGVX message from Mackensen to the Supreme Army Command – November 3, 1918

This long message was sent from Mackensen HQ (OKM, using callsign UKS) to the Supreme Army Command (OHL, callsign LP) via Nikolajev (callsign NKJ). Its decryption provided the

Allies with an insight into Mackensen's evaluation of the situation and of his planning. According to Kahn, the decrypted message was immediately handed over to the French Supreme War Council [1, p. 339].

On November 9, the Kaiser was forced to abdicate, and Germany became a republic. Two days later, the German government signed the armistice at Compiègne and started to withdraw its armies from the Western Front. The war was over but hundreds of thousands of German soldiers were still deployed in the Eastern Front, cut off from their homeland. Due to a shortage of coal, wagons, and engines, an organized evacuation could not be planned. The situation became critical as the Entente imposed a timetable which could not be met.

On November 25, 1918, at 16:30h, OKM sent to OHL in Berlin another message, also intercepted and decrypted by Childs. Mackensen informs the Supreme Army Command, the War Ministry, and the new Government at Berlin that the French commanding officer in the region, general Berthelot, has notified him that general Foch refuses to apply the conditions of the armistice to the Southeastern Front, and demands the immediate disarmament and internment of Mackensen's troops. Mackensen is awaiting a decision from OHL on the matter before confirming the reception of Berthelot's message. He also proposes that disarmament be performed by Hungary rather than by Entente countries, and that internment take place in Hungary, to guaranty appropriate food supply for the troops. He is concerned that the situation could deteriorate into a "mass exodus with indescribable misery and open revolt" [68, p. 221 A].

While waiting for a final decision, Mackensen is getting more and more concerned about communications. He cannot use Nikolajev (NKJ) anymore as a relay station, and he communicates with Berlin via the Breslau station (BSL). On November 27, 1918 OKM requests BSL "to staff the station with very good operators" because radio is now "the unique secure communication with the homeland" [68, p. 224 A].

On November 28, the Allies intercept a message from Berlin (LP) via Breslau (BSL) to OKM (ZON) with instructions "to avoid every kind of struggle and to accept the French demands. Further orders will follow" [68, p. 227 D]. Final instructions arrive November 29, 1918 and this message is also intercepted and decrypted by Childs. The fate of Mackensen's army is sealed:

ZÖN v BSL (04.20 a.m., December 2nd)
NR 132 2200 (11/29) 4 TLE

Part	Reconstructed German text	Translation	Cryptogram starting with
1	SOLDATENRAT DER ARMEE MACKENSEN KAMERADEN TROTZ ALLER BEMUEHUNGEN DER IN BETRACHT KOMMENDEN SITUATIONEN IST ES UNS NICHT GELUNGEN DIE ENTENTE VON IHREM UNGERECHTFERTIGTEN STANDPUNKTE ABZUBRINGEN	To comrades of the Soldiers Council of the Mackensen Army. Despite all efforts regarding the coming situation we have failed to convince the Entente to renounce its unjustified point of view.	FVVVA GDVXF
2	SIE BEHARRT AUF EURER INTERNIERUNG UND BETRACHTET DIE AUFRECHTERHALTUNG DES ABMARSCHBEFEHLS ALS EINEN BRUCH DES WAFFENSTILLSTANDES DER IHR DAS RECHT GIBT IN DEUTSCHLAND EINZUMARSCHIEREN	It insists on your internment and views the order to march as a violation of the armistice which gives it the right to invade Germany.	VVFFD FVFXG
3	ES IST UNSER BEMUEHEN EURE INTERNIERUNG SO ZU GESTALTEN DASS IHR WEDER NOT NOCH MANGEL LEIDET UND NACH DEM MOEGLICHST RASCHEN FRIEDENSSCHLUSS GLUECKLICH UND GESUND IN DIE HEIMAT ZURUECKKEHREN COENNT [68, p. 226 C]	Our efforts are to arrange your internment so that you do not have to suffer deprivation and that you can safely come home as fast as possible	XDFFV FVDGF
4	HALTET RUHE UND ORDNUNG DAMIT NICHT DURCH INNERE AUFLOESUNG IHR ALLE VERLOREN GEHT WIR HOFFEN EUCH ALLE RECHT BALD GESUND IN DEM NEUEN DEUTSCHLAND WIEDER ZU SEHEN VOLLZUGSRAT DER A UND S R ABTE GROSSBERLIN SOLKENBUHR MUELLER [68, p. 227 A]	Maintain law and order so that you will all not get lost by internal dissolution. We hope to see you all again, soon and healthy, in the new Germany. Executive Council of the W[orkers] and S[oldiers] Councils, Dept. Great Berlin Solkenbuhr, Müller.	FFFFV XGVXF

TABLE 6.4: ADFGVX message with final instructions to the 11th Army in Romania – December 2, 1918

The last message about Romania in Childs' collection dates from November 30, 1918. An unidentified officer named Papp, using callsign CV, asks the 11th Army to "send the laws of internment in time" [68, p. 230 E]. In December the Field Marshal enters Hungary with his

remaining troops. He is interned in a palace with comfortable conditions and guarded by French troops, while most of his former soldiers are sent back to Germany.

6.6.4 The German November Revolution

On October 28 a military and civil uprising against the war and the Imperial Government starts at Kiel, the main port of the Imperial Navy. The crews of the Imperial Navy refuse to obey an order for a last and futile attack on the British Navy. At first, the military authorities attempt to oppress the Kiel mutiny, without success. On November 3 the mutiny turns into an overt rebellion which begins to spread all over the German Empire and the occupied countries. The German government and the Supreme Command of the combined armies (Oberste Heeresleitung, OHL) are still trying to contain the spread of the rebellion. As part of this effort, OHL needs to secure the communications infrastructure to avoid the takeover of radio stations by the rebels.

At 13:31h on November 6 the OHL Chief Signal Officer (Nachrichtenchef) sends an order marked with "high urgency" from the main military radio station at Berlin (callsign LP) to the army radio stations at Constantinople (YZ) and Nikolayev (NKJ), and via them to all radio stations in the Black Sea Area, including Romania, Georgia, and Turkey [68, p. 140 A]. All radio stations should be taken over immediately by an officer who should "keep a firm grip on them". The officer should prevent any radio messages from Kiel from being recorded and distributed. Any such previously received texts should be confiscated and destroyed. The radio station at Nikolayev forwards this message to the station ASO (probably Tiflis in Georgia), and the first part of this transmission is intercepted and decrypted by Childs on November 7th [68, p. 146 A].

On November 8 the signal office informs the General Staff at Spa in Belgium (callsign SY) that the German military radio station in Warsaw has "surrendered to the Soldiers' Council" [68, p. 151 C]. One day later, signal officer Seifert (his name also appears in other messages) informs SY that from 22:00h the radio station will be under the control of the Soldiers' Council, but that he still has a "firm grip" on the station and its personnel [68, p. 168 A].

On November 9, the German Republic is proclaimed in Berlin. On the same day, a new message is intercepted and decrypted by Childs. It describes how the Revolution is spreading from the Baltic Sea to the western parts of the German Empire. The signals division (N4) informs the HQ at Spa that the Soldiers' Council of Cologne is trying to take over the local radio station [68, p. 168 B]. Cologne and its infrastructure were of strategic importance for both the Navy and the Army. Still trying to maintain control over communications, the Supreme Command sends an order – also intercepted by the Allies – to stop using the cable network (phone and telegraph), and instead use only radio transmission and army ciphers. All messages should be confirmed twice [68, p. 168 C and D].

Undoubtedly, the November Revolution had a major impact on Germany's willingness and haste to agree to an armistice. The Allied powers were able to follow the travel preparations of the German delegation to the negotiations, and to identify the unit in charge of this operation. A message from November 10, 22:57h, for instance, refers to captain von Weiher, commander of a company of the 1st Bataillon of the 1st Foot Guards Regiment. Since the Napoleonic Wars all princes of Prussia served in this Regiment, whose commander traditionally was the King of Prussia. The intercepted message indicates that the "Erste Garde Regiment" is responsible for organizing the transport "by train or car of the members of parliament to Spa". They are scheduled to arrive in Berlin on November 11 at 15:00h, after having signed the armistice [68, p. 172 C and p. 175 A].

On November 11, at 19:06h, the military radio station in Berlin (LP) sends the details of the armistice to Nikolayev (NKJ) with instructions to forward them "by all means" to the Mediterranean Division (MMD) and to the Special Command of the Navy Operational Command (SKL). This message was probably sent quite late as it states that the armistice should take effect at noon on the same day. "Every act of war has to be stopped" and the naval forces should be confined to their bases. Submarines also have to be informed – using a ciphered message – that the armistice had come into effect. Weapons should be used only for self defence. Finally LP informs that permission has been requested from neutral governments for safe passage of German ships over neutral territorial waters [68, p. 178 A and p. 178 B].

One of the conditions of the armistice was the complete demilitarization from German troops of the region between the Rhine river and Germany's western border. As a result, the Army GHQ moved to Kassel. However, the revolution also reached this city. On November 10, the lower army ranks disarm their officers. A Workers and Soldiers' Council is established, and a red flag hoisted over the city's arch of triumph. An officer is shot while trying to tear down the flag. Those events are also reflected in the intercepted ADFGVX traffic. In a message from November 12 at 22:09h, a certain lieutenant Bauman[n], probably the signal officer in charge of a radio station, is ordered "to be ready to move to Kassel via an M[ilitary?] transport, as soon as possible" [68, p. 180 E].

On November 13 the Soldiers' Council in Warsaw asks the radio station at Nikolayev whether the rumors that field marshal Mackensen is marching into Warsaw with his troops are true. Mackensen was known a strong supporter of the monarchy, and the Soldiers' Council feared that Mackensen would cross Poland with his 170, 000 soldiers on their way from Romania to Germany [68, p. 181 A].

6.6.5 A Code within a Code

Three messages, divided into four cryptograms, contain enigmatic references to unidentified people or entities, such as 'Dictator' and 'Herkules'. The first message was sent on November 4, at 14:05h, from Berlin (LP) via Nikolayev to Bucharest. The Kriegsamt (the War Office) informs Mackensen's economic staff firstly that "the gold transport has arrived", and that "Herkules" rejects again any responsibility for new emergency calls from "Andreas" [68, p. 122 C].

On the same day, at 19:56h, a second message was sent from the Navy radio station at Eilvese (call sign OUI) to Osmanié (OSM), the major radio station at Constantinople. It consists of three parts, which appear on separate pages in Childs's book. The first part on page 120, while the second and third parts are on page 123 [68, p. 120 B and p. 123 A]. The sender is the supply department of the German Empire (Reichsversorgung). It informs the "Feigenhaus" ('house of figs') in Constantinople that "Diktator" has given the instruction "not to make emergency calls to [sic] Andreas", and that the Reichsversorgung has again contacted "Herkules". This enigmatic person or institution rejects any further responsibility "for new calls for help". "We cannot make new emergency calls" states the Reichsversorgung and orders "Feigenhaus" to act accordingly. In addition Feigenhaus should report via telegraph "what it is planning to do regarding the political situation, which keeps changing."

The third message is from November 15, at 16:50h, and is signed by the supply department and by the Admiralty Staff. It was sent by the main radio station at Berlin (LP) to the radio station with the new callsign XYZ, probably a German Army radio station in Constantinople. According to this message, "Herkules" has informed Berlin, that on November 12 several steamers from

"Operette" have reached Nikolayev with nearly 10 tons of liver and cacao on board. "The cargo remains under military guard" because of transport problems. The economic staff is required to gather more information on that issue, and to inform the supply department about OKM's intentions for this shipment [68, p. 199 A].

We may only speculate about the identities of the persons and organization. The "Diktator" could simply refer to von Hindenburg, the head of the OHL. "Andreas" is possibly a codename for Georgia, a newly created state, where Germany had a military presence and strong economic interests often conflicting with the interest of its ally, the Ottoman Empire. "Herkules" could be a codename for either OHL or Naval Command. "Feigenhaus" could be the supply or the economic staff of the German Army in Constantinople, and "Operette" the name of a port in the region. Still, it is not clear why codenames were used in those messages, while most of the other messages did not use them.

6.6.6 Hagelin Mentioned in an ADFGVX Message

Among the decrypted messages we have found a message mentioning Hagelin, most probably Karl Wilhelm Hagelin, the father of the famous cryptographer Boris Hagelin. The latter was born in Adshikent, Azerbaijan in 1892, where his father was director of the "Naphtaproduktionsgesellschaft Gebrüder Nobel". The company had several oil production facilities in Baku and its headquarters in Saint Petersburg. Karl Wilhelm was a member of the management staff of Emanuel Nobel, the "Swedish Rockefeller" and the nephew of the famous Alfred Nobel, the inventor of dynamite. After Alfred's death, Emanuel implemented his will and established the Nobel Prize foundation. After the Bolshevik revolution, the Hagelin family had to leave Russia. With the help of his father and Emanuel Nobel, Boris Hagelin joined Arvid Damm's Aktiebolaget Cryptograph and later turned it into one of the most successful vendors of encryption machines.

On October 5 1918 , at 03:23h, the Allies intercepted a message mentioning Hagelin [68, p. 64 A]. The sender is probably the quartermaster of the German Army in the East, the Ostarmee. The message was sent via the Nikolayev radio station (NKJ). The receiving station is unknown since the beginning of the message is missing. The message is for captain Benedikt, the representative of the Austria-Hungarian Army Command in "Atom". The officer should deliver the telegram to "Gebrueder Nobel Atom" and make sure that the response is sent back to the Ostarmee. Details are requested about stocks of "naphtha production of all sorts in our possession and with other Atom companies, how much has been delivered and the same information about Novorossiysk, if known – Hagelin". Novorossiysk is a Black Sea port in the Caucasus region, from which oil supplies from Baku were shipped. The name 'Hagelin' appears near the signature, but the context is not clear, whether the information should be sought from Hagelin, or whether the sender is not sure whether or not captain Benedikt knows Hagelin.

— v NKJ (3.23 a.m., October 5th)
(Beginning lost)

Part	Reconstructed German text	Translation	Cryptogram starting with
1	HPTM BENEDIKT VERTRETER KUK AOK IN ATOM ATOM ERSUCHE DRINGENDE ZUSTELLUNG FOLGENDEN TELEGR U VERANLASSUNG DASS ANTW TELEGR ANHER UEBERMITTELT WIRD GEBRUEDER NOBEL ATOM DRAHTET VORRAETE	Capt. Benedikt, representative of the Austria-Hungarian Army Command in Atom (repeat: Atom) requesting urgent delivery of the following telegram and that answer be transmitted telegraphically to here Nobel Brothers Atom. Telegraph stocks of	GDDD FGFAF
2	NAPHTA PRODUKTION SAEMTLICHER SORTEN BEI UNS UND ANDEREN FIRMEN ATOM FERNER WIEVIEL BIS JETZT GELIEFERT DIESELBEN DATEN BETREFFEND NQVOROSSISK FALLS SOLCHE IHNEN BEKANNT HAGELIN OSTARMEE O QU 12059	naphtha production of all sorts in our possession and with other companies Atom, how much has been delivered and the same information about Novorossiysk, if known – Hagelin Ostarmee Quartermaster	AVDFF VDVVG

TABLE 6.5: ADFGVX message mentioning Hagelin – October 5, 1918

6.7 Summary

The new attack on ADFGVX presented in this chapter and based on the new methodology described in Chapter 4, performs better than previous methods. In Table 6.6, we summarize the how we applied the principles of the methodology.

Principle	**Application of the methodology principle**
GP1	Hill climbing, sequential (2-phase) search. First HC to recover the transposition key, second HC for substitution key
GP2	Divide-and-conquer – transposition then substitution
GP3	Specialized IC-based scores with high resilience to errors
GP4	Non-disruptive transformations applied on key segments Variable neighborhood search
GP5	Multiple restarts

TABLE 6.6: ADFGVX – applying the methodology

More importantly, the attack has been validated by successfully recovering the keys from a unique collection of historical ADFGVX messages from WWI. The survey of some of the historical findings uncovered with the help of those decipherments also demonstrates how the study of classical ciphers and their cryptanalysis using modern methods can contribute to historical research.

7

Case Study – The Hagelin M-209 Cipher Machine

In this chapter, we present a case study about of the cryptanalysis of the Hagelin M-209, the most popular historical encryption machine. Section 7.2 describes the machine and its functioning. Section 7.3 presents related work and prior cryptanalytic methods. Sections 7.4 and 7.5 present two novel algorithms which we developed, a known-plaintext attack as well as a ciphertext-only attack, implemented along the guidelines of our new methodology. Those methods achieve significant improvement over prior methods, and in terms of performance, they are today state of the art. With those new attacks, we were able to solve a number of public challenges related to this cipher machine.

Some of the results presented in this chapter have also been published in Cryptologia [35][36].

7.1 Background

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. About 140 000 M-209 units were produced in total. It was used in WWII by both sides of the conflict and by neutral countries, including the US, France, Sweden, the Netherlands, and Italy. After the war, a large number of countries purchased those devices as well as their successors, and the US continued to use it up to the Korean War. The Hagelin M-209 has been the focus of intensive efforts by codebreaking agencies to read the traffic of their enemies and of their allies. More recently, statistical methods, and methods based on linear algebra have been applied for its cryptanalysis, and lately, methods based on local search metaheuristics.

7.2 Description of the Hagelin M-209 Cipher Machine

We present in this section 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, and finally we analyze the size of the keyspace of the Hagelin M-209.

7.2.1 Functional Description

The M-209 is built only of mechanical components, and does not require any power source. Figure 7.1 shows the mechanical internals of the M-209 device. Figure 7.2 describes its logical functioning. 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 7.1 (C). The pseudo-random displacement generator consists of two parts: A rotating cage with 27 bars (see Figure 7.1 (B) and Figure 7.2), and a set of six wheels (see Figure 7.1 (A) and Figure 7.2). 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, exactly one of its pins is positioned right against the bars of the cage, also denoted as the *active position* (the role of the pins in the active position is described in the next paragraph). 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 7.2.3), 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 the cover closed on the left side of Figure 7.2). The operator encrypts the message letter by letter. He selects a plaintext letter using the disk on the left side of the device, and presses the power handle on the right side of the device. 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 positions. 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 7.2), 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 7.2), to form the ciphertext letter, according to Euation 7.1.

$$CiphertextLetter[i] = (Z - PlaintextLetter[i] + Displacement[i]) \; mod \; 26 \tag{7.1}$$

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 numbers 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.

7.2.2 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,

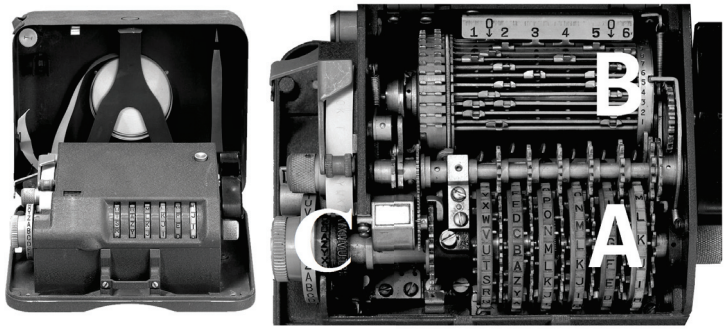


FIGURE 7.1: Hagelin M-209 – mechanical internals[1]

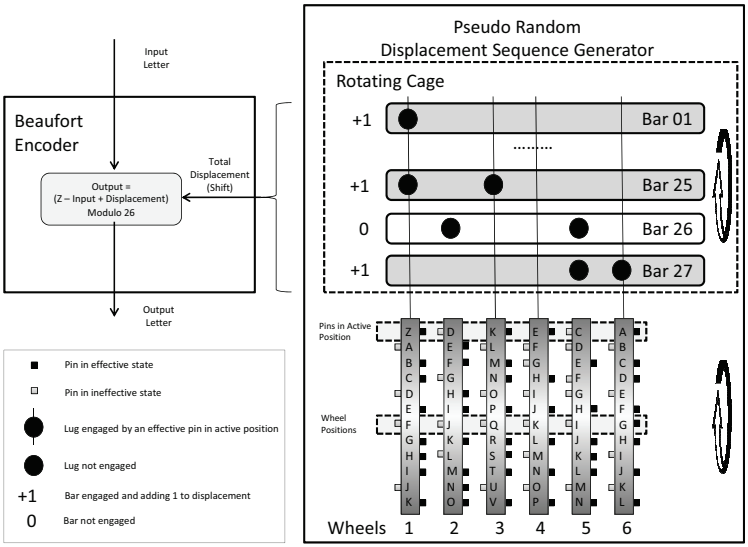


FIGURE 7.2: Hagelin M-209 – functional diagram (source: George Lasry)

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 [90], 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

[1]Source: 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

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:

$$CiphertextLetter[i] = (Slide - PlaintextLetter[i] + Displacement[i]) \; mod \; 26 \qquad (7.2)$$

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 [91]. 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 [92]. The Hagelin BC-38 and BC-543 were functionally equivalent to the C-38 but also included a keyboard and an internal motor. A model developed for the French Army had an additional substitution stage.

Post-war C-Series models, such as the C-52/CX-52, also had 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 [91]. 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.

7.2.3 Operating Instructions

Several versions of the US Army operating instructions for the M-209 device were in effect during 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 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 [93]. The list is presented in Table 7.1. We were able to obtain

Year	Details
1942	TM 11-380 Technical Manual Converter M-209, 33 pages April 27, 1942 [95]
1943	TM 11-380B Technical Manual Converter M-209, 42 pages September 20, 1943 [96]
1944	TM 11-380 Technical Manual Converter M-209, M-209A, M-209B, 78 pages, March 17, 1944 [97]
1947	TM 11-380 Technical Manual Converter M-209, M-209A, M-209B, 170 pages May 1947 [98] [94]
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 [99]

TABLE 7.1: Hagelin M-209 – versions of the operating instructions

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 changes in key selection procedures. The May 1947 version is included in its entirety in Barker's book [94].

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 $1 \leq w \leq 6$. According to the 1942 version of the technical manual, the requirements for Lug Counts are as follows:

1. $0 < LC[w] \leq 13$, 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 of 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, it 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 Technical Manual [95]. We show in Table 7.2 the full set of these Lug Count Patterns. 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 LC 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 7.2 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.

As mentioned above, 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 (Table 7.2), 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.

Pattern number	w1	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	9	11	6
38	1	2	4	6	7	13	6
39	1	2	4	7	9	10	6
40	1	2	4	7	8	11	6
41	1	2	3	6	11	12	8
42	1	2	3	6	10	13	8
43	1	2	3	7	10	12	8
44	1	2	4	5	11	12	8
45	1	2	4	5	10	13	8
46	1	2	4	6	9	13	8
47	1	2	4	7	10	11	8
48	1	2	4	7	9	12	8
49	1	2	4	7	8	13	8
50	1	2	4	8	9	11	8
51	1	2	3	6	12	13	10
52	1	2	4	5	12	13	10
53	1	2	4	6	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 7.2: Hagelin M-209 – lug count patterns for the 1942 operating instructions

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 than 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.

7.2.4 Keyspace

We present here a keyspace analysis of the device. The settings of the device consist 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.

7.2.4.1 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 131 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 position settings in the rare cases they are known, or simply assume default "AAAAAA" initial wheel positions, if they are not known. This is possible since any set of pin settings with initial wheel positions other than "AAAAAA", is logically equivalent to another set of pin settings in conjunction with the default "AAAAAA" initial wheel positions. To illustrate this, we consider the sample wheel pin settings shown in Listing 7.1, given the initial wheel positions "BBBBBB":

```
Wheel 1: 01101110011000111100001101
Wheel 2: 00111100011101001010011 0
Wheel 3: 00101110111101111011111
Wheel 4: 01001110111101111 0111
Wheel 5: 0101110111110111 1010
Wheel 6: 01101110111101111
```

LISTING 7.1: Hagelin M-209 – 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 in Listing 7.2.

```
Wheel 1: 10110111001100011110000110
Wheel 2: 00011110001110100101001 1
Wheel 3: 10010111011110111101111
Wheel 4: 10100111011110111 1011
Wheel 5: 00101110111111011101
Wheel 6: 10110111011110111
```

LISTING 7.2: Hagelin M-209 – 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}.

7.2.4.2 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 7.3.

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 7.3: Hagelin M-209 – 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 7.3, 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 7.3, 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 7.4, 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 7.4: Hagelin M-209 – lug settings equivalent to lug settings in Table 7.3

What actually matters is the number of bars set to 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 set to 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 7.5 which shows a non-redundant alternative representation of the lug settings of Table 7.3 or Table 7.4. We shall use this concise representation throughout this work, 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 7.5: Hagelin M-209 – non-redundant representation of the lug settings in Table 7.3

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 [100, p. 425, Theorem 2], the number of cryptographically distinct lug settings is as follows:

$$C(n+k-1,k) = C(21+27-1,27) = \frac{47!}{20!\cdot 27!} \approx 2^{43} \tag{7.3}$$

7.2.4.3 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 quantify accurately. 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 Table 7.2, 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 lug 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 lug 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 given in Equation 7.4.

$$C(15+v-1,v)\cdot C(6+(27-v)-1,27-v) \tag{7.4}$$

In Table 7.6 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).

Number of overlaps v	Possible lug settings	
2	17 100 720	2^{24}
4	300 736 800	2^{28}
6	2 549 632 800	2^{31}
8	13 591 504 080	2^{33}
10	51 647 715 504	2^{35}
12	149 732 980 800	2^{37}
Total	**217 839 670 704**	2^{38}

TABLE 7.6: Hagelin M-209 – lug settings options per number of lug overlaps according to Equation 7.4

7.2.4.4 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 $2^{131} \cdot 2^{38} = 2^{169}$.

For comparison, the size of the keyspace for a 3-rotor Enigma (see Section 10.2) is $2^{76.5}$. On the one hand, the Hagelin M-209 is susceptible to simple attacks on messages in depth (Section 7.3.1), unlike the Enigma. On the other hand, even state-of-the-art ciphertext-only and known-plaintext attacks (presented in Section 7.4 and 7.5) require much longer ciphertexts/known-plaintexts than the best performing attacks on Enigma (see Section 3.5.1).

7.3 Related Work – Prior Cryptanalysis

7.3.1 Historical Cryptanalysis

The Hagelin M-209 was one of the most widely used cipher machines, and it has been a focus of attention for several codebreaking organizations. Most of the historical cryptanalysis relied on the availability of messages in depth. Since the Hagelin M-209 encryption process is additive in nature, messages sent with the same settings can often be solved with the "Mots Probables" (probable words) method [94]. This is possible since the difference between corresponding plaintext letters at the same position in the two messages, is equal to the difference between the corresponding ciphertext letters. Therefore, if the analyst is able to guess a word in the first message, he can reproduce the letters of the second message at the corresponding positions by using a simple addition modulo 26 operation. As soon as he has recovered enough plaintext with the probable words method, he can use a known-plaintext attack to recover the full key settings and decrypt the rest of the message as well as other messages.

According to the declassified TICOM I-175 and DF-120 reports [101] [102], the German cryptographic services in WWII were able to recover key settings from ciphertext in special cases, such as messages "in-depth", and "near depth", e.g. messages transmitted with 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 [103], they were only able to solve a synthetic message in German with 5 000 letters.

7.3.2 Modern Cryptanalysis

Starting from the 1970s, efforts have been made for the cryptanalysis of the Hagelin M-209, using a variety of methods. In this chapter, we review the results of those modern attacks.

7.3.2.1 Morris (1978)

In [104], Robert Morris describes a known-plaintext attack for the recovery of the key settings. Since both the plaintext and the ciphertext are known, the displacement sequence (modulo 26) can be computed for all positions in the ciphertext. This attack is manual and is based on the iterative analysis and refinement of this displacement sequence. The method is complex and while some of the steps may be computerized, it relies heavily on the analyst's judgment, and it requires a great deal of practice and trial-and-error. We illustrate here the main concept of the Morris attack, i.e. the displacement histograms. To do so, we use the ciphertext shown in List 7.3, which has 75 letters. The corresponding plaintext is shown in Listing 7.4.

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

LISTING 7.3: Hagelin M-209 – sample message ciphertext

```
1   F R O M Z G E N E R A L Z A L E X A N D E R Z T O
    Z G E N E R A L Z P A T T O N Z O P E R A T I O N
3   Z H U S K Y Z S T O P Z T H E Z A M E R I C A N Z
```

LISTING 7.4: Hagelin M-209 – sample message plaintext

We use Equation 7.5 (derived from Equation 7.2) to compute the displacement sequence.

$$Displacement[i] = (CiphertextLetter[i] + PlaintextLetter[i] - Z) \; mod \; 26 \tag{7.5}$$

where $Z = 25$. We obtain the displacement sequence shown in Listing 7.5.

```
1   05,17,14,05,11,06,17,07,06,23,04,20,24,16,16,17,09,19,15,24,14,14,18,12,
    18,15,06,19,14,01,12,08,00,15,14,20,06,01,17,13,11,01,00,18,07,19,13,13,
3   15,13,18,16,04,00,23,15,15,04,25,02,16,21,13,19,20,06,01,22,04,16,05,19,
    13,14,00
```

LISTING 7.5: Hagelin M-209 – displacement sequence

Then, for each wheel, and for each one of its pins, we gather the relevant displacement values, and we compute the average displacement for each pin. At this stage we discard displacement values of 0 and 1, which may be ambiguous because of the modulo 26 operation. We assume that pin #1 of each wheel is at the active position when encrypting the first letter (position 0) of

the message. We start with wheel 1, which has 26 pins. In our example, pin #1 of wheel 1 is active at position 0, then again 26 steps later (position 26), as well as in position $26+26=52$. The displacement for position 0 is 5, it is 6 for position 26 and 4 for position 52. Therefore, the average displacement for pin #1 is $\frac{(5+6+4)}{3}=5$. Similarly, pin #2 of wheel 1 is active at position 1 (displacement = 17) and at position $1+26=27$ (displacement = 19). It is also active at position $1+52=53$, but the displacement is 0. Therefore, it is ambiguous because of the modulo 26 operation, and we ignore it at this stage. Hence, the average displacement for pin #2 of wheel 1 is $\frac{(17+19)}{2}=18$. The result for the pins of wheel 1 is displayed in Listing 7.6.

```
[pin 01]  05, 06, 04,    Average:  05
[pin 02]  17, 19, 00,    Average:  18
[pin 03]  14, 14, 23,    Average:  17
[pin 04]  05, 01, 15,    Average:  10
[pin 05]  11, 12, 15,    Average:  13
[pin 06]  06, 08, 04,    Average:  06
[pin 07]  17, 00, 25,    Average:  21
[pin 08]  07, 15, 02,    Average:  08
[pin 09]  06, 14, 16,    Average:  12
[pin 10]  23, 20, 21,    Average:  21
[pin 11]  04, 06, 13,    Average:  08
[pin 12]  20, 01, 19,    Average:  20
[pin 13]  24, 17, 20,    Average:  20
[pin 14]  16, 13, 06,    Average:  12
[pin 15]  16, 11, 01,    Average:  14
[pin 16]  17, 01, 22,    Average:  20
[pin 17]  09, 00, 04,    Average:  07
[pin 18]  19, 18, 16,    Average:  18
[pin 19]  15, 07, 05,    Average:  09
[pin 20]  24, 19, 19,    Average:  21
[pin 21]  14, 13, 13,    Average:  13
[pin 22]  14, 13, 14,    Average:  14
[pin 23]  18, 15, 00,    Average:  17
[pin 24]  12, 13,        Average:  13
[pin 25]  18, 18,        Average:  18
[pin 26]  15, 16,        Average:  16
```

LISTING 7.6: Hagelin M-209 – average displacement for the 26 pins of wheel 1

We repeat the process for all the remaining wheels. We can now draw a histogram of the average displacements, for each one of the 6 wheels. For example, for wheel 1 we have one pin with average displacement 5 (pin #1), one pin with average displacement 6 (#6), one pin with average displacement 7 (#17), two pins with average displacement 8 (#8 and #11), and so on. The average displacement histograms for all the wheels are shown is Listing 7.7, 7.8, and 7.9.

```
                                             X
                                             X
                                             X  X
             X    X XX                       X XX
        X   XXX  XX XX                     XXX XXX
     XXXXXX XXX XXX XX                X  XXXXXXXXX X X
__________________________      __________________________
00000000001111111111222222      00000000001111111111222222
01234567890123456789012345      01234567890123456789012345

          Wheel  1                        Wheel  2
```

LISTING 7.7: Hagelin M-209 – average displacement histograms for wheels 1 and 2

```
                X
            X   X
            X   X                             XX X
           XXXX X X                     X  XXX X
        XXXXXXXXX XX   X               XXXX  XXXXXXXXX
__________________________      __________________________
00000000001111111111222222      00000000001111111111222222
01234567890123456789012345      01234567890123456789012345

          Wheel  3                        Wheel  4
```

LISTING 7.8: Hagelin M-209 – average displacement histograms for wheels 3 and 4

```
             X X X
             XXX X                             X
            XXXX X                         XXXXXXX
          XXXXXX X                      X  XXXXXXX X
__________________________      __________________________
00000000001111111111222222      00000000001111111111222222
01234567890123456789012345      01234567890123456789012345

          Wheel  5                        Wheel  6
```

LISTING 7.9: Hagelin M-209 – average displacement histograms for wheels 5 and 6

Next, we look for the histogram with the best "bimodal distribution", i.e. with displacement values around two main distinct "peak" values. The best candidates are wheels #2 and #3. We select wheel 3, as we can see that there is a peak and a cluster of values around the displacement value of 12, and another peak (and cluster) around the displacement value of 16. Those two peaks and clusters most probably represent the set of ineffective pins and the set of effective pins, respectively. From the analysis of this histogram, we may assume that pins of wheel 3 with an average displacement of 12 or below are most probably pins in ineffective state, and that pins with an average displacement of 16 and above are most probably in effective state. No conclusion may be yet drawn for the other pins. With those assumptions, we may also reach conclusions in regard to some of the ambiguous values (0 or 1) in the displacement sequence. Using this knowledge about the pins of wheel 3, we further refine the displacement sequence

and the displacement histograms for other wheels, using interpolation techniques described in detail in Morris's paper [104]. We look for the next wheel for which the refined displacement histogram has the best bimodal distribution and repeat the process. We iteratively process all the wheels, until all pin settings have been recovered. When all pin settings have been recovered, the lug settings can also be recovered. One of the main challenges with this method, especially with messages shorter than 100 letters, is that after processing the first 2 or 3 wheels, very often none of the remaining wheels will show any discernible bimodal distribution pattern, even after applying the interpolations techniques proposed by Morris.

7.3.2.2 Barker (1977)

In his 1977 book [94], 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 at positions p_1 and p_2 on a given wheel, the analyst performs a Chi correlation test described in Equation 7.6:

$$\sum_{i=1}^{c} \frac{n_{i,p_1} \cdot n_{i,p_2}}{N_{p_1} \cdot N_{p_2}} \tag{7.6}$$

n_{i,p_1} is the count of the occurrences of the i-th letter of the A-Z alphabet, at pin position p_1, and similarly, n_{i,p_2} is the count of the same letter at pin position p_1. N_{p_1} and N_{p_2} are the total numbers of samples, at pin positions p_1 and p_2, respectively. c is the number of letters in the alphabet used by the Hagelin M-209, with $c = 26$.

The Chi test value indicates how close the letter frequency distributions for pins p_1 and p_2 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 of the method's performance.

In [105], Rivest presents a theoretical analysis showing that 8000 letters are required for cryptanalysis using Barker's Chi test method. He nevertheless concludes that in practice 2000 to 4000 letters are usually enough.

7.3.2.3 Beker and Piper (1982)

A ciphertext-only attack is presented by Beker and Piper in [29]. They propose different techniques to divide the pins into classes, and to solve ambiguities. They demonstrate the method on a sample ciphertext with 3000 letters. The plaintext of this sample message has an unusually high number of space symbols, internally converted to Z symbols. Baker and Piper claim that their method generally works with 2500 letters, and often with only 2000, but they do not provide any detailed quantitative analysis.

7.3.2.4 Sullivan (2002)

A method for ciphertext-only recovery of M-209 settings is presented by Geoff Sullivan in [106]. 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 2500 letters to recover most of the key settings. It tries to incrementally recover pin settings of certain wheels, while isolating the effects of the other wheels [106]. Sullivan's method relies on lug setting restrictions derived from the 1942 version of the operating instructions, as described in Section 7.2.3. 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. As depicted in Figure 7.2, 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 candidate 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 hill-climbing 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 hill climbing, 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 hill-climbing 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 hill climbing. It also keeps the candidate 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 lug overlaps. It applies a hill-climbing search with the Index of Coincidence as the fitness score, for the optimal settings of the pins of candidate wheel w. During hill climbing, 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 hill climbing, as well as the optimal number of lugs. When all candidate wheels have been tested, it selects the candidate wheel for which hill climbing 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 candidate pin settings for all the wheels, as well as candidate lug settings, although those lug settings are inaccurate since they do not take into account lug overlaps. Sullivan suggests an additional stage of hill climbing 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, lug 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 2 500 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, such a decrypted text is hard if not impossible to read.

7.3.2.5 Lasry, Kopal and Wacker (2016)

This ciphertext-only attack consists of a 4-stage hill-climbing algorithm [36]. Note that this method, which we also developed as part of this research, is presented as part of the prior work, since we have developed better methods, which we present later in this chapter.

For simplicity, this method is described in the context of the key setting guidelines specified in the 1942 version of the operating instructions (see Section 7.2.3). This method, however, is not restricted to the 1942 version. It can be applied to later versions as well.

The main challenge for any ciphertext-only attack on the Hagelin M-209 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 [106], it is possible to recover the pin settings using hill climbing, if the lug settings are known. Similarly, we were able to implement a simple hill climbing 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 7.3.

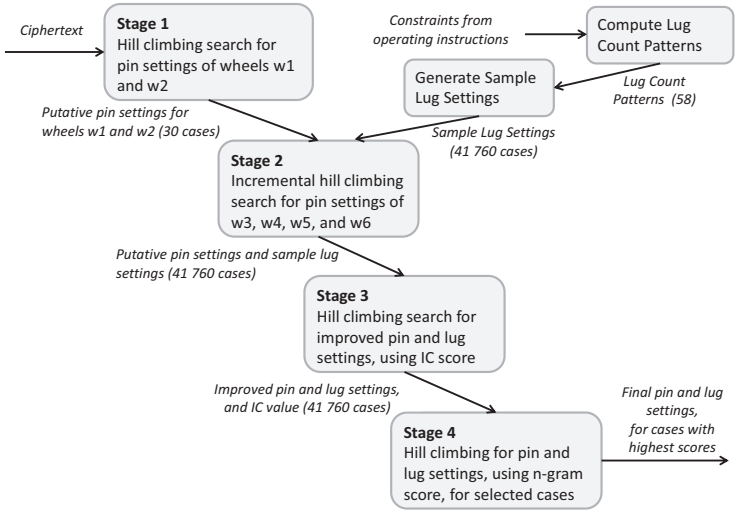


FIGURE 7.3: Hagelin M-209 – high-level flow diagram for the 4-stage algorithm

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 41 760 representative *Sample Lug Settings*. The outcome of Stage 2 is used as starting points for Stage 3, in which we perform a comprehensive hill-climbing search for improved pin settings and lug settings. We perform hill climbing on all 41 760 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 hill-climbing 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.

With this method, the keys for messages with 1 000 or more letters may be recovered, and if the number of lug overlaps is small (e.g. 2), for messages as short as 750 letters. This method, however, is ineffective for shorter messages (e.g. 500 or 600), and even with 750 letters if the number of lug overlaps is 6 or more. We also found solutions for the final exercises, #60 to #63, in Barker's book [94], by recovering the keys for a message with approximately 1 100 letters, encrypted with a key involving 6 lug overlaps. The cryptograms were created by Greg Mellen (1926-1998), Editor Emeritus of Cryptologia, and published in 1977, and they conclude a series of exercises with increasing difficulty.

The success rate for various numbers of lug overlaps, as well as for different versions of the operating instructions, is shown in Figure 7.4.

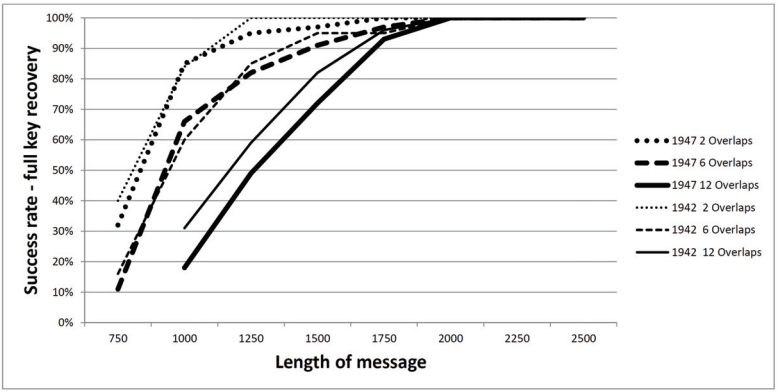


FIGURE 7.4: Hagelin M-209 – performance, Lasry and al. (2016)

This method also implements several of the principles described in this thesis:

1. Hill climbing (GP1).
2. Reducing the search space (GP2): A divide-and-conquer approach, with the first phase trying to recover the pin settings, while the lug settings keyspace is reduced to a set of approximate and representative lug settings.
3. Adaptive scoring (GP3): The first 3 stages use the Index of Coincidence, a scoring method with high resilience to key errors, while for the last stage, the less resilient to errors but more selective n-gram method is used.
4. Non-disruptive high-coverage transformations (GP4): All pin and lug setting transformations used for this attack apply a small change to the key settings.
5. Multiple restarts with optimal initial keys (GP5): Each stage produces an optimal starting point for the next stage, rather than starting from random key settings.

7.3.2.6 Morris, Reeds and Richie (1977)

An intriguing story was published by Dennis Ritchie in 2000 [107]. The story also appeared in Cyberpunk, Outlaws and Hackers on The Computer Frontier by Katie Hafner and John Markoff in 1991 [108]. 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 ciphertext-only 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 [104] (see Section 7.3.2.1). 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 lug 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 [109]. 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 [110]. The paper was finally released in 2015. The method is described here.

The concept is the similar to Barker, as well as Beker and Piper. The goal is, for each wheel, to divide its pins into two classes, one class with the pin set, and another class with the pin unset (or vice-versa). This is achieved by maximizing the Chi correlation between each pair of pin positions p_1 and p_2 in the same class, computed using Equation 7.6.

The procedure is described here, using the notation employed throughout this thesis, which is different from the notation used in the original paper. First, the relative frequency of the i-th letter of the alphabet, at pin position p, is computed, as described in Equation 7.7, for each i and p.

$$F_{i,p} = \frac{n_{i,p}}{N_p} \tag{7.7}$$

$n_{i,p}$ is the count of the occurrences of the i-th letter at pin position p, and N_p is the total number of samples, at pin position p. The frequencies of letters at each position are then normalized, according to Equation 7.8.

$$G_{i,p} = F_{i,p} - \frac{\sum_{i=1}^{c} F_{i,p}}{c} \tag{7.8}$$

Based on the normalized frequencies $G_{i,p}$, a correlation matrix between all pairs of positions p_1 and p_2 is computed, according to Equation 7.9:

$$C_{p_1,p_2} = \sum_{i=1}^{c} G_{i,p_1} \cdot G_{i,p_2} \tag{7.9}$$

The next step is to find the eigenvector of the C_{p_1,p_2} matrix, which corresponds to its largest eigenvalue. According to the sign of the corresponding element in the eigenvector, the wheel pin positions are divided into two classes. This procedure recovers the pins of a single wheel. To recover the pins of other wheels, the authors suggest an iterative process in which previous results (of dividing other wheels into two classes) are factored in, improving the accuracy of

the next step. The authors also describe a method to recover the lug settings, given that the pin settings have been recovered.

Simulations performed as part of the research for this thesis show that this method is effective for messages with 1 500 letters or more, and sporadically, for messages with 1 250 letters. It has, in addition, several advantages. First, it is very fast. Secondly, it is also effective for the case an additional substitution layer has been added, as for the case of the French Hagelin model, for which most of the other methods (including the methods we developed and present in this chapter) are ineffective.

7.4 A New Known-Plaintext Attack

7.4.1 Introduction

In this section we present a new known-plaintext attack which is also applicable to the case when only a part of the plaintext is known or can be guessed and/or the known plaintext is not contiguous. Only the number of ciphertext letters for which we know or can guess the corresponding original plaintext letter is relevant to our attack.

7.4.2 Description

Our attack is based on a hill-climbing algorithm, which repeatedly transforms the key in order to improve the corresponding fitness score. For the fitness score, we defined the Aggregate Displacement Error (ADE) function, as described in Section 7.4.2.4. For key transformations, we defined two types of transformations on the pin settings (see Section 7.4.2.2), and two types of transformations on the lug settings (see Section 7.4.2.3).

7.4.2.1 Main Algorithm

This section gives an overview of the steps of our algorithm:

1. Set the external initial wheels settings to the default value “AAAAAA”, or to the original external settings if those are known.
2. Generate initial random pin and lug settings.
3. Repeat the following three loops (3a, 3b, and 3c) until the ADE score no longer improves (i.e. no longer can be reduced):
   (a) Repeatedly perform the set of all the pin settings transformations as long as the ADE can be improved. For each transformation perform the following steps:
      i. Compute the ADE score.
      ii. If the ADE score with the new pin settings is lower than the score with the pin settings before the transformation, keep the new pin settings. Otherwise, discard the new pin settings.

(b) Repeatedly perform a subset of the lug settings transformations which do not involve lugs overlap as long as the ADE can be improved. For each transformation perform the following steps:

i. Compute the ADE score.

ii. If the score with the new lug settings is lower than the score with the lug settings before the transformation, keep the new lug settings. Otherwise, discard the new lug settings.

(c) Same as in (b), but for the complete set of lug settings transformations.

4. If an ADE score of 0 is reached, the correct settings have been found and the algorithm terminates. If neither loop 3a, 3b or 3c resulted in an improvement (reduction) of the ADE, repeat from Step 2, i.e. restart with new random settings.

7.4.2.2 Transformations on Pin Settings

In our automated known-plaintext attack we repeatedly perform transformations on the pin settings. The rationale behind each transformation is to make only a slight change in the pin settings starting from some initial settings. We never change the state of more than 1 or 2 pins in any single transformation. The full set of the pin settings transformations processed by hill climbing includes:

1. "Toggle" transformations: Toggle the state of one pin of one of the wheels. This means that if the pin is currently effective, then set it to ineffective, and if it is currently ineffective, then set it to effective.

2. "Swap" transformations: For a pair of pins, where the pins are not in the same state, toggle the state of both pins. For this transformation we consider any pair of pins, i.e. either two pins in the same wheel or in different wheels.

While any "Swap" transformation is logically equivalent to two "Toggle" transformations, the "Swap" transformations are needed to test the effect on the ADE of the two changes applied simultaneously, rather than sequentially computing the ADE after each one. This is necessary in case each one of the two equivalent Toggle transformations results in degrading the ADE, in which case such a transformation shall be discarded by the algorithm, while only the application of both changes at the same time (the "Swap" transformation) actually improves the ADE.

Since there is a total of 131 pins on the six wheels we have 131 possible "Toggle" transformations, and no more than $\frac{131 \cdot 130}{2} = 8\,515$ possible "Swap" transformations. Typically only half of the Swap transformations are relevant, as we consider only pairs of pins which are not in the same state (and typically, about 50% of the pins are in effective state).

7.4.2.3 Transformations on Lug Settings

In our automated know-plaintext attack we also iterate through a set of lug settings transformations. Each transformation affects a single bar. We use the concise and non-redundant representation presented in Section 7.2.4.2, basically an array counting the number of bars being set to each one of the possible 21 lug settings types. Those lug settings types include 6 types without

overlap and 15 types with overlap. In each transformation, the count for one of the types is decreased, and the count for another type is increased. With a physical M-209 device, this is the equivalent of changing the lug settings of a single bar. But since we employ the non-redundant representation of the lug settings (see Section 7.2.4.2), we ignore the physical representation of the individual bars, and instead apply transformations on the (abstract) type counts.

The full set of our lug settings transformations consists of the following categories:

1. Swap between two types of lug settings without overlap: For each one of the 6 non-overlap types, and each one of the remaining 5, reduce the count for the first type, and increase the count for the second type. If the first type already had a count of 0, we skip this case. Going back to our example in Table 7.5, an example of such a transformation would be to reduce the count for type 0-1 from 1 to 0, and to increase the count for type 0-2 from 3 to 4.

2. Swap between any two types of lug settings, with or without overlap: For each one of the 21 types, and each one of the remaining 20, reduce the count for the first type, and increase the count for the second type. If the first type already had a count of 0, we skip this case. Going back to our example in Table 7.5, an example of such a transformation would be to reduce the count for type 0-1 from 1 to 0, and set the count for type 1-2 to 1 (the previous count for 1-2 was 0, and therefore not shown in Table 7.5).

While the second set of swap transformations also contains the set of simpler swap transformations (i.e. the ones without overlap), the distinction is necessary, as we will prefer to check first the simpler transformations, as long as we can improve the ADE, before checking the more complex ones.

With the above rules there are no more than $6 \cdot 5 + 21 \cdot 20 = 450$ possible lugs transformations for our algorithm, and usually much less, as we can't reduce the count of a certain type if it was already 0.

7.4.2.4 The Aggregate Displacement Error Score (ADE Score)

We first implemented our hill-climbing algorithm using a simple scoring function, by counting the number of plaintext letters correctly reproduced after decryption using a candidate key, or more specifically, by counting incorrectly reproduced letters. With this simplistic function, the hill-climbing algorithm was able to recover the full key settings for messages with at least 300-400 known-plaintext letters. With 200 or less known-plaintext letters, our algorithm using this scoring method consistently failed. This simple scoring function is selective (see Section 3.2.4), as a key with a high score is most likely to have a small number of errors. On the other hand, this function has limited resilience to key errors: In order to reproduce the correct letter at a given position, it is necessary to recover all the key settings elements which contribute to the displacement at this position, i.e. the state (effective or ineffective) of the active pin of each one of the 6 wheels, and most of the lug settings. Therefore, the correction of only one of those elements is unlikely to be enough to correctly reproduce the plaintext letter, unless all the other elements were already correct.

We developed an alternative scoring function, the *Aggregate Displacement Error*(ADE), designed for improved resilience to key errors. To achieve that, the ADE takes into account the

individual contribution of each one of the key elements affecting the decryption of each one of the letters in the message, rather than just their aggregate effect (as for the simple count method described above). First, the concept of displacement error is introduced, for a single plaintext element (letter). It is then extended to the whole plaintext, by aggregating the displacements errors of all its elements.

Since our attack is a known-plaintext attack, both the plaintext and the ciphertext are known, and therefore the expected displacement at each step may also be computed, using Equation 7.10.

$$ExpectedDisplacement[i] = (CiphertextLetter[i] + PlaintextLetter[i] - Z) \; mod \; 26 \quad (7.10)$$

We define the *Displacement Error*, for a candidate key and for a certain position/letter in the ciphertext, as the absolute difference between the actual displacement, resulting from the current lugs and pin settings, and the expected displacement, as computed in Equation 7.10. The pseudo code for the computation of the displacement error is shown in Listing 7.10.

```
Input:
expectedDisplacement // The expected displacement
                     // calculated using crib and ciphertext
                     // It is equal to the original
                     // displacement, modulo 26.
actualDisplacement   // The actual displacement calculated
                     // using decrypted ciphertext and ciphertext
Output:
DisplacementError    // The error of expectedDisplacement and
                     // actualDisplacement
DisplacementError(expectedDisplacement, actualDisplacement)
{
    if (expectedDisplacement < 2)
    {
    // Optimistic approach -- we keep the lowest error value, either by:
    // (a) Assuming the original displacement was >= 26
    // (b) Assuming the original displacement was < 2
        return
            min[abs((expectedDisplacement + 26) - actualDisplacement),
                abs(expectedDisplacement - actualDisplacement)]
    }
    else
    {
        return abs(expectedDisplacement - actualDisplacement)
    }
}
```

LISTING 7.10: Hagelin M-209 – displacement error pseudo code

Special care is needed for expected displacement values 0 and 1, which can be ambiguous; the other values (2 to 25) are non-ambiguous. Because of the modulo 26 operation, there is no way to differentiate between original displacement values of 0 and 26, as well as between displacement values of 1 and 27. In case of ambiguity, we use an optimistic approach and assume the lowest error from the two possible alternatives. For example, if the expected displacement at a given position i is 0, the original displacement could have been either 0 or 26. The same reasoning applies to 1 and 27. When we compare the actual displacement resulting from a candidate

key, which may have any value from 0 to 27, to the expected displacement value, which may have values only from 0 to 25, we assume the lowest (i.e. optimistic) possible displacement error value. For example, if $ExpectedDisplacement[i] = 0$ and $ActualDisplacement[i] = 23$, then the error is assumed to be $26 - 23 = 3$, rather than an error of $23 - 0 = 23$.

While the displacement error applies to a single position, the ADE is simply the sum of the displacement errors for all the letters of the ciphertext for which the corresponding plaintext letter is known.

To verify the effectiveness of the ADE we compared it with the simple scoring method of counting incorrectly reproduced letters, using sample messages of various lengths, each with 100 known-plaintext letters, encrypted with random key settings. For each case, we modify the correct key settings, by introducing a series of 100 consecutive settings errors, each time in either one of the pin settings or one of the lug settings. At each step, we decrypt the ciphertext with the key settings (with 0 to 100 errors), and compute the ADE and the number of incorrect letters after decryption. A fitness-distance analysis (see Section 2.3.10) for 1000 sample messages is shown in Figure 7.5. For convenient comparison in the graph, the count of incorrectly reproduced letters has been scaled, so that its maximum value, for 100 errors, is the same as the ADE value for 100 errors. As expected, both the ADE and the incorrect letters count are 0 with the correct key, and they increase when additional settings errors are introduced, as long as the number of errors is not too high. However, the ADE graph is much smoother, and continues to be almost always monotonic, even with a high number of settings errors, displaying very high resilience (see Section 3.2.4) to key errors. This allows for the detection of subtle improvement or degradation after modifying key settings. The simple count measure is more selective, but has less resilience to key errors.

Due to its high resilience to key errors and relatively good selectivity, the ADE is a more effective scoring method for our hill-climbing attack, as also shown by the overall performance of the full algorithm – only 50 known-plaintext letters vs. at least 300 required for full key recovery with the simplistic score.

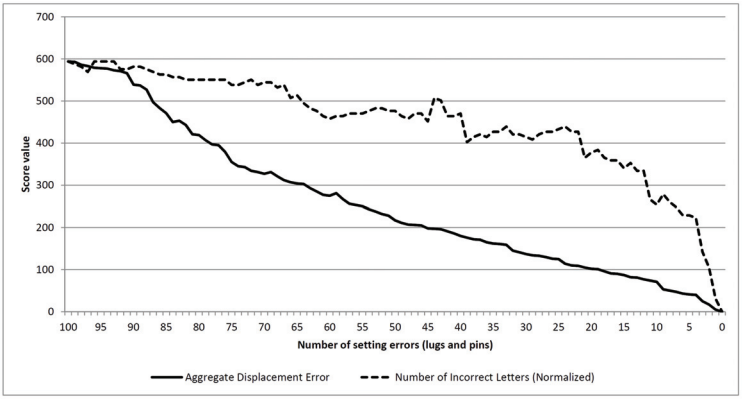


FIGURE 7.5: Hagelin M-209 – fitness-distance analysis of ADE vs. simple score

Message length	2 overlaps	6 overlaps	12 overlaps
50	66%	42%	31%
60	97%	91%	77%
65	100%	100%	93%
75	100%	100%	100%
100	100%	100%	100%

TABLE 7.7: Hagelin M-209 – performance of new known-plaintext attack with different lug overlaps

7.4.3 Evaluation

This section shows the soundness of our algorithm. At first we present its performance, which we analyzed with simulations. After that we analyze the work factor of our algorithm. Finally, we solved with this algorithm several publicly available challenges and we show the results.

7.4.3.1 Performance

We tested this algorithm with simulated cases of messages of various lengths, and encrypted with random keys settings. In the M-209 Technical Manual, a list of mandatory rules is provided, to avoid settings which may weaken the cryptographic security of the device [95]. One of the rules is that the number of lug overlaps in bars should be at least 2 and 12 at most. Therefore, we tested the algorithm using lug settings with 2, 6, and 12 lug overlaps. The number of known-plaintext letters was between 50 to 100. The results are shown in Table 7.7 and Figure 7.6. Each result is based on a sample of 100 simulation runs. A run is considered successful only if the full correct settings were recovered within an arbitrary time-out limit of 10 hours. The tests were run on a 3.4 GHz Core i7 PC with 8GB of RAM.

Key settings for messages with 65 letters or more can be easily recovered, often in less than a minute. For example, the 75 letter sample message given by Morris was solved in 5 seconds. For shorter messages, we can see that the number of lug overlaps affects the success rate. It also affects the speed of convergence of the hill-climbing algorithm, as described in more detail in Section 7.4.3.2. Therefore, the lug overlap feature of the M-209 adds somehow to the cryptographic security of the M-209, at least for this type of attack. In comparison, Morris's method requires at least 75-100 known plaintext characters, in order to fully recover the key settings.

Note that when hill climbing fails to recover the full key settings, it is nevertheless often possible to recover most of the correct settings. If such a partial solution can be achieved, for example when only a small part of the message plaintext is known, this partial solution may help in revealing additional probable words or sentences in other parts of the message, even though they may show up with some errors after decryption. By using this additional known plaintext and running the algorithm again, the full key settings can then be recovered.

The shortest length of plaintext required by the algorithm to succeed is 50, although sporadic success was achieved with shorter ciphertexts. With ciphertexts of length 40, we sometimes obtained several keys that would reproduce the correct plaintext. In Section 3.2.5, the unicity distance of the Hagelin M-209 was estimated to be 47, very close to our limit of 50. This may suggest that the performance of our known-plaintext attack is probably very close to the best performance that may be achieved by any known-plaintext attack.

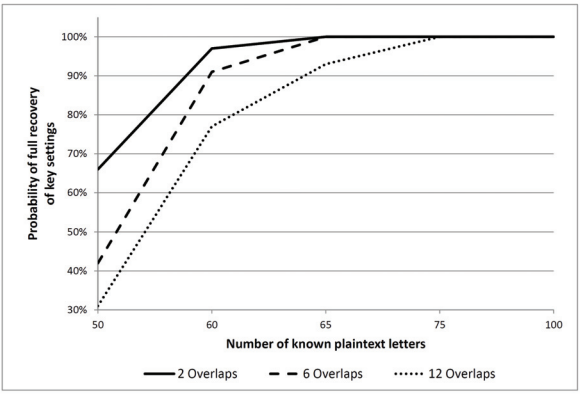


FIGURE 7.6: Hagelin M-209 – performance of new known-plaintext attack

Transformation type	Number of transformations
Toggle transformations	131
Swap transformations	$(131 \cdot 130)/2 = 8\,515$
Total:	On average about 4 300 since 50% of the possible swap transformations are irrelevant (the two pins involved must have different states).

TABLE 7.8: Hagelin M-209 – number of pin settings transformations

Transformation type	Number of transformations
Swaps of types without lug overlap	$6 \cdot 5 = 30$
Swaps of any type	$21 \cdot 20 = 420$
Total:	On average about 120 transformations (almost 30% of the total possible 450), since one of the two types involved must be non-zero.

TABLE 7.9: Hagelin M-209 – number of lug settings transformations

7.4.3.2 Analysis of Work Factor

As described in Section 7.4.2.1, each main cycle of the hill-climbing algorithm starts with a random key, i.e. with random pin and lug settings. It continuously performs loops of transformations of various types on the pin and lug settings. After each transformation, it decrypts the ciphertext with the resulting key, and computes the ADE. If the ADE has improved, the resulting new key is kept. The number of possible transformations of each type is summarized in Table 7.8 for pins and in Table 7.9 for lugs.

Hill climbing always checks first the pin settings transformations, repeating cycles of testing all such transformations as long as the ADE improves. It then checks all possible lug settings

Message length	Average number of decryptions	Minimum	Maximum
50	384000000	21000000	2925000000
75	1783000	268000	5962000
100	311000	160000	597000

TABLE 7.10: Hagelin M-209 – work factor of our new known-plaintext attack

transformations not involving lug overlaps. Last, it checks all possible lug settings transformations. The actual number of cycles for each type, as well as the number of overall start/restart cycles, varies according to the length of the message and the complexity of the key (number of lug overlaps). However, in general about 80%-90% of the transformations actually tested are pins "Swap" transformations.

In Table 7.10 we present the total number of decryptions/transformations required for recovering the full key settings, for several message lengths, using settings with 6 lug overlaps. The average, best-case and worst-case results are shown. The data is based on batches of 100 successful simulation runs per each length. A run is considered successful if the full key was recovered within 10 hours. Note that even in cases of unsuccessful runs, the majority of key settings can often be recovered.

The worst-case message with 50 letters and 6 lug overlaps requires about $3 \cdot 10^9$ or 2^{32} transformations, decryptions, and ADE computations. This can easily be computed with a single high-end PC – in contrast to a brute-force search over the complete keyspace with a size of 2^{174}. On a 3.4 GHz PC and without multithreading, for messages with 50 known-plaintext letters, about 300000 transformations and ADE computations per second can be processed. For this worst-case message, this amounts to about 3 hours. The average time to solve a message with 50 known-plaintext letters (6 lug overlaps) is approximately 20 minutes. With 100 letters, the process takes 1-2 seconds on average, and requires about 2^{19} transformations. In Figure 7.7, we present the work factor of our algorithm for additional scenarios, including 2, 6, and 12 lug overlaps. We show the number of the required transformations on a log2 scale.

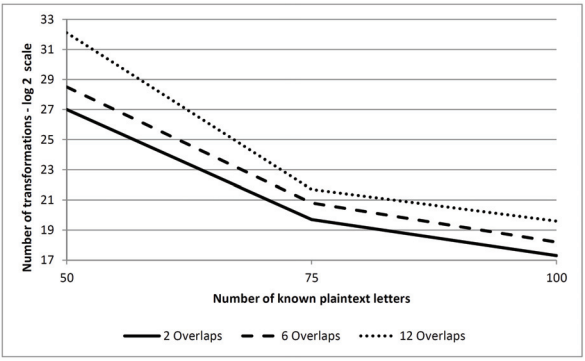


FIGURE 7.7: Hagelin M-209 – work factor of known-plaintext attack

The work factor of our method cannot be compared directly with Morris' method for recovering key settings, which is a manual method. Some elements of Morris' method, such as the computation of the displacement histograms, may be computerized. However the analyst's judgment

is required at each step, especially on short messages, to decide on which wheel to focus, which pins to mark as known or ambiguous, and when to backtrack and start again. In contrast, our method is fully automated.

7.4.4 Challenges

We also applied the method to several challenges, such as those developed by Jean-François Bouchaudy. Bouchaudy is a French amateur codebreaker who has studied the Hagelin M-209 device, its historical uses and cryptanalytical methods. On his website, he published a series of challenges with increasing difficulty, and requiring a wide range of techniques for their solution [111]. In the bonus section, there is a known-plaintext challenge (#12) with 50 letters, which was easily solved by our method. There is also a challenge with 40 letters (#14), which we also solved. This took about 10 hours on a 4-core 3.4 GHz PC and using 8 threads. The author of the challenges has requested not to publish the solutions, but more details can be found on the challenge website [111], including a mention of our results.

We also applied the technique to the 4th and most difficult of the challenges that Morris himself published [104]. This message has 60 letters, but only 50 of the corresponding plaintext letters are given. Our program was able to find the settings in about 20 minutes. The lug settings have 6 overlaps. With those settings, the second ciphertext message from this challenge could be deciphered:

"CONGRATULATIONSZGOODZBUDDYZZZZ".

7.5 A New Ciphertext-Only Attack

As described previously in Section 7.3.2.5, a 4-stage ciphertext-only attack which we previously developed required at least 750 letters in the best case in order to recover the key, and usually 1 000 or more [36]. With this method, we were able to solve challenges with messages of approximately that size [94]. We also solved a number of challenges from Bouchaudy's M-209 Challenge site with a similar number of letters [111]. Still, the last problem, challenge #12, was yet unsolved. It involved messages with 600 or fewer letters, too short for our initial method [36].

In order to overcome the $750 - 1\,000$ limit, we developed a new ciphertext-only attack, which applies the principles of our new methodology. We describe this algorithm and its performance in the next sections.

7.5.1 Description

The algorithm is based on a *hybrid nested* approach (see Section 4.3.2). An outer hill-climbing process HC_{outer} searches for optimal lug settings, and an inner simulated annealing process SA_{inner} searches for optimal pin settings. The outer process HC_{outer} is described in Algorithm 7. It starts with initial random lug settings, and iteratively tries to improve them. It systematically applies transformations on the current lug settings, to test new candidate settings in their neighborhood. At each step, in order to evaluate the score for those candidate lug settings, HC_{outer} invokes the nested SA_{inner} process, which searches for optimal pin settings *given the current*

candidate lug settings. SA_{inner} does not modify the lug settings. Given the pin settings obtained from SA_{inner} and the candidate lug settings, HC_{outer} computes a score, and uses that score to decide whether or not to accept the new lug settings.

Algorithm 7 Hagelin M-209 – ciphertext-only attack with nested HC-SA

```
1:  procedure HC_outer(C)                                   ▷ C = ciphertext
2:      (BestLugs, BestPins) ← RandomLugsAndPins()
3:      repeat
4:          Stuck ← true
5:          for CandidateLugs ∈ Neighbors(BestLugs) do
6:              CandidatePins ← SA_inner(CandidateLugs)▷ Find best pins given candidate lugs
7:              Score ← LogMonoScore(Decrypt(CandidateLugs, CandidatePins, C))
8:              if Score > LogMonoScore(Decrypt(BestLugs, BestPins, C)) then
9:                  (BestLugs, BestPins) ← (CandidateLugs, CandidatePins
10:                 Stuck ← false
11:                 break
12:     until Stuck = true
13:     return BestLugs, BestPins
```

HC_{outer} is invoked multiple times, with multiple restarts, until a readable decryption is obtained.

For scoring, we use log monograms. In our prior work [36] we found that for ciphertexts shorter than 750 letters, the method in principle with the best resilience to key errors, IC, stops being effective. This is due to spurious high scores (see Section 3.2.4), obtained with keys with almost no correct elements. Those occur too frequently to allow any effective search using IC.

On the other hand, scoring methods with high selectivity such as bigrams or trigrams also are not applicable, as they have very poor resilience to key errors. As a compromise between selectivity and resilience to errors, we found log monograms to be most effective.

To search for the lug settings (outer HC), we use a canonized representation of the lug key spaces, as described in Section 7.2.4.2. In order to survey the neighborhood of candidate lug settings, we use a Variable Neighborhood approach, as suggested in Section 4.6. This is quite necessary as the cost of evaluating candidate lug settings is high: for each candidate lug settings we need to invoke the inner SA process. Using this Variable Neighborhood approach, we first try to improve lug settings using simple transformations. Those simple transformations consist of reducing the count of one type of bar, and increasing the count of another type. When we reach a local optimum and cannot further progress using simple transformation, we instead try some complex transformations. Those consist of reducing the count of *two* types of bar, and increasing the counts of two other types. As soon as we have found a complex transformation which produces neighboring lug settings with a better score, we go back using the more simple transformations. If neither of the simple or complex transformations are able to produce better candidate lug settings, the outer HC stop, and we restart it.

The inner SA process, which searches for the optimal pin settings given the candidate lug settings, uses the following transformations:

- Toggle the state of a single wheel pin (change its state from 0 to 1, or from 1 to 0).
- Toggle the state of a pair of two pins, from the same wheel, which have different state (one is 0 and the other is 1, or vice versa).

Message length	2 overlaps	6 overlaps	12 overlaps
$<= 400$	0%	0%	0%
450	23%	3%	0%
500	100%	98%	91%
$>= 550$	100%	100%	100%

TABLE 7.11: Hagelin M-209 – performance of new ciphertext-only attack (Lasry2017)

- Toggle the state of a pair of two pins, from different wheels, which have a different state.
- Toggle the state of all the pins of a wheel.

The inner SA process is described in Algorithm 8. It runs 5 times, and returns the best pin settings it has found. Each cycle uses a cooling schedule which reduces the temperature by dividing it by 10%.

Algorithm 8 Hagelin M-209 – ciphertext-only attack – inner simulated annealing

procedure $SA_{inner}(C, Lugs)$ ▷ C = ciphertext
  $T \leftarrow T_0$
  $\alpha \leftarrow 0.9$ ▷ the cooling factor
  $BestPins \leftarrow CurrentPins \leftarrow RandomPins()$
  for $I = 1$ to 5 **do**
    for $CandidatePins \in NeighborsPins(CurrentPins)$ **do** ▷ survey all neighbors
      $Score \leftarrow LogMonoScore(Decrypt(Lugs, CandidatePins, C))$
      $D \leftarrow (Score - LogMonoScore(Decrypt(Lugs, CurrentPins, C)))$
      if $D > 0$ or $Random(0..1) < e^{-\frac{|D|}{T}}$ **then**
        $CurrentPins \leftarrow CandidatePins$ ▷ accepted
        if $Score > LogMonoScore(Decrypt(Lugs, BestPins, C))$ **then**
          $BestPins \leftarrow CandidatePins$
        break
    $T \leftarrow \alpha \cdot T$ ▷ reduce temperature
  return $BestPins$

7.5.2 Evaluation

We show the success rate of this new attack in Table 7.11. This new algorithm performs significantly better than our previous ciphertext-only attack, which was previously the state-of-the-art [36]. It can find solutions in all cases for cryptograms with only $500-550$ letters. Performance seems, however, to rapidly deteriorate with less than 500 letters.

For comparison (see Figure 7.8), our previous method (Lasry2016 in the chart) needed $750-1\,000$ letters in the best case, and most often about $1\,000-1\,250$. The Reeds-Ritchie-Morris algorithm (see Section 7.3.2.6), based on our simulations, requires $1\,250-1\,500$ letters.

The shortest length of ciphertext required by the algorithm to succeed is 450-500. While the unicity distance of the Hagelin M-209 is around 50, Reeds's extended unicity distance applied to monograms was found to be 491 (see Section 3.2.6). This may indicate that the performance

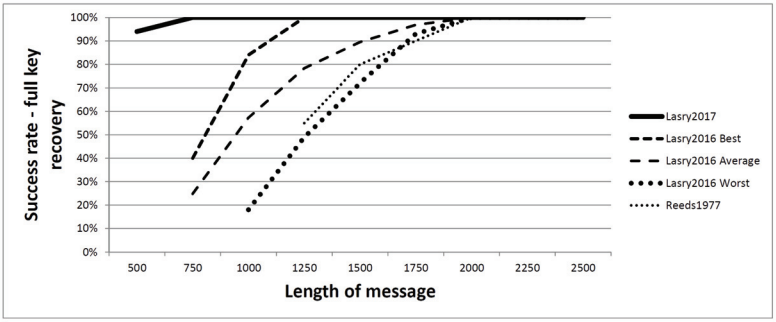


FIGURE 7.8: Hagelin M-209 – comparison with previous ciphertext-only attacks

Message length	Average decryptions
500	2^{34}
750	2^{32}
1000	2^{30}
1250	2^{29}
1500	2^{28}

TABLE 7.12: Hagelin M-209 – work factor for new ciphertext-only attack

of our new attack is very near to the best performance that may be achieved using a scoring function based on monograms.

We also analyzed the work factor, and more specifically, the number of decryptions needed by the algorithm to recover the key. The average number of decryptions for various lengths is displayed in Table 7.12. With a 10-core Intel Core i7-6950x PC and using multithreading, about 2-5 minutes are required for messages with 1 250 letters or more, 5-10 minutes with 1 000, about one hour with 750, and 10 – 20 hours with 500.

7.5.3 Solving the M-209 Challenge

With this new method, we solved challenge #12, the last problem in Bouchaudy's M-209 Challenge [111]. The algorithm independently found the key for two messages, one with 600 letters, the other one with 585. It needed 8 hours and 15 hours, respectively, on a 10-core Intel Core i7-6950x PC and using multithreading. In the lug settings, there were 10 overlaps, which is a relatively complex key. With this solution to problem #12, we also won the challenge contest [112].

7.6 Summary

The M-209 was one of the most widely used cipher machines. As such, it has been the focus of extensive research. In this chapter, we presented two algorithms for the cryptanalysis of the

Hagelin M-209, a known-plaintext attack and a ciphertext-attack, which we both developed as part of this research. Both attacks are today the state-of-the-art for this cipher machine in the public domain, and close to some theoretical limits (based on unicity distance computations).

With those attacks, we were able to find solutions for a number of public challenges, including challenge #12, the last and hardest problem in Bouchaudy's M-209 contest [111].

Both attacks illustrate the effective use of several of the principles of our new methodology, as summarized in Table 7.13 and in Table 7.14.

Principle	Application of the methodology principle
GP1	Hill climbing, parallel search for pins and lugs
GP2	Ignoring redundant lug settings
GP3	The ADE, a highly selective and resilient scoring function. Despite the large key space, there is no need for a divide-and-conquer approach, due to this powerful scoring method.
GP4	Variable neighborhood search
GP5	Multiple restarts. Random initial keys are enough when using the ADE.

TABLE 7.13: Hagelin M-209 – applying the methodology for the known-plaintext attack

Principle	Application of the methodology principle
GP1	Nested search – outer HC (for lug settings) and inner SA (for pin settings)
GP2	Ignoring redundant lug settings
GP3	Log-monograms – good tradeoff between selectivity and resilience to key errors
GP4	Variable neighborhood search
GP5	Multiple restarts

TABLE 7.14: Hagelin M-209 – applying the methodology for the ciphertext-only attack

8

Case Study – Chaocipher

Chaocipher is a manual encryption method designed by John F. Byrne in 1918. Until he passed away in 1960, Byrne fervently believed that his cipher system was unbreakable, regardless of the amount of material available to a cryptanalyst. For several decades he tried, unsuccessfully, to propose the Chaocipher to government agencies. In 1953, he exposed his Chaocipher in his autobiography, Silent Years, providing several examples of texts encrypted with Chaocipher as challenges, but without divulging the inner workings of the cipher. Those were made public only in 2010, when Byrne's family donated the entire corpus of Chaocipher papers to the National Cryptologic Museum (NCM) in Fort Meade.

In this chapter we present a ciphertext-only attack for the cryptanalysis of Chaocipher in-depth messages, which we developed along the guidelines of our new methodology (see Chapter 4). This algorithm is based on a divide-and-conquer approach and it takes advantage of a major weakness in the design of the cipher, which we uncovered. We also present a known-plaintext attack for short in-depth messages. For both attacks, we present two variants, one for the original Chaocipher, and another for an extended version of Chaocipher. All methods use highly specialized scoring methods, and hill climbing. Using one of the methods, we solved for Lou Kruh's and Cipher Deavours's alternate Exhibit 5, also known as "Exhibit 6".

The chapter is structured as follows. In Section 8.1, we provide some historical background. In Section 8.2, we describe the Chaocipher encryption system, including Byrne's classic version as well as Kruh and Deavours's extended version, and also analyze the size of the keyspace. In Section 8.3, we present prior work on the cryptanalysis of Chaocipher. In Section 8.4, we present our new methods for the cryptanalysis of Chaocipher messages in-depth. In Section 8.5, our solution for Exhibit 6 is presented. In Section 8.6, we reevaluate the security of the Chaocipher in view of those findings, with the conclusion that in its classic form, as designed by Byrne, the Chaocipher was a relatively weak cipher, despite Byrne's rather strong assertions to the contrary. Finally, in Section 8.7, we summarize our results.

The results presented in this chapter have also been published in Cryptologia [37].

8.1 Introduction

John Francis Byrne was born in 1880 in Dublin, Ireland. He was an intimate friend of James Joyce, the famous Irish writer and poet, studying together in Belvedere College and University College in Dublin. Joyce based the character named Cranly in Joyce's "A Portrait of the Artist as a Young Man" on Byrne, used Byrne's Dublin residence of 7 Eccles Street as the home of Leopold and Molly Bloom, the main characters in Joyce's "Ulysses", and made use of real-life anecdotes of himself and Byrne as the basis of stories in Ulysses. Byrne left Ireland for the United States in 1910, and was employed as a reporter, editorial writer, financial editor, and daily columnist, contributing at different times to several newspapers and publications.

In 1953 Byrne published his autobiographical "Silent Years: An Autobiography with Memoirs of James Joyce and Our Ireland" [113]. While adding much insight about the Irish struggle at the turn of the 19th century and of Byrne's intimate acquaintance with James Joyce, the primary reason for publishing the book was to be on record regarding 'Chaocipher', a cryptographic system he invented in 1918. Included in Chapter 21 [114], dedicated entirely to Chaocipher, were four exhibits (1-4) consisting of significant amounts of ciphertext and their corresponding plaintext and a challenge to readers to solve. Byrne offered a prize of $5 000 to the first one to decipher the relevant portion of Exhibit 4. No one ever claimed to having solved any of the challenges.

It is important to realize that Byrne's Chaocipher challenges differed from other cipher challenges. In most challenges, the underlying cipher system is known but the enciphering key is kept secret, in accordance with the principles formulated by Auguste Kerckhoffs. In Byrne's case, however, he kept the cipher system a total secret, only providing the reader with copious amounts of plaintext and corresponding ciphertext. Prior to the publication of his book, Byrne unsuccessfully tried for several decades to interest U.S. military and government agencies to adopt Chaocipher. Chaocipher was to take up much of Byrne's energies until his death in April 1960.

The publication of Silent Years sparked numerous efforts, mainly by members of the American Cryptogram Association (ACA) working in small groups, to reconstruct the encryption algorithm and solve the four exhibits from the book. Hypotheses ranged from Gary Knight's suggestion that it was "a crude rotor or Vernam tape system that produced a polyalphabetic cipher with a fairly long period" [115], to David Kahn's belief that is was an autokey-type cipher [1].

An important milestone for Chaocipher research was the publication in Cryptologia, in 1990, of an article entitled "Chaocipher Enters the Computer Age When its Method is Disclosed to Cryptologia Editors" [116], by Lou Kruh and Cipher Deavours. Lou Kruh had managed to find John F. Byrne's son, John Byrne, an architect living in Vermont. After resisting Kruh's attempts for many years, Byrne's son finally agreed to disclose the inner workings of Chaocipher to Kruh and Deavours. To the disappointment of Cryptologia readers, Kruh and Deavours did not disclose the Chaocipher algorithm, presumably because they were not authorized to do so by John Byrne. Their article included a new ciphertext-only challenge, known as Exhibit 5, consisting of three in-depth Chaocipher messages, but without their corresponding plaintext.

The Chaocipher algorithm was finally disclosed to the general public in June 2010 [117]. This was made possible when John's wife Pat donated the entire corpus of Chaocipher papers to the National Cryptologic Museum (NCM), in Fort Meade, Maryland. Shortly after, the key settings for Exhibits 1 and 4 were reconstructed using a known-plaintext algorithm developed

by independent researchers [118, 119], while Exhibits 2 and 3 turned out to be aberrant and non-relevant variants [120]. Still, Exhibit 5 could not be solved, despite numerous efforts by an active Chaocipher research community [121].

In May 2013, Jeff Calof, a Chaocipher researcher, visited the Chaocipher archives at NCM. In the course of his research he uncovered documents containing the solution to Exhibit 5. He also discovered that the messages in Exhibit 5, which Kruh and Deavours had published in 1990, had been encrypted using an extended version of Chaocipher, different from Byrne's original version. Calof also uncovered another challenge, also created by Kruh and Deavours but never published, which Calof renamed as "Exhibit 6". Exhibit 6 consisted of the ciphertext for fifty in-depth Chaocipher messages, without the plaintexts. The impetus for their creating this new exhibit was, as they write:

> "In a letter of 7 September 1922, Friedman responded to a previous question of Byrne's about what type of material would be needed to solve the Chaocipher system. Friedman's response was "With respect to the amount of material I consider necessary for the decipherment of your system, I would say that a series of fifty messages of approximately twenty-five words each might be sufficient, providing they were enciphered upon a machine operating in principle exactly like the one I had." We believe that Friedman meant to say "letters" instead of "words" in this last sentence. At any rate, the following 50 lines are an "in-depth" specimen of Chaocipher. The plaintext is English. Encipherment was done by computer and can thus be considered error-free."

It is important to note that while Exhibit 6 messages were encrypted using Kruh and Deavours's extended version of Chaocipher, the version of Chaocipher proposed by Byrne to Friedman did not include those modifications introduced many years after Byrne's death.

Details about Calof's findings were published in Cryptologia [122] and on The Chaocipher Clearing House website [123] in 2013. Attempts by Moshe Rubin to tackle Lou Kruh's and Cipher Deavours's Exhibit 6 challenge using hill climbing and log frequencies of monograms as the fitness function were unsuccessful [124]. To date, Exhibit 6 had not been solved.

The research described in this chapter started with an attempt to solve the last unsolved exhibit, Exhibit 6. After we solved Exhibit 6, using a mix of textual analysis and cryptanalytic techniques, we focused our efforts on new and more generic known-plaintext and ciphertext-only attacks on the Chaocipher, for the case of messages in-depth, i.e. encrypted using the same key settings. We developed methods for both the classic version of Chaocipher, developed by Byrne, and the extended version, as proposed much later by Kruh and Deavours. All the methods are based on hill climbing. Based on our findings, we reach the conclusion that Chaocipher, in its classic version, was a relatively weak cipher, and that Kruh and Deavours's enhancements seem to have increased its cryptographic security.

8.2 Description of the Chaocipher Cryptosystem

In this section, we present the working principle of Chaocipher in its original form and the extended version developed by Kruh and Deavours. We examine the autokey behavior of the cipher, and analyze the size of its keyspace.

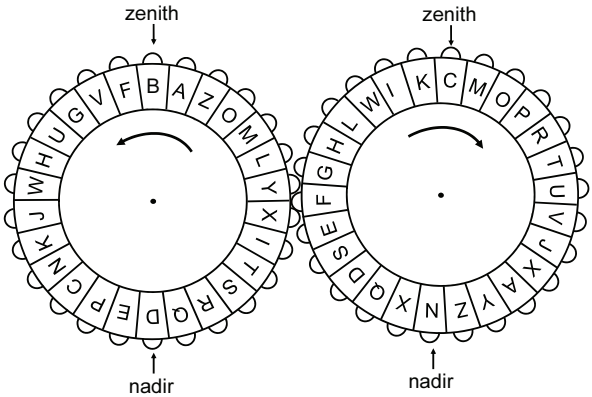


FIGURE 8.1: Chaocipher – disks in engaged mode

8.2.1 The Physical Embodiment

In Byrne's embodiment of Chaocipher, the system consists of two disks, referred to as the left and right disks, each having 26 equal sized removable tabs around its periphery. These removable tabs contain the 26 letters of the alphabet (i.e. A through Z) in some order. On the circumference of each disk are studs that allow the two disks to 'engage' or interlock. When engaged, turning one disk in one direction (e.g., clockwise) will cause the other wheel to turn in the opposite direction (e.g., counterclockwise). The tabs are removable, meaning that a tab can be removed from the periphery, another block of tabs shifted, and the extracted tab inserted into an empty space in the periphery.

At any point in time the disks can be engaged to each other so that moving one moves the other. Similarly, engaged disks can be disengaged, at which point a disk can be turned without moving the other disk. Engagement and disengagement could conceivably be performed by moving a lever up or down. Byrne had blueprints for just such a mechanical iteration; though it was never built, the blueprints are available to view on the NCM site [125].

The two disks mentioned above sit on a platform consisting of two spindles. Around each disk are two marks known as the 'zenith' and the 'nadir'. The zenith can be thought of 12 o'clock on an analog clock, while the nadir is 6 o'clock. Figure 8.1 shows what Chaocipher might look like when assembled.

It is important to note Byrne's classic convention that the right disk is used for finding the plaintext letter, while the left disk is used for finding the corresponding ciphertext letter. Lou Kruh and Cipher Deavours used a modified version of Chaocipher, in which it is possible to alternate between locating the plaintext letter in the right or left disk based on some prearranged pattern. We describe the "classic" method used by Byrne in the following section (Section 8.2.2). We present the extended method, which Kruh and Deavours used to encrypt the messages of Exhibits 5 and 6, in Section 8.2.3.

8.2.2 The Classic Chaocipher Algorithm

Byrne's original Chaocipher algorithm, referred in this chapter as the "classic" Chaocipher system, was used by Byrne to encipher Exhibit 1 and 4 (Exhibit 2 and 3 made use of different algorithms).

We present here a high-level description of the classic Chaocipher enciphering algorithm, and we provide more details in the following subsections. The enciphering algorithm consists of the following steps:

1. Generate and apply the key settings. This includes the left and right alphabets, and aligning them relative to each other.
2. For each letter in the plaintext:
   (a) Encrypt the plaintext letter by locating the plaintext letter in the right alphabet, using the corresponding letter in the left alphabet as the ciphertext letter.
   (b) Permute the right alphabet.
   (c) Permute the left alphabet.
   (d) Repeat (a) to (c) until all plaintext letters have been encrypted.

We also illustrate the algorithm by using the same steps Byrne used to encipher the plaintext of Exhibit 1, which starts with

"ALLGOODQQUICKBROWNFOXESJUMPOVERLAZYDOGTOSAVETHEIRPARTYW".

Generate and apply key settings

In the classic version of Chaocipher, the key settings consist of three items: (a) the alphabet on the right disk (the "right alphabet"), used to match or retrieve plaintext letters, (b) the alphabet on the left disk (the "left alphabet"), used to match or retrieve ciphertext characters, and (c) their initial alignment.

In his unpublished papers [126], Byrne describes a method to generate left and right alphabets from a keyphrase. This method was also used by Kruh and Deavours, and is described in Appendix 2. The alphabets may also be selected randomly, as long as the sender, who encrypts the plaintext, and the recipient, who needs to decrypt the ciphertext, use the same key settings. Chaocipher is a symmetric cipher in the sense that the same key settings are used for encryption and decryption.

To illustrate the encryption process, we use the settings used by Byrne for Exhibit 1, with the following left/right alphabets:

```
Left Alphabet:  BFVGUHWJKNCPEDQRSTIXYLMOZA
Right Alphabet: CMOPRTUVJXAYZNBQDSEFGHLWIK
```

Of equal importance is to decide how the two alphabets will be aligned or juxtaposed relative to each other. In our example, 'B' in the left alphabet is initially aligned with 'C' in the right alphabet.

Encrypt the plaintext letter

Determining the ciphertext letter consists of locating the plaintext letter in the right alphabet and noting the corresponding ciphertext letter in the left alphabet.

In our example, the first plaintext letter is 'A'. Locating the letter 'A' in the right alphabet, we see that the corresponding letter in the left alphabet is 'C', which is our ciphertext letter.

Permute the right alphabet

Permuting the right alphabet consists of the following steps:

1. Shift the entire right alphabet cyclically so that the letter just enciphered on this alphabet is positioned at the zenith (position 1).
2. Now shift the entire alphabet one more position to the left (i.e. the leftmost letter moves cyclically to the far right), moving a new letter into the zenith position.
3. Extract the letter in this alphabet at position 3, taking it out of the alphabet, temporarily leaving an unfilled 'hole'.
4. Shift all letters beginning with position 4 up to, and including, the nadir (position 14), moving them one position to the left.
5. Insert the just-extracted letter into the nadir (position 14).

In our example, permuting the right alphabet consists of the following steps:

```
00000000011111111112222222
12345678901234567890123456
Z===========N============
CMOPRTUVJXAYZNBQDSEFGHLWIK   (starting right alphabet)
AYZNBQDSEFGHLWIKCMOPRTUVJX   (plaintext letter A brought to zenith)
YZNBQDSEFGHLWIKCMOPRTUVJXA   (alphabet shifted one position left)
YZ.BQDSEFGHLWIKCMOPRTUVJXA   (letter N extracted from position 3)
YZBQDSEFGHLWI.KCMOPRTUVJXA   (block 4-14 shifted to the left)
YZBQDSEFGHLWINKCMOPRTUVJXA   (extracted letter N inserted at nadir)
```

Permute the left alphabet

Permuting the left alphabet, is similar to that of the right alphabet, with small but significant differences:

1. Shift the entire left alphabet cyclically so that the letter just used on this alphabet is positioned at the zenith (i.e. position 1).
2. Extract the letter found in this alphabet at position 2 (i.e. the letter to the right of the zenith), taking it out of the alphabet, temporarily leaving an unfilled 'hole'.

3. Shift all letters in positions 3 up to, and including, the nadir (position 14), moving them one position to the left.

4. Insert the just-extracted letter into the nadir position (position 14).

In our example, permuting the left alphabet consists of the following steps:

```
00000000011111111112222222
12345678901234567890123456
Z==========N===========
BFVGUHWJKNCPEDQRSTIXYLMOZA   (starting left alphabet)
CPEDQRSTIXYLMOZABFVGUHWJKN   (ciphertext letter 'C' brought to zenith)
C.EDQRSTIXYLMOZABFVGUHWJKN   (letter P extracted from position 2)
CEDQRSTIXYLMO.ZABFVGUHWJKN   (block 3-14 shifted to the left)
CEDQRSTIXYLMOPZABFVGUHWJKN   (extracted letter P inserted at nadir)
```

Enciphering the next plaintext letters

To encipher the next plaintext letter ('L'), repeat the steps above. Repeating the process for the whole plaintext phrase produces the first 55 ciphertext characters of Exhibit 1:

```
plaintext:  ALLGOODQQUICKBROWNFOXESJUMPOVERLAZYDOGTOSAVETHEIRPARTYW
ciphertext: CLYTZPNZKLDDQGFBOOTYSNEPUAGKIUNKNCRINRCVKJNHTOAFQPDPNCV
```

Decrypting a ciphertext message

The process for decrypting a ciphertext message is almost identical, except that in Step 2.a we decrypt a ciphertext letter by locating the letter to be deciphered in the left alphabet, noting the corresponding plaintext letter from the right alphabet.

8.2.3 Kruh and Deavours's Extended Chaocipher Algorithm

Kruh and Deavours used an extended Chaocipher algorithm to encipher Exhibits 5 and 6. The extended enciphering algorithm is very similar to Byrne's classic algorithm, differing in one significant way. Rather than the plaintext letter to encipher always being located in the right alphabet, the letter may be located in either the right or the left alphabet, based on a predetermined sequence, which Kruh and Deavours named "takeoff pattern for message". We simply denote this pattern as the "takeoff pattern". The idea behind the use of a takeoff pattern and allowing both left and right alphabets to be used either for plaintext or ciphertext, was drawn from Byrne's method for generating alphabets, described in Appendix 2. It is not clear why Kruh and Deavours introduced this extension, but as we show in the next sections, this extension adds to the cryptographic security of the system.

The high-level description of the extended Chaocipher enciphering algorithm is as follows (the differences from the classic version are marked in **bold**):

1. Generate and apply the key settings. This includes the left and right alphabets, aligning them relative to each other, **and selecting the takeoff pattern**.
2. For each letter in the plaintext:
   (a) **According to the takeoff pattern determine which disk/alphabet, left or right, is to be used to match the current plaintext letter, the other disk being used to retrieve the corresponding ciphertext letter.**
   (b) Encrypt the plaintext letter by locating the plaintext letter in the alphabet selected for matching the plaintext letter, and retrieving the corresponding ciphertext letter in the other alphabet.
   (c) Permute the right alphabet.
   (d) Permute the left alphabet.
   (e) Repeat (a) to (d) until all plaintext letters have been encrypted.

We use Kruh and Deavours's enciphering of Exhibit 5, message 3 as an example. This message starts with "WECANCONCEIVETHAT...". In their exhibits 5 and 6, Kruh and Deavours used the process proposed by Byrne to generate left and right alphabets from keyphrases, as described in Appendix 2. The left and right alphabets they generated for Exhibit 5, and their alignments were as follows:

```
Left Alphabet:  HNPJRBMYAZTGVKQDIWLOXSFECU
Right Alphabet: XYCWPIEMTKGABHUDQLVRFSJOZN
```

Note that when using the extended version of Chaocipher, it is also possible to use randomly generated alphabets.

For their Exhibit 5, they selected the following predetermined 63-letter takeoff pattern:

```
RLRLRLLLRLRRRRLRLRLRLLLRLRRRRLRLLRLRLLLRLRLRRRLRLRLRRRRRLRLRLRL
```

In the takeoff pattern, the letters 'L' and 'R' denote the left and right alphabets, respectively. The first letter in the takeoff pattern is 'R': this should be understood that the first plaintext letter should be located in the right alphabet, with the left alphabet being used to retrieve the ciphertext letter. Similarly, the second letter in the pattern is 'L', which means that the second plaintext letter should be found in the left alphabet, with the right alphabet serving as the ciphertext alphabet, and so on. When the takeoff pattern is exhausted, we start again from the beginning of the pattern.

Apart from this, the encryption process is similar to the encryption process with the classic Chaocipher algorithm. The permutations are also identical. We illustrate the process with our example, message 3 of Exhibit 5 ("WECANCONCEIVETHAT...").

1. As the first element of the takeoff pattern is 'R', we locate the first plaintext letter ('W') in the right alphabet. From the left alphabet, we retrieve the ciphertext letter at the corresponding position ('J'). We then permute the left and the right alphabets. Note that the left alphabet is always permuted in the usual way, as described in the previous section, regardless of its being used to locate a plaintext or ciphertext letter.

2. To encipher the second plaintext letter ('E'), we now use the left alphabet, as the second element of the takeoff pattern is 'L', retrieve the corresponding character from the right alphabet, and permute the alphabets.

3. After repeating those steps for all plaintext letters, we have enciphered Byrne's plaintext phrase, producing the first 17 characters of Exhibit 5 message 3:

```
plaintext:  WECANCONCEIVETHAT...
ciphertext: JZHASQNRTKTTLZDYO...
```

Deciphering a message encrypted using the extended version of Chaocipher is similar, except that at each step we first need to locate the ciphertext letter, using the alphabet other than the one indicated by the takeoff pattern, and retrieve the corresponding plaintext character from the alphabet indicated by the pattern. Again, the permutations of the right and left alphabets are always the same, regardless of their use.

8.2.4 Deriving Alphabets from Keyphrases

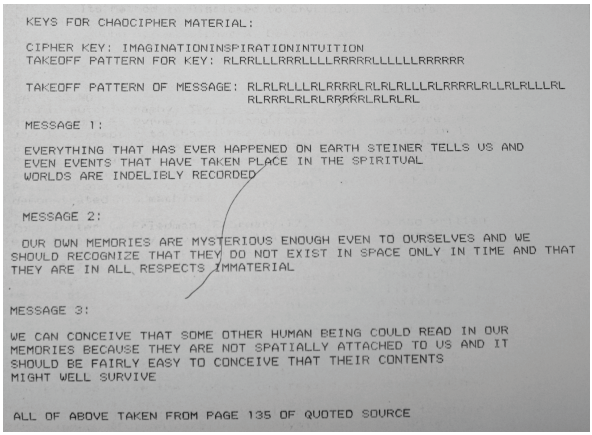
In his unpublished papers [126], Byrne's describes a method he used to generate left and right alphabets from keyphrases, as an alternative to generating random alphabets which may be difficult to memorize and communicate. Byrne used this method to create the alphabets for his Exhibit 4. Kruh and Deavours also used the same method to generate alphabets for their exhibits. It should be noted, however, that this method also exposes the cipher system to dictionary attacks, although dictionary attacks are more difficult when long keyphrases are used, rather than shorter keywords.

This method for generating alphabets from keyphrases is based on the same encryption algorithm used for the extended version of Chaocipher. In fact, Kruh and Deavours developed their extended algorithm based on Byrne's method to generate alphabets. Kruh and Deavours differentiate between the takeoff pattern used for generating the alphabets, which they refer as the "Takeoff Pattern for Key", and the takeoff pattern used for encrypting a message, referred to as the "Takeoff Pattern of Message".

We describe here the process for generating alphabets from keyphrases, with the settings used by Kruh and Deavours in Exhibit 5. We start with two straight left and right alphabets ("ABCDEFGHIJKLMNOPQRSTUVWXYZ") and the takeoff pattern (referred to as the "Takeoff Pattern for Key"), "RLRRLLLRRRLLLLRRRRRLLLLLLRRRRR", to be used for generating the new alphabets. We do so by using the extended Chaocipher encryption algorithm, and encrypting the keyphrase "IMAGINATIONINSPIRATIONINTUITION" (referred to as the "cipher key"). The newly generated key settings consist of the final left and right alphabets remaining after enciphering the entire phrase, "NPJRBMYAZTGVKQDIWLOXSFECU" and "XYCWPIEMTKGABHUDQLVRFSJOZN" respectively, their resulting alignment ('N' aligned with 'X'), and of another takeoff pattern, "RLRLRLLLRLRRRRLRLRLRLLLRLRRRRLRLLRLRLLLRLRLRRRLRLRLRRRRRLRLRLRL" (the "Takeoff Pattern of Message").

The full settings used to generate the alphabets for encrypting the messages of Exhibit 5 are shown in Figure 8.2 [122].

Below, we show the steps in the encryption process (using the extended Chaocipher encryption algorithm), which generate the new key settings (left and right alphabets):


KEYS FOR CHAOCIPHER MATERIAL:

CIPHER KEY: IMAGINATIONINSPIRATIONINTUITION
TAKEOFF PATTERN FOR KEY: RLRRLLLRRRLLLLRRRRRLLLLLLRRRRRR

TAKEOFF PATTERN OF MESSAGE: RLRLRLLLRLRRRRLRLRLRLLLRLRRRRLRLLRLRLLLRL
RLRRRLRLRLRRRRRLRLRLRL

MESSAGE 1:

EVERYTHING THAT HAS EVER HAPPENED ON EARTH STEINER TELLS US AND
EVEN EVENTS THAT HAVE TAKEN PLACE IN THE SPIRITUAL
WORLDS ARE INDELIBLY RECORDED

MESSAGE 2:

OUR OWN MEMORIES ARE MYSTERIOUS ENOUGH EVEN TO OURSELVES AND WE
SHOULD RECOGNIZE THAT THEY DO NOT EXIST IN SPACE ONLY IN TIME AND THAT
THEY ARE IN ALL RESPECTS IMMATERIAL

MESSAGE 3:

WE CAN CONCEIVE THAT SOME OTHER HUMAN BEING COULD READ IN OUR
MEMORIES BECAUSE THEY ARE NOT SPATIALLY ATTACHED TO US AND IT
SHOULD BE FAIRLY EASY TO CONCEIVE THAT THEIR CONTENTS
MIGHT WELL SURVIVE

ALL OF ABOVE TAKEN FROM PAGE 135 OF QUOTED SOURCE

FIGURE 8.2: Chaocipher – Exhibit 5 settings

```
Left starting alphabet:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
Right starting alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Left zenith: A
Right zenith: A
Takeoff pattern for key: RLRRLLLRRRLLLLRRRRRLLLLLLRRRRRR
Keyphrase (Cipher Key):  IMAGINATIONINSPIRATIONINTUITION

( 0) leftAlphabet:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
( 0) rightAlphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ
( 0) Plain disk is the RIGHT disk, pt(I) = ct(I)

( 1) leftAlphabet:  IKLMNOPQRSTUVJWXYZABCDEFGH
( 1) rightAlphabet: JKMNOPQRSTUVWLXYZABCDEFGHI
( 1) Plain disk is the LEFT disk, pt(M) = ct(N)

( 2) leftAlphabet:  MOPQRSTUVJWXYNZABCDEFGHIKL
( 2) rightAlphabet: OPRSTUVWLXYZAQBCDEFGHIJKMN
( 2) Plain disk is the RIGHT disk, pt(A) = ct(Y)

...
...
...

( 29) leftAlphabet:  AZTVQDIWLOXSFECUGHKNPJRBMY
( 29) rightAlphabet: EMTKABUDQLVRFSJOZNGXYHCWPI
( 29) Plain disk is the RIGHT disk, pt(O) = ct(U)

( 30) leftAlphabet:  UHKNPJRBMYAZTGVQDIWLOXSFEC
( 30) rightAlphabet: ZNXYHCWPIEMTKGABUDQLVRFSJO
( 30) Plain disk is the RIGHT disk, pt(N) = ct(H)

( 31) leftAlphabet:  HNPJRBMYAZTGVKQDIWLOXSFECU
( 31) rightAlphabet: XYCWPIEMTKGABHUDQLVRFSJOZN

Encrypted output: INYDNRGNACLJKSOOXHXPTQRVIXZQAUH (not used for any purpose)
```

8.2.5 Autokey Behaviour of Chaocipher

The alphabets on the left and right disks are changed after every encryption (or decryption) step, using the permutation rules described earlier. Furthermore, the exact permutations, at each step, primarily depend on the last plaintext or ciphertext characters (and in the case of the extended version of Chaocipher, also on the takeoff pattern). Therefore, the encryption key after N steps, to be applied to character N in the plaintext or ciphertext, is determined by a combination of the initial key settings, and of the first N characters of the plaintext (or ciphertext in the case of decryption). This effectively turns Chaocipher into a special form of an autokey cipher.

In theory, a cipher with an autokey mechanism results in ciphertext sequences for which it is more challenging to detect statistical patterns, as it hides repetitions, within the same message, or within several messages in-depth. On the other hand, the main drawback of any autokey scheme is the fact that any error in transmission or reception, as well as in the encryption or decryption process, at any stage, is likely to result in the inability to decrypt and read the remainder of the message.

The cryptologic literature abounds with methods of solving autokey ciphers using isomorphic sequences [127]. In the case of Chaocipher, the system does not generate isomorphs [128]. Chaocipher is autokey-like in that ciphertext is highly coupled to previous plaintext and ciphertext, but the nature of the ciphertext generated differs significantly from ciphertext generated by other classic autokey ciphers due to the ongoing alphabet transformations. While Byrne may not have been aware of the isomorph method, he considered the lack of any repetitions in Chaocipher to be the true measure of its security, as demonstrated by his writings.

8.2.6 Analysis of the Keyspace

In the case of Byrne's classic version of Chaocipher, the keyspace consist of the combinations of all possible starting left and right alphabets. The total number of such combinations is $(26!)^2$. This number must be divided by 26, as both disks may be rotated – in sync – to any position before starting encryption, without affecting the encryption process (Chaocipher is invariant under rotation). Therefore, the size of the keyspace is $\frac{(26!)^2}{26} \approx 2^{173}$. This number is approximately equivalent to the keyspace size of a 3DES cipher with 3 different keys.

In the case of the Kruh and Deavours's extended version of the ciphers, this number must be multiplied by the number of possible takeoff patterns. Since the cryptanalyst does not know the length of the takeoff pattern, he can only speculate about its maximum length. Given a takeoff pattern with a maximum length of N elements, there are $2^1 + 2^2 + ... + 2^N$ possible sequences, or approximately $2^{(N+1)}$. Therefore, the total keyspace size is approximately $2^{(174+N)}$, N being the maximum expected length of the takeoff pattern.

8.3 Related Work – Prior Cryptanalysis

Until 2010, all cryptanalytic work on Chaocipher was focused on reconstructing the algorithm given the plethora of plaintext and ciphertext sequences. All attempts to do so did not succeed. Once the algorithm was revealed in 2010, work shifted from the reconstruction of the algorithm to the solution of the exhibits, and included the development of known-plaintext and ciphertext-only attacks.

To date, the only cryptanalytic method developed for reconstructing Chaocipher starting alphabets is a known-plaintext attack used to determine the initial left and right alphabets given a sufficient amount of ciphertext and its corresponding plaintext [118, 129].The method reconstructs the initial alphabets by beginning with empty left and right alphabets, and building them up by inserting plaintext and ciphertext pairs of letters into the alphabets, testing each possible location in the alphabet, permuting the alphabets after each insertion, and backtracking if a conflict is detected. We illustrate here the first steps for the following matching plaintext and ciphertext, taken from Exhibit 1, at positions from 7173 to 7226 in the original ciphertext/plaintext:

```
             |7173.....................|7200.....................|7226
             .......8.........9.........0.........1.........2......
             34567890123456789012345678901234567890123456789012345678
             ==========================================================
Ciphertext: DCQMVFLCBBYKBM**E**SGHSOEPSKPKEWMEQWCOQNBURIIQBNQOGAAXPEIT
Plaintext:  CHISNOWTHENECE**S**SITYWHICHCONSTRAINSTHEMTOALTERTHEIRFORM
```

We start with initial empty alphabets, and first process the pair of ciphertext and plaintext letters at position 7187, in our case E and S respectively. We align E in the left alphabet on top of S in the right alphabet and apply the permutations described in the previous section for the left and the right alphabets. Full details about this example may be found in [118], including why we start at this position. After processing the pair at position 7187, we obtain the following alphabets:

```
          00000000011111111112222222
          12345678901234567890123456
          Z===========N============
Left: **E**????????????????????????
Right:????????????????????????**S**
```

Next, we process the pair at position 7188, in our case S (ciphertext) and S (plaintext). Since plaintext letter S already appears in the right alphabet, we simply insert the ciphertext letter S on top of it, and apply the permutations on both alphabets, obtaining:

```
          00000000011111111112222222
          12345678901234567890123456
          Z===========N============
Left: **S**???????????**E**???????????
Right:????????????????????????**S**
```

Next, we process the pair at position 7189, G (ciphertext) and I (plaintext). The letter G did not appear yet in the left alphabet, and the letter I did not appear before in the right alphabet. Therefore, we need to check all possible positions for that pair, in which we can align G on top of I. There are 23 such free positions. At each iteration, after we have assumed a certain position and performed the permutations on the alphabets, we process the next pair, H and T, and so on, until we either run into a conflict and need to backtrack, or we have determined the full alphabet.

In order to reduce the number of combinations to check, which may rise exponentially after each step, we prefer to process a sequence of ciphertext, which includes as many repetitions as possible, matching a plaintext also with many repetitions. Carl Scheffler provides guidelines

on how to select the optimal sequence [118, 119]. He also shows that it is possible to process pairs not only forward, as illustrated above, but also backward. For example, after processing pairs on the right of the starting pair E and S, we may process the pair M and S on its left at position 7186, then the pair B and C further on the left, at position 7185. It is also possible to alternate between processing pairs on the right or on the left side. The purpose is to minimize the number of combinations to check by relying on repetitions which help in generating unique assignments. More details are available in the references [118, 129].

With an optimal and rather fortuitous plaintext/ciphertext sequence, the alphabets can be reconstituted with as few as 50-55 contiguous matching ciphertext and plaintext characters. Usually, about 80 to 100 matching characters are needed. This known-plaintext attack, however, is limited to the cryptanalysis of a relatively long single message, encrypted using Byrne's classic version of the cipher, for which the original plaintext or some part of it is known. Neither this method, nor any other method published so far, addresses the more generic case of a ciphertext-only attack. Furthermore, it is not applicable to the case of (possibly shorter) messages in-depth, nor does it take advantage of such depth. In addition, it is limited to the classic version of Chaocipher, and not applicable to Kruh and Deavours's extended method.

8.4 New Attacks for Chaocipher Short Messages In-Depth

The focus of this work is the cryptanalysis of Chaocipher messages in-depth, i.e. encrypted using the same initial key settings. From the operational perspective this scenario is very likely, unless the key settings are changed for each message, a procedure that Byrne did not suggest. In such a case, a strong cipher system is expected to be resilient to cryptanalytic attacks on messages in-depth. This may have been the reason why Friedman asked Byrne to provide him with a series of messages in-depth, as the basis for evaluating the cryptographic strength of the system. For some reason, Byrne never fulfilled Friedman's request [129].

In this section we describe new known-plaintext and ciphertext-only attacks for the cryptanalysis of Chaocipher in-depth messages, for the case of Byrne's classical Chaocipher encryption method, as well as for Kruh and Deavours's extended method.

8.4.1 Common Building Blocks

All the new methods we present in this chapter are based on hill climbing. In all cases, the methods are applied to a series of messages encrypted in depth, using the same key settings. The new methods we describe here differ mainly with regards to the scoring method used, and whether transformations are applied on only one alphabet or on both alphabets. We describe here the possible transformations on the key settings at each step of hill climbing, as well as the scoring methods.

The transformations applied by the algorithm include:

1. **Left Alphabet Simple Swaps**: Swap elements i and j in the left alphabet.
2. **Right Alphabet Simple Swaps**: Swap elements i and j in the right alphabet.
3. **Left and Right Same Index Swap**: Swap elements i and j in the left alphabet, and swap elements i and j in the right alphabet.

When checking the transformations as described in the algorithm above, we check **all** combinations of i and j so that $i < j$, and not just random combinations of i and j. In comparison, experiments in which only a subset of all possible swap transformations was tested resulted in much lower success rates. This is aligned with the guidelines of our methodology (see Section4.6).

As Chaocipher is essentially an autokey cipher (see Section 8.2.5), it has a strong element of diffusion. Therefore, any change(transformation) on a candidate key, even a minor one, will be highly disruptive, when looking at the whole text obtained by decrypting a ciphertext with the modified key. For example, when using some n-gram scoring method, or even IC, any change in the key may significantly affect the score. This results in a non-smooth neighboring landscape (see 4.6), and a HC or SA algorithm that relies on such scoring functions applied on the whole decrypted text will not be able to converge towards the correct key.

Instead of using scoring methods which apply to the whole decrypted text, we instead use scoring methods that apply only to the first n letters of each message, or give strong precedence to the initial letters, usually, no more than the 12 first letters. Since it is not feasible to build any reliable statistical measure on just 12 or fewer letters, we need to combine initial letters from several ciphertexts. Therefore, all our algorithms require a certain amount of messages in "depth" (encrypted with the same key).

Along the lines of the methodology recommendations for adaptive scoring (see Section 4.5), the scoring methods we employ are highly specialized and tailored to the Chaocipher problem, as follows:

1. **Partial IC** for ciphertext-only attacks: This partial index of coincidence is computed on the putative plaintext, obtained by concatenating the first n letters of all messages, decrypted with the candidate key. The number n of letters used for that purpose depends on the specific attack. This partial IC is the basis for a powerful divide-and-conquer attack, described in Section 8.4.2.
2. **Plaintext-Ciphertext Weighted Match** for known-plaintext attacks. To compute this score, we separately score the plaintext-ciphertext character matches for each message and its pair of associated (known) plaintext and ciphertext, and sum up their scores. For each message:
   (a) First, decrypt the ciphertext using the candidate key settings.
   (b) Compare the characters of the putative plaintext with those of the known plaintext, at all positions. If the character at position i in the putative plaintext matches the known plaintext character at the corresponding position (i.e. at position i), add $\frac{1000}{i}$ points to the score. This effectively assigns a higher value to matches at the beginning of the text, as an early decryption error is most likely to result in the wrong decryption of the rest of the message.

All algorithms assume that the takeoff pattern is known. For the classic version of Chaocipher, it assumes the sequence is "RRRRRR....", i.e. the right alphabet is always used to match plaintext characters (and the left alphabet to match ciphertext characters). When hill climbing is applied to Kruh and Deavours's extended version of Chaocipher, a takeoff pattern must be specified as input for the hill-climbing algorithm.

8.4.2 Ciphertext-Only Attack – Classic Chaocipher

In this section, we present a ciphertext-only attack for in-depth messages, encrypted using the classic Chaocipher version (without a takeoff pattern). This version of Chaocipher always uses the right alphabet to match plaintext characters, and the left alphabet to match ciphertext characters. This ciphertext-only algorithm for in-depth messages takes advantage of a major weakness we uncovered, which we describe here.

We denote as C the ciphertext which has been obtained by encrypting a plaintext P, using a key composed of a left alphabet KL, and a right alphabet KR. We want to evaluate two special candidate key settings, K_1 and K_2, which only differ by their right alphabet (KR_1 and KR_2, respectively), while their left alphabets, KL_1 and KL_2, are identical, i.e. $KL_1 = KL_2$. To do so, we decrypt the ciphertext C using $K_1 = \{KL_1, KR_1\}$ and obtain a putative plaintext P_1. Similarly, we decrypt the ciphertext C using $K_2 = \{KL_2, KR_2\}$ and obtain a putative plaintext P_2. We show here that in such a case of candidate keys K_1 and K_2, the corresponding putative plaintext P_1 can be obtained by applying a **simple substitution** on P_2, and vice-versa. In other words, P_1 and P_2 are isomorphic with regards to a simple substitution.

With the classic version of Chaocipher, all ciphertext characters are always matched against characters in the left alphabet, and plaintext characters are always retrieved from the right alphabet. The permutations applied at any step of the Chaocipher decryption process, on both the left and the right alphabets, therefore, depend only on the ciphertext characters and their matching against characters of the left alphabet, i.e. their location on the left alphabet at the relevant step of decryption. For the case of our special keys K_1 and K_2, the initial right alphabets KR_1 and KR_2, differ, while the left initial alphabets are identical. Thus, in every step of the decryption, the same exact permutations are applied on the right alphabet, either when decrypting C with K_1 or when decrypting C with K_2. Therefore, the series of the right alphabets obtained at each decryption step when decrypting C with K_1, is isomorphic (with regards to a simple substitution), with the series of the right alphabets obtained when decrypting C with K_2. As a result, the putative plaintexts P_1 and P_2, obtained by decrypting C with K_1 and K_2, are also isomorphic with regards to the same simple substitution. Finally, the Index of Coincidence of the resulting P_1 putative text, is equal to the Index of Coincidence of the resulting P_2 putative text, i.e $IC(P_1) = IC(P_2)$. Since $IC(P_1) = IC(P_2)$ for any $K_1 = \{KL, KR_1\}$ and $K_2 = \{KL, KR_2\}$, this is also true in case KL is the correct initial left alphabet used for encrypting the original plaintext P, i.e. $IC(P) = IC(P_1) = IC(P_2)$.

A major implication of these findings is that the Index of Coincidence of the plaintext obtained by decrypting a ciphertext using a candidate key, depends only on the initial left alphabet of this candidate key. We can, therefore, implement a powerful divide-and-conquer attack. We can search for the correct left alphabet separately from the search for the correct right alphabet, using hill climbing and the Index of Coincidence as the scoring method. We simply set the right alphabet to "ABC...Z" and do not modify it during hill climbing. During hill climbing, we only apply transformations on the left alphabet. The left alphabet resulting in the highest resulting IC, is most likely to be the correct one, provided we have long enough ciphertext material. After we have found the correct left alphabet, we decrypt the ciphertext using this alphabet, and "ABC...Z" as the right alphabet. All we need now is to solve a simple substitution cipher. The key for this simple substitution is also the correct right alphabet.

Note that the equality $IC(P) = IC(P_1) = IC(P_2)$ given any $K_1 = \{KL, KR_1\}$ and $K_2 = \{KL, KR_2\}$, does not hold for the extended version of the Chaocipher, developed by Kruh and Deavours. If the takeoff pattern is not "RRRR.....", then at various steps of the decryption process, ciphertext

characters may be matched against either the left or the right alphabet, depending on the takeoff pattern. Therefore, the ciphertext affects the permutations of both alphabets, left and right, and this divide-and-conquer attack does not work. We address the case of the extended version of Chaocipher in the next section.

Still, for the classic version, we cannot simply apply this algorithm to a single ciphertext, even if it is very long. Any wrong element in a candidate left alphabet, almost invariantly results in a wrong left alphabet at the next step of decryption as well as at the following steps, in corrupted decryptions and in unreliable IC scores. This is a major issue as hill climbing heavily relies on the process of incrementally improving partially correct settings. Consider the extreme case of a left alphabet almost completely correct, but unfortunately, the few errors actually affect the first ciphertext letters. In such a case, this almost complete knowledge of the settings is practically useless. Therefore, when only one message is available, this attack does not work.

We can, however, take advantage of the fact that we are processing messages in depth, if we take into account only the initial n characters of each message, using the Partial IC score described in Section 8.4.1. The shorter the message, or the part of the message (n first letters) being processed, the better the chance to obtain more correct decryptions using partially correct keys. We still need enough material in order to measure a statistically significant IC score (usually, a few hundreds of characters). We ran several experiments and it turned out that the best balance between reducing the chance of corruption by propagation of early errors, and the need for enough material, is to process the first $n = 8$ characters of each message. With $n < 5$, a much higher number of in-depth messages were required. With $n > 12$ or more, the algorithm could not converge in most cases.

The ciphertext-only algorithm to recover the left alphabet is listed in Algorithm 9.

Algorithm 9 Chaocipher – ciphertext-only attack – classical version

```
1:  procedure RecoverLeftAlphabet(CinDepth, n)   ▷ CinDepth = ciphertexts in depth, n = number of initial letters
2:      BestLeft ← ComputeInitialLeft(1000)
3:      Stuck ← false
4:      while Stuck = false do
5:          Stuck ← true
6:          for NewLeft ∈ Neighbors(BestLeft) do          ▷ Iterate over all transformations
7:              if PartialIC(NewLeft, CinDepth, n) > PartialIC(BestLeft, CinDepth, n) then
8:                  BestLeft ← NewLeft                    ▷ Found better left alphabet
9:                  Stuck ← false
10:     return BestLeft
```

Along the guidelines of the methodology (see Section 4.3), the initial key is computed by generating 1 000 random left alphabets, and selecting the one with the maximum Partial IC score. In addition, we employ multiple restarts.

As shown in the Table 8.1, this attack usually requires 60 to 80 messages in-depth, from which we process only the first $n = 8$ characters. Note that it is also possible to apply the same attack on 60-80 very short messages having only 8 characters each, although this is not a very likely operational scenario.

For reference, the size of the keyspace of the classic version of Chaocipher is 2^{173}. It is also interesting to compare those results to the amount of material that Friedman requested from

Number of messages	Probability of success	Work factor (average decryptions)
50 or less	Sporadic	
60	67%	$60 \cdot 463M \approx 2^{35}$
80	97%	$80 \cdot 160M \approx 2^{34}$
100	100%	$100 \cdot 3.7M \approx 2^{29}$
120	100%	$120 \cdot 1.9M \approx 2^{26}$

TABLE 8.1: Chaocipher – performance of ciphertext-only attack for classical version

Byrne in order to evaluate Chaocipher, namely 50 messages of about 25 characters, encrypted with the same key, amounting to 1250 characters in total. For comparison, our algorithm requires more messages, but those messages may be very short, and the total amount of required material is 600 or 700 characters.

8.4.3 Ciphertext-Only Attack for the Extended Chaocipher Version

In the modified version of the Chaocipher, proposed by Kruh and Deavours, the alphabet used for matching the plaintext character is selected based on a takeoff pattern which is part of the encryption key, rather than always selecting the right disk alphabet as with the classic version.

A major implication is that the ciphertext-only divide-and-conquer method described in the previous section does not work. With the modified version of Chaocipher, we need to reconstruct the correct left and right alphabets at the same time. For that purpose, we still use the Partial Index of Coincidence as the scoring method, and look only at the first $n = 8$ characters of each message, but we now perform transformations on both the left and right alphabets. We also need to repeat hill climbing for all $2^8 = 256$ possible takeoff patterns.

Due to its increased complexity and larger size of the search space, this modified algorithm requires many more in-depth messages to succeed. The success rate is only 22% with 400 messages, 73% with 500 messages, 93% with 600 messages, and 100% with at least 800 messages. Those high numbers of messages in-depth are highly unlikely from an operational perspective. The average work factor for a successful run with 500 messages is $500 \cdot 80M \approx 2^{35}$ decryptions, and $800 \cdot 11M \approx 2^{33}$ with 800 messages. By preventing the simple divide-and-conquer ciphertext-only attack described in the previous section, the modifications introduced by Kruh and Deavours provide for substantial additional cryptographic security.

8.4.4 Known-Plaintext Attack – Classic Chaocipher

In [129] a known-plaintext algorithm was proposed for a single message of at least 55-80 characters. However, this attack did not address the case of shorter messages in-depth.

We describe here a solution for in-depth messages, as short as 10 characters each, encrypted using the classic version of Chaocipher (takeoff pattern "RRRRRRRR.....".). Our proposed algorithm is based on hill climbing using the transformations described in Section 8.4.1 and uses the Plaintext-Ciphertext Weighted Match as the scoring method. The reason for using the Weighted Match method, and not the method of just simply counting matches, is the need to mitigate the propagation effect of early errors in encryption, by giving a higher weight to early

correct matches. We tested several weighting methods, and we obtained the best results with a simple $\frac{1000}{i}$ formula (i being the position in the ciphertext/plaintext), and looking at the first $n = 10$ characters of each message, with $i <= n$.

When looking at the first $n = 10$ characters of each message, this algorithm needs at least 10 messages for 100% probability of success. It also works with 20 messages and looking only at the first 5 characters. This algorithm is very fast, and when successful, the work factor ranges from hundreds of thousands of decryptions, to a few tens of millions at most, processed at a rate of 200 000 decryptions/second on an Intel Core i7 3.4Ghz PC.

8.4.5 Known-Plaintext Attack – Extended Chaocipher Version

The modified version of Chaocipher uses a takeoff pattern, which specifies, for each encryption/decryption step, which alphabet disk should be used to match plaintext or ciphertext characters, respectively.

We extend the known-plaintext attack described in Section 8.4.4, to the case of in-depth messages encrypted using the extended version of Chaocipher. The goal is not only to recover the left and right alphabets, but also the takeoff sequence. This extended attack consists of first finding the initial 5 elements of the takeoff pattern, as well as the left and right alphabets, and then iteratively finding the remaining elements of the takeoff pattern, one by one.

To find the first 5 elements of the takeoff pattern, as well as the left and right alphabets, we simply apply the algorithm described in Section 8.4.4, on the $2^5 = 32$ possible takeoff patterns of length 5, each time processing only the first $n = 5$ characters of each message. For such a hill-climbing run to succeed, at least 20 in-depth messages are needed. The initial pattern (the first 5 elements) which produces the highest score after hill climbing is the correct one, and hill climbing is also able to recover the correct left and right alphabets.

To recover the remaining elements of the takeoff pattern, we continue as follows:

- Putatively extend the takeoff pattern by adding “R” at its end (as its 6th element).
- Using this pattern and the left and right alphabets previously recovered, decrypt the first 6 characters of each message and compute the Plaintext-Ciphertext Weighted Match score.
- Repeat the process, this time using “L” as the 6th element of the takeoff pattern instead.
- Compare the two scores, and keep the takeoff pattern (now with 6 elements) that produced the highest score.
- Repeat the process for the 7th element of the takeoff pattern, then for the 8th, and so on, until a long enough sequence has been recovered, whose length is equal to the length of the longest message.

While this algorithm requires at least 20 in-depth messages with known-plaintext, it is also possible to start with a takeoff pattern of 10 characters and first running hill climbing $2^{10} = 1024$ times, if only 10 messages are available.

8.5 Solution of Exhibit 6

This exhibit, published in 2013 [123], consists of 50 messages, of about 25-30 characters each, encrypted using Kruh and Deavours's extended version of Chaocipher, with a takeoff pattern different from the simple case of "RRRRR....". The corresponding plaintexts were not provided.

The ciphertext-only method described in Section 8.4.2 obviously does not apply, as it applies only to the classic Chaocipher version. The ciphertext-only attack for the extended version of Chaocipher, described in Section 8.4.3, also does not apply as it requires several hundreds of messages, while only 50 messages are available.

To solve the exhibit, we had no choice but to find the original plaintexts. A few hints were of significant help:

1. Some of the messages started with the same letter, or with the same sequence of characters, in some cases up to 6 identical initial characters. With Chaocipher, two ciphertexts starting with the same initial ciphertext characters and encrypted using the same key settings, necessarily originate from plaintexts which start with an identical sequence of characters, of the same length.
2. The messages were all of relatively similar and uniform length, about 25 letters each, and this structure does not match a typical prose text. This was more indicative of the structure of a poem, or of another type of structured text, e.g. a table of contents, or a list of items. A poem made more sense given Byrne's biography and relationship with James Joyce.

We decided to follow this direction, and for that purpose used a program we had previously developed to parse all the books from the Gutenberg Project, after downloading all the files in zipped txt format. The parsing program was tuned to split the text into lines, and look for sequences of lines roughly matching the lengths of the ciphertext messages and their ordered sequence. Additional "points" were also given to sequences of lines showing the same initial sequences as displayed by the ciphertexts. After running for less than an hour, the program showed the poem "To His Coy Mistress" by Andrew Marvell as the top candidate, with the best match. While there was a perfect match for more than 34 messages and for most of the repeating initial characters, there were minor discrepancies with the length of 12 other messages, and the first 4 ciphertexts did not match anything. Still, the matches in the lengths, and in the repetitions of the initial letters, could not have resulted from random chance. There was no doubt that the plaintext was indeed generated – with some possible changes or errors – from that poem.

Next, we applied the known-plaintext algorithm described in Section 8.4.5, restricting it to process only the 34 plaintext verses and ciphertexts with perfectly matching lengths. We initially looked at the first 5 characters of each managed, and were able to reconstruct the left and right alphabets, and the first 5 elements of the takeoff pattern. This process took a very short while. We then iteratively recovered the remaining elements of the takeoff pattern. The complete solution is given here, including the explanations for the discrepancies. The plaintext was taken from "To His Coy Mistress" by Andrew Marvell, with some additions, adaptations, and errors. The encryption key settings are as follows:

```
Left alphabet:    RMDFUJZXOQIBAKTNVWSEGHCYLP
Right alphabet:   ISAVCQLPMZGODUTJKNBFRXEHYW
Takeoff pattern:  LRRLRRRLLRLRLLLRLRRLRLRLRRRLRLLRLL
```

The ciphertexts and plaintexts are shown below. The first 4 messages are introductory sentences, and not part of the original poem. In the poem verses, it seems that Kruh and Deavours inconsistently decided to replace punctuation signs in the original text, such as periods and columns, with the X character. This explains the discrepancies between the length of some of the verses, and the length of the corresponding ciphertext. Also, Kruh and Deavours's transcription is sometimes erroneous, such as "honour" shortened to "honor". In case of discrepancy in transcription, we also show the original verse (*) beneath Kruh and Deavours's transcription.

#	Ciphertext & Plaintext	#	Ciphertext & Plaintext
01	TIHULRZNXNSDGQLMYGNUQQAXFH TOHISCOYMISTRESSCOLONAPOEM	21	DVRHSLOCDSVJQXCFPAZTOFV ANAGEATLEASTTOEVERYPART
02	OYRBQNNZEGZECMZMOMKOAMEBLHB BYANDREWMARVELLCOMMASIXTEEN	22	DVINEVMEMJJMCFROJMQHHBVGVLCBGDLBD ANDTHELASTAGESHOULDSHOWYOURHEART**X** ANDTHELASTAGESHOULDSHOWYOURHEART
03	XANQDWXZSTSJMLRXNSLCYPDJDUO HUNDREDTWENTYONEDASHSIXTEEN	23	VISSLXWQIEDRIPNMZONCOWNMTY FORLADYYOUDESERVETHISSTATE
04	XANQDWXOAOJZSYMOXEQELAC HUNDREDSEVENTYEIGHTSTOP	24	JISRXQMPQIQGNSJGDZFQMEDKC NORWOULDILOVEATLOWERRATE**X** NORWOULDILOVEATLOWERRATE
05	XMIREDOHSJHHSNQQFHZLLHZMHG HADWEBUTWORLDENOUGHANDTIME	25	OKLNCFSRRCZRIRCBWXBWCO BUTATMYBACKIALWAYSHEAR
06	TCYXMXFROWACWTYQEVMFITXTOA THISCOYNESSLADYWERENOCRIME	26	TPZEYGGMVAXZKSQLYBEBTXTRMYEDHK TIMESWINGEDCHARIOTHURRYINGNEAR
07	NHCDQDEGRGOMQBDRKBJXHRKQNLPOOD WEWOULDSITDOWNANDTHINKWHICHWAY	27	DVIIXIXIPHBPZXQXTMBGNEQ ANDYONDERALLBEFOREUSLIE
08	TIYJDICMSRPTVHBEUSQDKVYITRWDL TOWALKANDPASSOURLONGLOVESDAY**X** TOWALKANDPASSOURLONGLOVESDAY	28	AHPEVCEUYHRTANPGKZIRTS DESERTSOFVASTETERNITY**X** DESERTSOFVASTETERNITY
09	TCQPKEGAUNTKMWKXNBNOGDUTR THOUBYTHEINDIANGANGESSIDE	29	TCCTSOLEBNBFHIDBURNIIVPPZUC THYBEAUTYSHALLNOMOREBEFOUND
10	BCQXDOGMPEMKBLTYUAUQDPTJRJZH SHOULDSTRUBIESFINDIBYTHETIDE	30	JISUACDUPGEZPKSVXUOEVWBBSBVSA NORINTHYMARBLEVAULTSHALLSOUND
11	MSQEJKMEYKJQSXGQJDHBYYEKQNHX OFHUMBERWOULDCOMPLAIN**X**IWOULD OFHUMBERWOULDCOMPLAINIWOULD	31	SYGYEZEEJJVGBYFBJGNCBMNRUXQPUN MYECHOINGSONG**THEM**WORMSSHALLTRY MYECHOINGSONG**THEN**WORMSSHALLTRY *
12	YIMEHZZRVCZAHYZVYFZPJXAFSGQFE LOVEYOUTENYEARSBEFORETHEFLOOD	32	TCRBDXCAPMNTPHKHKNFZJNHIKL THATLONGPRESERVEDVIRGINITY
13	DVIIXLLUBELWWQYLMFODQQCSKKHMQ ANDYOUSHOULDIFYOUPLEASEREFUSE	33	DVIIXLKBKHLUJLYNIVOYAGULAZZY ANDYOURQUAINT**HONOR**TURNTODUST ANDYOURQUAINT**HONOUR**TURNTODUST *
14	TPUSBSWAJZSPMBHHTSFORXYEDDA TILLTHECONVERSIONOFTHEJEWS**X** TILLTHECONVERSIONOFTHEJEWS	34	DVIUTQJXIOZVMRWBLCYHQC ANDINTOASHESALLMYLUSTX ANDINTOASHESALLMYLUST
15	SYMEZWGNPHQEPLZLSZITDPYIT MYVEGETABLELOVESHOULDGROW	35	TCGHVHLNNJIDIKUFNUNUIHXNRAPTC THEGRAVESAFINEANDPRIVATE**PLANE** THEGRAVESAFINEANDPRIVATE**PLACE** *
16	KMPNSTOABITQGSUJMXRZKKNJNUXV VASTERTHANEMPIRESANDMORESLOW	36	OKLBXQNFLPYDIKLOHMTHPWUWMSH BUTNONEITHINKDOTHEREEMBRACE
17	DVHPAXKJZNVSDCQCJLEGAWONPLNIXG ANHUNDREDYEARSSHOULDGOTOPRAISE	37	JIYNEWKTISMLDKVSFZGHVZRHVDBBXJY NOWTHEREFOREWHILETHEYOUTHFULHUE
18	TCYCSWGOOHUJRDBEGFXPTTRHUCJLY THINEEYESANDONTHYFOREHEADGAZE	38	BPAEXPQVQVWAVDGEBESZXJHAKFZ SITSONTHYSKINLIKEMORNINGDEW
19	TLQYILXKSUWMWAEJUZHIWPQNTTI TWOHUNDREDTOADOREEACHBREAST	39	DVIQEGMDMTRTIEMBACBTWQYUUGTXXNYX ANDWHILETHYWILLINGSOULTRANSPIRES
20	OKLAECKPKNLTSGYAISULNIFFRY BUTTHIRTYTHOUSANDTOTHEREST	40	DTGFWTSCNIDOAVRLCPAEAIXRQYM ATEVERYPOREWITHINSTANTFIRES

#	Ciphertext & Plaintext	#	Ciphertext & Plaintext
41	JIYSSCOITPXYJNPQZZOZMIZOM NOWLETUSSPORTUSWHILEWEMAY	46	MASXQWVXWAUAOIVBMTIXYWONN OURSWEETNESSUPINTOONEBALL
42	DVIBXGVFQPQIQXTOMDARMPBYGKXTW ANDNOWLIKE**AMOUROUS**BIRDSOFPREY ANDNOWLIKE**AMOROUS**BIRDSOFPREY *	47	DVINSPRNDLLBUKCRFEFSMETGMJNLNTXHGZ ANDTEAROURPLEASURESWITHROUGHSTRIFE
43	IDAYSNYRVCBBBKUUUDGZWABAA RATHERATONCEOURTIMEDEVOUR	48	TCQNOQUTHKTVGHALVAJSCIUSQMQ THOROUGHTHEIRONGATESOFLIFE**X** THOROUGHTHEIRONGATESOFLIFE
44	TCRVFCCAERNDIXROTHJXXZTGGQDSTBFENB THANLANGUISHINHISSLOW**CHAPPED**POWER**X** THANLANGUISHINHISSLOW**CHAPT**POWER *	49	TCBXISZAOKOYQNTPKQODJKUKPZFK THUSTHOUGHWECANNOTMAKEOURSUN
45	YHAPCNUQLSMHRBYMVDKRNDNMCPDQN LETUSROLLALLOURSTRENGTHANDALL	50	BKRBOCOBBLKYOQPIINKFEUYENJRGMD STANDSTILLYETWEWILLMAKEHIMRUN**X** STANDSTILLYETWEWILLMAKEHIMRUN

8.6 Security of Chaocipher

In Silent Years [113], Byrne makes several categorical assertions regarding the security of his Chaocipher. Those refer to the classic version, as Kruh and Deavours's extensions were introduced 37 years after the publication of Silent Years. In his book, Byrne claims that *"...[his] method for achieving the complete annihilation of order and design in written language is more noteworthy than [Ernest Rutherford's] method for the disruption of the atom..."*. He also states that Chaocipher is *"... a cipher which would be materially and mathematically indecipherable... by anyone except the persons for whom the message is intended..."*. Furthermore, he writes that *"... [the] possession of [his] device together with knowledge of the general principle involved, would not enable any person to decipher any message whatever written by anyone else and not intended for him."*, and he also claims that *"... the publication of the plain text of a trillion documents enciphered by my cipher system would not be of the least use or assistance to anyone attempting to cryptanalyze the cipher product of my system"*. Although the computing industry was in its infancy when he published Silent Years, he added: *"And finally, I issue to the believers in the wonderful capabilities of electronic calculating machines, a warm invitation to take up my challenge."*. Interestingly enough, Kruh and Deavours also believed in the security of Chaocipher. In 1990, after the inner workings of Chaocipher were disclosed to them, they wrote that *"Attempts to cryptanalyze the cipher appear to substantiate [Byrne's] claims"* and that *"... even the original system would not be easy to solve today using a computer."* [116].

To date, and this includes this current work, no method has been proposed for the ciphertext-only cryptanalysis of a single message, when no other messages are available which were encrypted in depth with the same key. On the other hand, prior work [118, 129] had already established that the classic version of Chaocipher is susceptible to a known-plaintext attack on a single message with at least 55 to 80 characters. Furthermore, in this case study, we show that for the more likely case of messages in-depth, including very short messages, an efficient known-plaintext attack may be applied, requiring as few as 10 messages in-depth. In addition, an efficient ciphertext-only attack may be applied which requires 60 to 80 messages in-depth. This attack on Byrne's classic Chaocipher version is based on a major flaw in the cipher system. This flaw enables an effective divide-and-conquer attack, based on the use of the Index of Coincidence. We, therefore, speculate that William F. Friedman, the inventor of the Index of Coincidence, may

have been aware of this flaw, and this could have been one of the reasons for rejecting Byrne's repeated proposals. At any rate, our latest findings, as well as prior work, seem to undermine the validity of most of Byrne's assertions.

Kruh and Deavours's extended version of Chaocipher is more resilient to ciphertext-only attacks, as such an attack requires many more messages. It is still, however, highly susceptible to a simple known-plaintext attack, as demonstrated by our solution of Exhibit 6.

While still an ingenious cipher for the time is was conceived, Chaocipher had additional weaknesses, when used in operational settings. When applied manually, encryption and decryption are tedious and error-prone. While the autokey nature of the cipher has the advantage of generating a seemingly chaotic stream of ciphertext characters, it also means that a single error in the transmission, reception, encryption or decryption of a message, could result in the complete inability for the receiving side to decrypt the message. And as demonstrated by the divide-and-conquer attack presented in Section 8.4.2, the autokey feature of Chaocipher may have been no more than a "complication illusoire".

It may, therefore, be argued that the fact the Chaocipher had not be "solved" for many years after it was first exposed in 1953 was primarily due to the secrecy around its inner workings, which ended in 2010, rather than its inherent cryptographic security. In that sense, Chaocipher probably failed to meet the principles formulated by Auguste Kerckhoffs in 1883 [12]:

1. *"Le système doit être matériellement, sinon mathématiquement, indéchiffrable"* (The system must be practically, if not mathematically, indecipherable).
2. *"Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi"* (It must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience).

A similar requirement was formulated by Claude Shannon in 1949 [13]: *"The enemy knows the system being used"*. We conclude with Charles Babbage's words:

One of the most singular characteristics of the art of deciphering is the strong conviction possessed by every person, even moderately acquainted with it, that he is able to construct a cipher which nobody else can decipher. I have also observed that the cleverer the person, the more intimate is his conviction [130].

8.7 Summary

In this chapter, we presented new effective attacks on Chaocipher, designed in accordance with our new methodology, as summarized in Table 8.2 and in Table 8.3.

One of those attacks was employed for the solution of the last remaining Chaocipher challenge, Exhibit 6. A ciphertext-only attack has been demonstrated for the first time, based on a powerful divide-and-conquer approach, made possible by the use of a highly specialized scoring function. Finally, those new attacks enable us to more accurately assess the cryptographic security of Chaocipher.

Principle	Application of the methodology principle
GP1	Hill climbing, parallel search for left and right alphabets
GP2	
GP3	Specialized weighted match score, prioritizing the initial letters of the messages, achieving high resilience to errors
GP4	Simple non-disruptive swap transformations
GP5	Multiple restarts

TABLE 8.2: Chaocipher – applying the methodology – known-plaintext attack

Principle	Application of the methodology principle
GP1	Hill climbing, sequential search, first HC to recover the left alphabet key, second HC to recover the right alphabet
GP2	Divide-and-conquer – left alphabet, then right alphabet
GP3	Specialized IC-based score, applied to the beginning of the messages in depth, with high resilience to errors
GP4	Simple non-disruptive swap transformations
GP5	Multiple restarts

TABLE 8.3: Chaocipher – applying the methodology – ciphertext-only attack

9

Case Study – Solving The Double Transposition Cipher Challenge

The structure of this case study is different from the structure of the other cases studies, where we described how we applied our new methodology for the cryptanalysis of various ciphers. Rather, we describe here how we solved the Double Transposition Challenge in 2013 ([38]). This work had a major contribution to the development and the formulation of the concepts and principles behind the methodology. At the end of the chapter (Section 9.6), we describe the contributions this project had on the development of our new methodology. This case study also illustrates the iterative thought process and the experimental steps applied while developing new methods for a hard cryptanalytic problem, considered to be unsolvable by the former head of the German "Zentralstelle für das Chiffrierwesen" [38].

The double transposition cipher was considered to be one of the most secure types of manual ciphers. It was extensively used in both World Wars and during the Cold War. In 1999, Otto Leiberich, the former head of the German federal office for information security, suggested that a double transposition challenge be published with specific parameters designed to ensure its security. Such a challenge was published by Klaus Schmeh in 2007 (see Section 9.3). In November 2013 we solved the challenge using a ciphertext-only hill-climbing attack. We also solved the challenge using a dictionary attack. We describe both methods, which are based on a divide-and-conquer approach. We additionally discuss the impact of our solutions with respect to the general security of the double transposition cipher.

The results presented in this chapter have also been published in Cryptologia [32].

9.1 The Double Transposition Cipher

The double transposition cipher, also known as the double columnar transposition, has been one of the most popular manual ciphers. It did not require the use of a device for encryption and decryption. Because of its simplicity and its high level of security, it was often the cipher of choice for intelligence and secret operations organizations [38].

The process of encryption and decryption is relatively simple. First, two transposition keys, K_1 and K_2, must be chosen and agreed in advance. Keys are usually derived from keywords or key

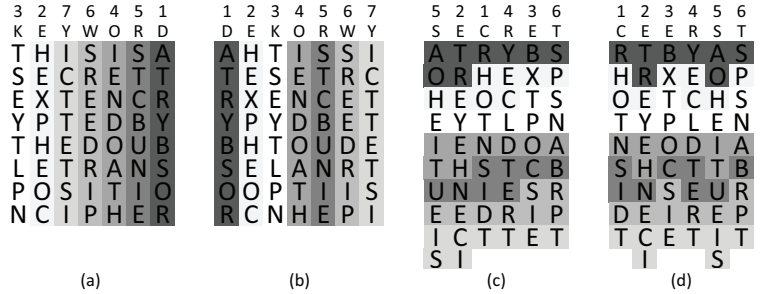


FIGURE 9.1: Double transposition – example

phrases which in turn are converted to numerical keys. Such keywords or key phrases are often taken from books or newspapers. An example of encryption using the double transposition cipher is presented in Figure 9.1. Using the keys $K_1 =$ "KEYWORD" and $K_2 =$ "SECRET", we encrypt the plaintext $P =$ "THISISASECRETTEXTENCRYPTEDBYTHEDOUBLETRANSPOSITIONCIPHER".

First, we prepare the first transposition rectangle by writing down the first key K_1 and its numerical equivalent (Figure 9.1(a)). Underneath the key we fill the rectangle by writing the plaintext row-by-row. We then apply the first columnar transposition by changing the order of the columns according to the numerical equivalent of key K_1. This results in an intermediate rectangle (Figure 9.1(b)). After that, we prepare the second transposition rectangle by writing down the second key K_2 and its numerical equivalent (Figure 9.1(c)). Then we extract column-by-column the text from the intermediate rectangle in Figure 9.1(b) and write it row-by-row into the rectangle underneath the key K_2. We then apply the second columnar transposition by changing the order of the columns according to the numerical equivalent of key K_2. This results in the final rectangle (Figure 9.1(d)). The final ciphertext is extracted column-by-column from this rectangle yielding the ciphertext
$C =$ "RHOTNSIDTTREYEHNECIBXTPOCSIEYECLDTERTAOHEITUEISSPSNABRPT". To decrypt the ciphertext we apply the inverse steps. We first need to undo the second columnar transposition with K_2. After that, we undo the first transposition with K_1.

The German Army used the double transposition cipher (in German: "Doppelwürfel"[1]) in WWI in a less secure form by using the same key for K_1 and K_2. The French "Bureau de Chiffre", who called this cipher Übchi, regularly solved the cipher until the German Army replaced it with another cipher, the ABC cipher, following leaks in the French press [38]. During WWII it was extensively used by different countries. In the US it was used by the Army either with the same or with different K_1 and K_2 keys and by the Office of Strategic Services (OSS) as an emergency cipher. In Great Britain it was used by the British Special Operations Executive (SOE) to communicate with its agents in continental Europe. The Czechoslovakian government in exile in London used it as well as the French Resistance and the German Abwehr operatives in Latin America [2, 38]. During the Cold War, the East Germany's Stasi used double transposition ciphers to communicate with agents in West Germany, including the Granit cipher which added a monoalphabetic substitution before the double transposition. West Germany's cryptographic agency, the "Zentralstelle für das Chiffrierwesen" (in English: center for ciphers), was able to find solutions using a computerized keyword dictionary attack [38]. In his 2012 book about

[1] The term "Doppelwürfel" literally translates to "double cube" which refers to the two transposition rectangles.

unsolved ciphers, Klaus Schmeh estimated that the double transposition cipher might still be in use [38].

9.2 Related Work – Prior Cryptanalysis

Several NSA declassified publications present the classical manual methods for the cryptanalysis of the double transposition cipher. In [5], Friedman presents a set of special cases of double transposition messages which can be solved relatively easily. Such as a perfect rectangle where the length of the ciphertext is equal to the product of the lengths of the two keys or when the operator forgets to apply the second transposition. In [131], Kullback presents the classical method of multiple anagramming which requires several messages of the same length and encrypted with the same key. He also presents a method to recover the key once the multiple anagram sequences have been identified. Furthermore, his book includes a dictionary attack, where keys are derived from words. However, this attack cannot be used for the challenge at hand as its keys were derived from longer key phrases.

In the most comprehensive document about classical cryptanalysis of the double transposition cipher, Barker [31] presents several manual methods including solutions for special cases, multiple anagramming, and a known-plaintext attack. The most generic method is the rotating matrix. Barker establishes an equivalence for a given ciphertext length between a double transposition cipher with keys of lengths $|K_1|$ and $|K_2|$ and a single transposition cipher with a key K of length $|K| = |K_1| \cdot |K_2|$. The rotating matrix method requires a ciphertext of at least $2 \cdot |K|$ letters making this method not applicable to the challenge at hand with only 599 letters, and given that K_1 and K_2 have each at least 20 elements, $2 \cdot |K| \geq 800$.

Apart from those declassified publications, very few publications are available on this subject and even fewer presenting modern methods for the cryptanalysis of the cipher. In [132], Tim Wambach presents a known-plaintext attack, which requires the knowledge of the full plaintext of the encrypted message. In the public Google Group sci-crypt and in a private mail Jim Gillogly briefly describes a hill-climbing approach, which can achieve solutions for ciphers encrypted with short keys of up to 12 elements.

9.3 The Challenge

Since the 1950s, Otto Leiberich worked for Germany's main cryptographic agency, the "Zentralstelle für das Chiffrierwesen" in Bonn. He eventually became its director in 1972. During the Cold War, he and his team worked intensively on the cryptanalysis of double transposition ciphers. One of their results led in 1974 to the discovery of the spying activities of Günter Guillaume who was Willy Brandt's senior aide. Later, Otto Leiberich became the head of the "Bundesamt für Sicherheit in der Informationstechnik (BSI)" (in English: federal office for information security) in Germany. [38]

In order to encourage research on the double transposition cipher he suggested in 1999 that a double transposition challenge be published [133]. Leiberich's recommendations for the challenge included:

- Both transposition keys should be long enough: 20 to 25 elements.

- The lengths of the two keys should be co-primes (no common divisor except 1).
- The length of the ciphertext should not be a multiple of the length of either key.
- A ciphertext of approximately 500 characters (which is also approximately the product of the lengths of the two keys) should be used.

Those requirements were intended to avoid vulnerabilities, known to exist in special cases [5, 31, 131]. They were also based on Otto Leiberich's own experience with cryptanalysis of double transposition ciphers. In 2007 Klaus Schmeh published the double transposition challenge based on those guidelines [134]. He chose an English plaintext and encrypted it using two transposition keys, both derived from English key phrases longer than 20. The length of the plaintext was 599. The challenge was also published in the cryptographic challenges site "MisteryTwister C3" [135], in another book by Klaus Schmeh [38], and in several websites and blogs [135–137]. The double transposition challenge was ranked as #5 of the top 25 unsolved ciphers [137], and was included in a list of the "World's Greatest Unsolved Ciphers" [138]. Otto Leiberich considered the cipher to be unbreakable. He literally stated, "Für mich wäre es eine Sensation, wenn jemand dieses Rätsel lösen könnte." with respect to Klaus Schmeh's challenge [38]. This sentence loosely translates to, "it would be a real sensation to me if someone could solve this riddle". Below we provide the ciphertext:

```
VESINTNVONMWSFEWNOEALWRNRNCFITEEICRHCODEEAHEACAEOHMYTONTDFIFMDANGTDR
VAONRRTORMTDHEOUALTHNFHHWHLESLIIAOETOUTOSCDNRITYEELSOANGPVSHLRMUGTNU
ITASETNENASNNANRTTRHGUODAAARAOEGHEESAODWIDEHUNNTFMUSISCDLEDTRNARTMOO
IREEYEIMINFELORWETDANEUTHEEEENENTHEOOEAUEAEAHUHICNCGDTUROUTNAEYLOEIN
RDHEENMEIAHREEDOLNNIRARPNVEAHEOAATGEFITWMYSOTHTHAANIUPTADLRSRSDNOTGE
OSRLAAAURPEETARMFEHIREAQEEOILSEHERAHAOTNTRDEDRSDOOEGAEFPUOBENADRNLEI
AFRHSASHSNAMRLTUNNTPHIOERNESRHAMHIGTAETOHSENGFTRUANIPARTAORSIHOOAEUT
RMERETIDALSDIRUAIEFHRHADRESEDNDOIONITDRSTIEIRHARARRSETOIHOKETHRSRUAO
DTSCTTAFSTHCAHTSYAOLONDNDWORIWHLENTHHMHTLCVROSTXVDRESDR
```

9.4 Solving the Challenge

The purpose of the work presented here was to develop new cryptanalytic methods with the immediate goal of solving the challenge published by Klaus Schmeh in 2007. Nevertheless, we were able to identify several general vulnerabilities, which we present in the last section (see Section 9.6).

The remainder of this section is organized as follows: We first give a brief walk-through of the entire process and an overview of the methods. After that, we present in Section 9.4.2 our own prior work on the single columnar transposition and in Sections 9.4.3 to 9.4.8 we discuss each step and its consequences in detail.

9.4.1 Overview

Some of the building blocks and insights for solving the double transposition challenge are a direct result of our prior research on solving difficult cases of single columnar transposition ciphers (see Chapter 5). We present the relevant facts from this research in Section 9.4.2.

We discuss our first attempt to solve the double transposition cipher challenge in Section 9.4.3 and refer to it as *Step 1*. In this step we explored a hill-climbing algorithm, which searches in parallel over both the K_1 and K_2 keyspaces. However, we found that this approach was only successful for lengths up to 15, which is less than required for the challenge at hand.

Our second attempt (*Step 2*) was the application of a known-plaintext attack as described in Section 9.4.4. In order to apply this attack, we tried to guess parts of the original plaintext. However, this was also a dead end for the challenge, since the attack itself required a relatively long known-plaintext sequence, which we did not have. Additionally, we were not able to successfully guess any parts of the plaintext.

It became clear, that we needed to reduce the huge search space spanned by K_1 and K_2 together. Hence, in *Step 3* we evaluated several alternatives for an effective divide-and-conquer approach. The general idea is, that if we can first find the second transposition key K_2 then finding K_1 is just a matter of solving a single transposition cipher. For that purpose, we developed the Index of Digraphic Potential (IDP) as an effective and cost-efficient scoring function for K_2 as discussed in Section 9.4.5.

Armed with the knowledge from Step 1 and Step 3 we modified the hill-climbing algorithm to incorporate the IDP and to search over the K_2 keyspace while ignoring K_1. On November 25th, 2013, we achieved our first breakthrough and solved the challenge based on a partial solution from this hill-climbing method. We obtained the numerical keys for both K_1 and K_2 as well as the original plaintext. We sent the solution to Klaus Schmeh, who acknowledged its correctness. We refer to this first breakthrough as *Step 4* and describe its details in Section 9.4.6.

Even though the challenge was solved, we did not stop there. We also implemented an efficient K_2 dictionary attack based on the IDP and using a database of known expressions as described in *Step 5*. With this dictionary attack and also using the IDP we achieved our second breakthrough: It also produced – independently from our first solution – the correct solution for the challenge and additionally the original English key phrase used to create the K_2 transposition key. We discuss the details of this step in Section 9.4.7.

The only part missing at this point was the original English key phrase for K_1 equivalent to the already known numerical value of K_1. In our final step we, therefore, relied on earlier work by Jim Gillogly to deduce the English key phrase for K_1. Hence, in *Step 6*, as described in Section 9.4.8, we finally had all the elements of the solution, i.e. the plaintext, the transposition keys, and the key phrases.

9.4.2 Own Preliminary Work

Preceding our work on the double columnar transposition cipher and the challenge at hand, we investigated and evaluated new general solutions for the single columnar transposition cipher (see Chapter 5). The most difficult cases are with very long key lengths and incomplete transposition rectangles. Furthermore, for incomplete transposition rectangles the worst case is when about half of the columns are longer than the other columns. We found that our algorithm (described in Chapter 5) is able to successfully decrypt cipher messages in the worst case with key lengths up to 120. In the best case, our algorithm can even cope with key lengths up to 1 000.

Our algorithm for the single columnar cipher is based on hill-climbing with two phases. For the first phase we developed two new scoring functions, i.e. the *adjacency score* and the *alignment score*. The adjacency score relates to the likelihood of any column j being right next to another

column i. The alignment score reflects the precise starting position and alignment of the text of each column. For the second phase of hill climbing, we use plaintext quadgram log-frequencies for scoring. Additionally, we show that segment-wise transformations, e.g. swapping n consecutive elements of the key with another segment of n consecutive elements, are more effective than transformations on singular key elements, such as swapping key element i with key element j.

With our research on the single columnar transposition we gained important insights for the work on the double transposition, and in particular for the development of the Index of Digraphic Potential. We also reused some of the building blocks such as segment-wise transformations.

9.4.3 Step 1: Hill Climbing over K_1 and K_2 in Parallel

We first implemented an algorithm for the double transposition with two HC processes, which search for K_1 and K_2, in parallel. In each iteration the algorithm looks for a change in either K_1 or K_2 which may improve the score of the resulting decrypted text. We used log-frequencies of plaintext trigrams. The transformations were segment slides (rotating or shifting a segment of the key), segment swaps as well as 3-party swap transformations.

This algorithm requires the knowledge of the correct key length. In case this is not known, hill climbing must be applied on the ciphertext for each possible combination of the key lengths. For the case of the challenge at hand the lengths could only be 21, 22, 23, 24, or 25 and co-primes. Hence, there are only 16 possible key length combinations. For the remainder of this case study and for each method which we present, we implicitly test all those combinations and can therefore assume to know the key lengths.

We tested our new algorithm on simulated texts and keys with particular attention on worst-case incomplete rectangles. As already mentioned before, such a worst case occurs, when the number of long columns almost equals the number of short columns. We tested worst cases for the K_1 transposition rectangle as well as for the K_2 transposition rectangle. In these cases we found this method to have about 90% probability of success for keys lengths up to 13, and about 50% for key lengths of 14-15. Additionally, we observed that for the following special cases our algorithm also succeeded with longer keys:

- The length of the ciphertext is exactly equal to the product of the lengths of the two keys, i.e. $|C| = |K_1| \cdot |K_2|$. In this case we have a *perfect transposition rectangle* i.e. a complete transposition rectangle for both the K_1 and K_2 transpositions.
- The length of the ciphertext $|C|$ is a multiple of only one of the two key lengths, i.e. $|K_1|$ or $|K_2|$. In this case one of the two columnar transpositions results in a complete transposition rectangle.
- The length of the ciphertext only slightly deviates from the product of the lengths of the two keys, i.e. $|C| = |K_1| \cdot |K_2| \pm 1$, resulting in an *almost perfect transposition rectangle*.

The first two cases are well known in the classical literature [5]. Solutions are easier to find if one or both of the transposition rectangles are complete, as there is less ambiguity in the position of the columns after transposition. The case of an almost perfect transposition rectangle was less known, and could be relevant for the challenge at hand if the used key lengths would have been 24 and 25, since $|C| = 599 = 24 \cdot 25 - 1$ or $|C| = 599 = 25 \cdot 24 - 1$. Therefore, we tested this assumption by running our algorithm with the challenge ciphertext with the key lengths

$|K_1| = 24$, $|K_2| = 25$ and $|K_1| = 25$, $|K_2| = 24$. Unfortunately, those tests failed to produce a solution, but they allowed us to rule out those two combinations of key lengths. The number of possible key lengths combinations was thus reduced from 16 to 14. Still, we needed another approach to solve the challenge.

9.4.4 Step 2: Known-Plaintext Attack

Next we tried to solve the cipher with a known-plaintext attack. No such text was available for the challenge, but we assumed that the plaintext might be related to Klaus Schmeh domains of expertise. Therefore, some words or expressions might be guessed. For that purpose, we created a database of expressions and sentences based on several cryptography and computer security books and articles. We implemented a known-plaintext attack by adapting the hill-climbing algorithm described in Step 1 and only changing the scoring function. The scoring function was simply the number of correctly recovered letters in the decrypted text compared to the expected plaintext.

We showed with simulations that obtaining a solution for keys longer than 20 would require hundreds of characters. Obviously, this is more than one can possibly guess. To improve the algorithm, we implemented a "hybrid" approach. We allocated 50% of the score to plaintext recovery, as described above, and the other 50% to a trigram score, as described in Step 1. With this improved hybrid method, we could solve simulated ciphers with parameters similar to those of the challenge with only 100 letters (out of 599) of known or guessed plaintext and in rare cases with only 60. Again, that was not enough to solve the challenge. While this improved algorithm was useful at a later stage (Step 4), we needed a new and more powerful approach.

9.4.5 Step 3: Reducing the Search Space

One of the main challenges in searching for the keys of a double transposition cipher is the size of the combined keyspace. Basically, the keyspace is $|K_1|! \cdot |K_2|!$. For keys with lengths up to 25, this is about 2^{162} or roughly equivalent to the keyspace of a 3DES encryption with three different keys. This creates significant challenges with our algorithm in Step 1, which searches K_1 and K_2 in parallel. The size of the keyspace creates similar challenges for dictionary attacks.

The complexity of a dictionary attack, when searching for both K_1 and K_2 in the dictionary is $O(n^2)$ where n is the dictionary size. Short keys are usually derived from single keywords (e.g. "transposition") while longer keys are usually derived from full sentences or expressions such as "adifficultcipher" or "thiscipherisnoteasy". Thus, dictionary attacks for double transpositions are easier for shorter keys compared to longer keys, as there are fewer combinations to check. Word dictionaries are available with a number of entries n, varying from tens of thousands and containing the most common words, up to several hundreds of thousands with inflection, rare words, and names of places. Thus, there may be between $n^2 = (10000)^2 \approx 2^{27}$ to $n^2 = (400000)^2 \approx 2^{38}$ combinations to check. In practice, we don't need to check all those combinations since only words matching the required lengths need to be tested. We roughly estimated that testing about $100000000 \approx 2^{27}$ combinations of dictionary keywords would be a feasible attack for our available hardware. However, this attack would cover only all combinations of single words, which is not good enough for the challenge at hand. A key of length 20 to 25 may only be derived from an expression or a sentence, such as "doubletranspositioncipher" or "thisisaveryhardproblem". Databases of the most common expressions and sentences are also available, starting with $n = 1000000$ entries [139]. However, for good coverage, a larger

database may often be required. Such databases are available, with up to 98 billions of entries [140]. Assuming that 10% of the entries are of lengths relevant to the challenge, we would need to test about $(98 \cdot 10^9/10)^2 \approx 2^{67}$ cases. Clearly, this was not practical for this work.

9.4.5.1 A Divide and Conquer Approach

To achieve any kind of breakthrough, we needed a method to reduce the keyspace. Specifically, we needed an approach to find one key independently of the other. Furthermore, if we could find a method for *scoring* one key without knowing the other, this would open the door for effective divide-and-conquer hill-climbing or dictionary attacks. We focused on the case of first finding or scoring the second transposition key K_2, independently of K_1. Once the second transposition is undone using a known or candidate K_2 key, we are just left with a single transposition, i.e. the first transposition with the K_1 key. We considered a first, albeit naive, method for scoring a given candidate K_2 without knowing K_1 a priori as follows:

1. First undo the second transposition with the assumed K_2. This candidate K_2 could be the result either from the current iteration of hill climbing or from a dictionary search.
2. Find that K_1 which produces the plaintext with the highest bigram score after decryption. This can be done by solving the K_1 transposition, which can now be treated as a single transposition. For doing so we could rely on our own prior work for solving single columnar transpositions with hill climbing as described in Section 9.4.2. As the scoring function we use bigram log-frequencies.
3. Use the best bigram score found in (2) as the score for the assumed K_2.

For the challenge at hand the transposition rectangle is incomplete for all possible K_1 key lengths. Based on our prior findings, our algorithm for solving the incomplete single columnar transposition on K_1 would take around one minute to complete on a standard home PC. This time-consuming process must be repeated for each candidate K_2 key. Clearly, this approach is not practical for key phrase dictionary attacks on K_2 with approximately $n = 98 \cdot 10^9/10 \approx 2^{33}$ sentences and expressions to check. It is neither practical for hill climbing over K_2. For a key of $|K_2| = 25$ elements there are about $3 \cdot 25^3 = 46\,875 \approx 2^{16}$ possible transformations and resulting candidate K_2 keys to be checked for each iteration of hill climbing. This translates into hundreds of hours per iteration, while tens of such iterations may be required.

9.4.5.2 The Index of Digraphic Potential

Even though the naive approach from the last section is not practical, it provides some valuable insight paving the way for our new scoring function as described in this section. The naive method – though highly inefficient – provides a value, which reflects the best possible plaintext bigram (or digraph) score achievable with a candidate K_2 key. This value may also be viewed as a quantitative index, reflecting the "digraphic potential" of a given K_2. Therefore, we named this score the *Index of Digraphic Potential* (IDP).[2]

We hypothesized that the closer a candidate K_2 key is to the correct K_2 key, the higher the potential to reconstruct the original plaintext and its bigrams. From this we deduced that there

[2] An alternative name for this score could have been the *Index of Bigraphic Potential (IBP).*

should be a negative correlation between the number of errors in K_2 and its IDP. Hence, we looked for an efficient method to compute this IDP for a candidate K_2 key without having to fully solve the K_1 transposition as described in (2) of the naive approach.

The following insight came from our prior research and solutions for single transpositions with incomplete rectangles. For simplicity, we only consider here a single transposition with an incomplete rectangle and a K_1 key of length $|K_1|$. In the encryption process the plaintext is written in rows, where the length of each row is $|K_1|$. In an incomplete rectangle and before transposition, the leftmost columns are longer than the rightmost columns by one row. Next in the encryption process, the order of columns is reshuffled using the transposition key K_1 and the resulting text is extracted column by column to form the ciphertext. A cryptanalyst does not know where the text of each original plaintext column appears in the ciphertext since he does not know K_1. Nevertheless, he does know that each one of the transposed columns appears in the ciphertext as a continuous segment. Furthermore, for a key K_1 of length $|K_1|$ and a ciphertext C of length $|C|$ the number of *full* rows in the transposition rectangle equals to $r = \lfloor \frac{|C|}{|K_1|} \rfloor$. Therefore, he may also assume that the length of each column may either be r or $r+1$. As a result, there can be several possible starting and ending positions for each column since preceding columns might each be either a long or a short column. It is clear that the first column in the ciphertext always starts at position 1 and that the last column always ends at position $|C|$. If the first column is short then the second column will start at position $r+1$. If instead the first column is long, then the second column starts at $r+2$. The same reasoning can be further applied to the next columns and also backward from the end of the ciphertext to determine the range of possible starting positions for each column. If the number of long columns is approximately equal to the number of short columns, then columns in the "middle" of the ciphertext have approximately up to $\frac{|K_1|}{2}$ possible starting positions. In order to solve such a single transposition and to find the $|K_1|$, we need to determine the original order of the columns. For that purpose, we want to evaluate the likelihood that a given column j in the ciphertext was the right neighbor, in the original plaintext, of another ciphertext column i. One way to evaluate this is to compute the sum of log frequencies of all the bigrams created by juxtaposing column j next to the right of i. As discussed above, there can be several possible starting positions for both columns. Therefore, this bigram score must be computed for all those possible starting positions of both columns i and j in the ciphertext. Such a process may not only indicate which column j is likely to be the neighbor next right to column i, but it may also help to determine the exact positions of those columns in the ciphertext and how they are aligned next to each other. This analysis and the techniques described above lie at the core of our prior solutions for solving complex single transpositions.

We applied the same analysis and a similar approach to develop an efficient algorithm for computing the IDP of candidate K_2 keys. This algorithm is deterministic and it computes the IDP of a candidate K_2 key without solving the K_1 columnar transposition:

1. First undo the second transposition with the candidate K_2 key.
2. Prepare an empty matrix $B[i, j]$ and perform for each possible combination of ciphertext columns i and j the following calculations:
   (a) Compute the sum of log frequencies of all the bigrams created by juxtaposing column j to the right side of column i. To normalize the result, divide this sum by the number of rows. Store this value in the matrix cell $B[i, j]$.
   (b) Repeat this calculation for all possible starting positions of both i and j continuously updating the matrix cell $B[i, j]$ with the best (highest) value found.

3. As a result of the last operation we have a matrix $B[i,j]$ with the best possible digraphic values for all pairs of columns i and j. However, in a real transposition, each column (or key element) can have only one right neighbor, as well as only one left neighbor. Therefore, in order to further reduce the matrix perform the following operations iteratively:
   (a) Select the column pair (i,j) from the matrix $B[i,j]$ with the highest value.
   (b) Mark j as the likely right neighbor of i, and i as the likely left neighbor of j. Furthermore, mark all $B[i,*]$ and $B[*,j]$ cells except $B[i,j]$ as invalid.
   (c) Repeat (a) and (b) with columns i which do not yet have a likely right neighbor and columns j, which do not have yet a likely left neighbor, until all columns have been assigned both a right and a left likely neighbor.
4. Sum all $B[i,j]$ values of all pairs, considered as likely neighbors. We define this sum as the value for $\mathrm{IDP}(K_2)$ for the given key K_2.

This implementation of the IDP can be performed on a 3.4 GHz Intel Core i7 PC with 8 cores and parallel processing at the rate of 20000 calculations per second. Compared to the approximate 60 seconds with the naive approach, this is an speed improvement by a factor of $1:(60\cdot 20000) = 1:1200000$. Additionally, with the naive approach, solving for K_1 may take more or fewer cycles for different keys K_2. In contrast, the above method for calculating the IDP is deterministic in time, i.e. it always takes the same time (cycles) for any K_2. This eases the implementation of parallelization.

9.4.5.3 Evaluation of the IDP

Before integrating the IDP into either hill climbing or a dictionary attack, we first needed to assess its suitability as a scoring function for K_2. We performed a fitness-distance analysis (see Section 2.3.10) to assess how the IDP value is affected by the number of erroneous elements in the key. To do so, we defined the degradation of the IDP, $D(x)$, for x perturbations in a key as the difference between the IDP of the correct key $K^{(0)}$, and the IDP of the same key but after x perturbations, which we denote as $K^{(x)}$. Hence, we formally define the degradation of the IDP as $D(x) := \mathrm{IDP}(K^{(0)}) - \mathrm{IDP}(K^{(x)})$.

We measured this value by starting with a correct key $K_2^{(0)} = K_2$ and inserted iteratively a number of artificial perturbations. To create the erroneous keys $K_2^{(x)}$, we swap in each iteration two single elements of the key $K_2^{(x)}$ making sure that a swap does not correct an error created by a previous swap. We simulated $D(x)$ for 100000 different K_2 keys of length 23 and applied to each key up to 23 perturbations, i.e. $x \in [0\ldots 23]$. We present the results from this simulation in Figure 9.2. The X-axis represents the number of perturbations and the Y-axis the degradation in the IDP, $D(x)$.

From this simulated data we were able to verify the two most important properties of the IDP. First, it is highly *selective*: We found only one single case with a K_2 slightly different from the correct one, which produced an IDP better than for the correct key. Therefore, the IDP can be highly effective for a K_2 only dictionary attack with $O(n)$ complexity. The K_2 key phrase, which produces the highest IDP, is almost certainly the correct key or at least an almost correct one. Our second important finding is, that the IDP is on average *monotonic* for a wide range of perturbations and that the changes are steeper as we get closer to the solution (fewer than 10 errors). This property is especially useful and important for hill climbing on K_2 without knowing K_1, as we show in the next section.

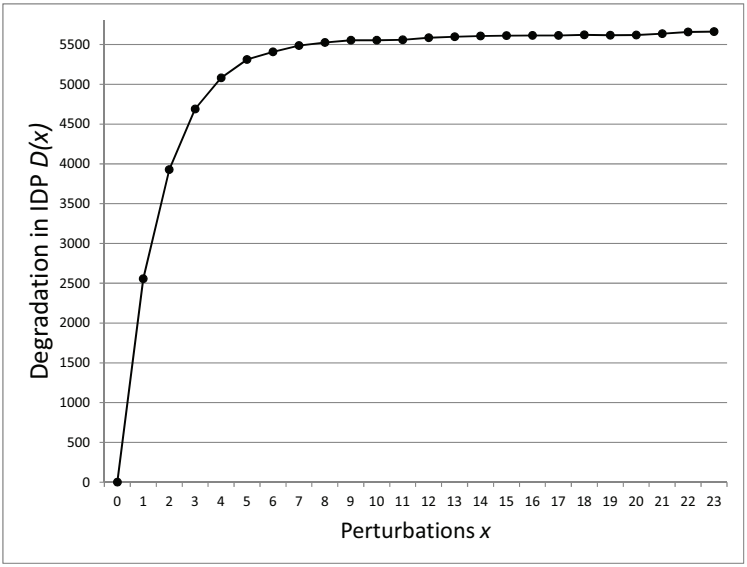


FIGURE 9.2: Double transposition – fitness-distance analysis of the IDP

9.4.6 Step 4: Improved Hill Climbing

From the evaluation of the IDP in the last section, we knew that this scoring function is highly selective, and also monotonic for partially correct K_2 keys. These properties make it a highly suitable choice for a K_2 hill-climbing algorithm to solve double columnar transposition ciphers. Therefore, we developed an improved "divide-and-conquer" hill-climbing algorithm based on the IDP with the main search being over the K_2 space while first ignoring K_1:

1. Generate a number of random K_2 keys and for each one compute the IDP. Keep a subset of the keys with the highest IDPs.
2. For each key in the subset obtained by (1), apply a simple "left-to-right" improvement heuristic and keep only the subset with the best IDPs after this improvement. The main purpose of this improvement is to provide the hill-climbing part in the next phase (3) with such initial K_2 keys which are more likely to be closer to the solution. Perform the following operations to achieve the "left-to-right" improvement heuristic:
   (a) For each key element i starting with $i = 1$, check whether swapping the key element i with another key element j on its right ($j > i$) results in a key with a higher IDP score. If there is an improvement, perform the swap and keep the resulting key.
   (b) Repeat (a) for the next i until the penultimate key element $i = |K_2| - 1$ is reached.
3. For each key in the subset obtained by (2), run a K_2 hill-climbing algorithm as follows:
   (a) Perform all possible segment slide, segment swap, and 3-partite swap transformations on the current K_2. Assess the scoring for each transformed K_2 key by calculating the IDP.

(b) If IDP has improved, keep the new key as the current best key and repeat (a). When reaching a maximum and no more improvements are possible, compare this best key IDP to the overall best IDP, achieved by hill climbing, with any of the keys from the original (2) subset. If higher, keep that key as the best overall K_2 key.

4. Using the best overall K_2 key from (3), perform the following:

   (a) Undo the second transposition using that best overall K_2. Solve the remaining K_1 single columnar transposition using hill climbing and trigrams for scoring. This yields a key K_1 which in turn results in a plaintext with the highest trigram score. This may already be the correct K_1.

   (b) In order to improve and finalize the K_2 key, use the best K_1 key from (a) to perform the following hill-climbing process:

      i. Check all possible segment slides, segment swaps, and 3-partite swaps transformations for the current K_2. However, instead of computing the IDP for each transformation, do the following:

         A. Fully decrypt the ciphertext using the current best K_2 and the best K_1 from (a) and compute the decrypted plaintext trigram score.

         B. If a trigram score higher than the previous best was found, keep this key as the new best K_2.

      ii. Repeat (i) until the trigram score cannot be further improved, i.e. no better K_2 has been found.

5. Repeat (1) to (4) until a score threshold is reached. In that case the keys have probably been found. The algorithm probably failed if a maximum number of iterations have been done without reaching this threshold.

We tested this new algorithm, and it provided an immediate and significant improvement. The new algorithm has a success probability of over 60% for finding keys of length 19 and 20 for K_1 and K_2, respectively. For key lengths of 18 to 19 we even have a success probability of 90%. This is a significant improvement in comparison with the previous hill climbing of Step 1, which could only solve keys with lengths up to 13-15. Unfortunately, this was still not enough to solve the challenge at hand. For now, we turn our attention to further optimizations we introduced to the algorithm, specifically tailored to the challenge at hand.

9.4.6.1 Optimizations

It is publicly known that the keys for the challenge are derived from English key phrases. We hypothesized that based on this knowledge the keyspace for hill climbing could be reduced – or at least the search could be optimized. In any language some letters occur more often than others. Hence, in an English sentence or expression some low-frequency letters, such as Z or J, are likely to be absent, while high-frequency letters such as E and T are likely to appear several times. Additionally, when a letter appears more than once in a key phrase, the various occurrences of that repeated letter are represented in the equivalent numerical key by successive numbers. As an example consider the keyword "`REFERENCE`", with the letter E occurring four times. The numerical key for this word is "8,**2**,6,**3**,9,**4**,7,1,**5**", i.e. the occurrences of E are represented by the consecutive numbers **2**,**3**,**4** and **5**. A long key derived from sentences or expressions should contain at least several occurrences of such patterns. This also means that an extreme case of a key, such as "21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5,

4, 3, 2, 1" is unlikely to be derived from a sentence or expression. Furthermore, the longer the key phrase the higher the probability of the numerical value of each key element being closer to the position in the alphabet of the letter it represents. For example, if a numerical key has 25 elements and one of the elements is 5, it is more likely to represent an original E (or D or F), than a T (or S or U). We adapted the K_2 scoring function, so that it takes those patterns into consideration and assigned them additional scoring points in addition to the IDP. While we could see some improvement – mainly with the speed of the convergence of the algorithm – it still was not enough to solve the challenge. To fully quantify the benefits of this optimization, further research is needed.

The second optimization was a seemingly minor one, but eventually with dramatic consequences. In simulations, we could often see cases where although the algorithm would fail to produce a complete solution, it sometimes produced either a partially correct K_2 or a partially correct K_1 and even sometimes a fully correct K_1. In those cases, the trigram score of the decrypted plaintext with partially correct keys would be relatively higher than a "noise" trigram score resulting from random keys. A closer look at the resulting decrypted plaintexts showed that sometimes some fragments of words were discernible. This caught our attention and we wanted to take advantage of such partial solutions. We added an offline task which processes decrypted texts with higher trigram scores that were produced by hill climbing. It would scan those texts for possible words against a database of 10000 words. To recognize possible words from fragments, we used a modified Levenshtein distance criteria. The standard Levenshtein measure takes into account possible wrong letters, missing letters, or displaced letters, but only in a sequential text. In transposition ciphers, displacement can often occur into cells in the rows right above or below the correct row and not just on its right or left. We modified the distance calculation accordingly, and the distance threshold was set to 3 or below. Additionally, we considered only words with 6 or more letters. During the development process, we kept one server (3.4 GHz Intel Core i7 PC with 8 cores) running with the latest version of hill climbing against the challenge cipher, checking each one of the possible 14 combinations of the key lengths.

9.4.6.2 First Breakthrough

While the server was running with our improved algorithm and with both optimizations in place, we periodically checked the corresponding logs. Most of the time, common words, such as "letter", were produced. This was most probably due to the fact that 4 of the 6 letters are high-frequency letters, appearing multiple times in the ciphertext. Thus, such particular assemblies of letters were most likely to be the result of random displacement and wrong keys. At some stage, however, an interesting and longer word was spotted. The word was "pernicious". And another word in the vicinity of the first word was "sphere", which also could be "atmosphere". We started a "detective work" to follow this lead.

In [38], Klaus Schmeh wrote that the text is at least 100 years old, so it was likely to be from a book. The word "pernicious" is not a very common word in modern English texts, so we tried a Google search for that word on the Gutenberg project. This produced a large number of books. To narrow down the search, we used both "pernicious" and "sphere" for the search, as well as "pernicious" and "atmosphere". The Google search for this last combination showed 6 books in the Gutenberg project. We downloaded the books and looked at candidate segments of length 599, on which we applied the known-plaintext attack from Step 2. We obtained almost instantly a full match with a paragraph from *Mistress Wilding* by Raphael Sabatini. We had finally solved the challenge, as we now had the ciphertext and the two numerical keys K_1 and K_2. The lengths of the keys turned out to be 21 for K_1 and 23 for K_2, not too far away from 20. A closer look

showed that 23 might not have been an optimal choice as $599 = 26 \cdot 23 + 1$ resulting in an almost complete K_2 transposition rectangle. This is still harder than the very weak case with key lengths 24 and 25, i.e. $599 = 24 \cdot 25 - 1$. However, probably it is not the most secure choice. In November 1913, we sent the full plaintext with the numerical keys to Klaus Schmeh, who acknowledged the solution. In the following we present the first results of our long journey, i.e. the decrypted plaintext and the numerical keys with a corresponding representation as letters:

```
THEGIRLHADARRIVEDATLUPTONHOUSEAHALFHOURAHEADOFMISSWESTMACOTTANDUPONHERARRIVALSH
EHADEXPRESSEDSURPRISEEITHERFEIGNEDORREALATFINDINGRUTHSTILLABSENTDETECTINGTHEALA
RMTHATDIANAWASCAREFULTOTHROWINTOHERVOICEANDMANNERHERMOTHERQUESTIONEDHERANDELICI
TEDTHESTORYOFHERFAINTNESSANDOFRUTHSHAVINGRIDDENONALONETOMRWILDINGSSOOUTRAGEDWAS
LADYHORTONTHATFORONCEINAWAYTHISWOMANUSUALLYSOMEEKANDEASELOVINGWASROUSEDTOANENER
GYANDANGERWITHHERDAUGHTERANDHERNIECETHATTHREATENEDTOREMOVEDIANAATONCEFROMTHEPER
NICIOUSATMOSPHEREOFLUPTONHOUSEANDCARRYHERHOMETOTAUNTONRUTHFOUNDHERSTILLATHERREM
ONSTRANCESARRIVEDINDEEDINTIMEFORHERSHAREOFTHEM
```

The first numerical key K_1 is "16,10,15,17,1,20,21,11,19,5,14,12,8,3,7,4,2,18,6,13,9" which can be represented with letters as K_1 = "PJOQATUKSENLHCGDBRFMI". The original cleartext from above was first transposed with this key K_1, resulting in the following interim ciphertext:

```
IHMHSGTGCVECNGDYWMOEETOHAURNEDUTEEFTTOEHYUNGOUORGHAOHOTSOIFALILSTTANTOATAMSEDAI
ITMENMEOOHETNARNDRROAFNLEUTNINTSEFDAERURLLUASTSESOINTHCOPFEUSIETAAATNEIIHRFHTDO
UIYTETSUALRHVHCSSAEHHNEOFLRTAENATACOERCERLMARABMOMOSDNUHOANRHDNPOHAIUEDEEIENTRN
EHMACLGNRTNTENAIRHTDPPFGTWHOEFVWLIYAANNEONOHEHHSSADDTEECUEEDGRTKEIIEHEYFOETRATV
PEARTDIENOOTWEAETERUHDRTLHNDHDTANEAHSOWNANAEAOASULREAHSIRRLALNTHANONSDOHEVELRNT
NMEOOORERIAETLAIIANSSNDEFSDNEDATPAUXRNCAOMDRARSELWDAECMATTVSGNFNEIUNSRHIINLDAOR
GHDRPCRRIRARTHIIDDWREOTEERSVGHRAUOTIARAEWRSOIAFEEDSDSTHADTEMEOHONDHROIESNHTAORI
TRIHAEURROMERTEDOLUSREESHRIQTNINOYESWNRTRRHMEF
```

The numerical key K_2 is "19,21,6,20,17,14,4,7,22,1,15,2,8,18,12,9,23,13,5,10,16,3,11" and can be represented in letters as K_2 = "SUFTQNDGVAOBHRLIWMEJPCK". Transposing the interim ciphertext with this K_2 yields the original ciphertext:

```
VESINTNVONMWSFEWNOEALWRNRNCFITEEICRHCODEEAHEACAEOHMYTONTDFIFMDANGTDRVAONRRTORMT
DHEOUALTHNFHHWHLESLIIAOETOUTOSCDNRITYEELSOANGPVSHLRMUGTNUITASETNENASNNANRTTRHGU
ODAAARAOEGHEESAODWIDEHUNNTFMUSISCDLEDTRNARTMOOIREEYEIMINFELORWETDANEUTHEEEENENT
HEOOEAUEAEAHUHICNCGDTUROUTNAEYLOEINRDHEENMEIAHREEDOLNNIRARPNVEAHEOAATGEFITWMYSO
THTHAANIUPTADLRSRSDNOTGEOSRLAAAURPEETARMFEHIREAQEEOILSEHERAHAOTNTRDEDRSDOOEGAEF
PUOBENADRNLEIAFRHSASHSNAMRLTUNNTPHIOERNESRHAMHIGTAETOHSENGFTRUANIPARTAORSIHOOAE
UTRMERETIDALSDIRUAIEFHRHADRESEDNDOIONITDRSTIEIRHARARRSETOIHOKETHRSRUAODTSCTTAFS
THCAHTSYAOLONDNDWORIWHLENTHHMHTLCVROSTXVDRESDR
```

9.4.7 Step 5: Dictionary Attack

Besides improving the hill-climbing algorithm, we started in parallel to work on an improved dictionary attack. The newly developed IDP enabled a powerful K_2 dictionary attack with only $O(n)$ complexity, where n is the number of entries in the dictionary. The IDP also provided a speed improvement of a factor of $1:1\,200\,000$ versus the naive approach. First, we started with

an attack where the key phrase is taken from a book. For a typical book, containing 500 000 characters and an average of 5 characters per word, there are about 100 000 possible starting positions for a key phrase. The program produced solutions for such simulated cases in less than 10 seconds by processing 20 000 K_2 keys per second. In all of our simulations with keyphrases from books, the K_2 key derived from the correct keyphrase consistently produced the highest IDP score. While those trials confirmed the validity of using the IDP for a K_2 dictionary attack, for the challenge we did not know from which book the key phrases were taken, nor did we know whether at all they were from a book. So we looked for more comprehensive solutions in the form of a database of expressions and sentences. We used a mid-size free database of word n-grams from the Corpus of Contemporary American English [139]. We integrated this database, which contains about $n = 1\,000\,000$ entries, into our dictionary attack. The dictionary attack took only a few minutes to test all the expressions and phrases from this database. The one with the highest IDP score was "PREPONDERANCEOFEVIDENCE" (Preponderance of Evidence), which is a legal term unrelated to cryptography. This key phrase was equivalent to the numeric key found in Step 4 – "19,21,6,20,17,14,4,7,22,1,15,2,8,18,12,9,23,13,5,10,16,3,11". Hence, the challenge could be solved independently with a second method. We found this second solution two days after the first one (see Section 9.4.6.2).

Besides its $O(n)$ complexity, a major advantage of this K_2 dictionary attack is that it works regardless of the length of K_2. Obviously, it can find a solution only if the key phrase appears in the used database. However, databases with a large number of entries are available [140]. Furthermore, it is possible for organizations with massive computing power to enlarge existing databases by creating more combinations and using syntax analysis to rule out low-value combinations. We now had the key phrase for K_2. The last piece of the puzzle still missing was the key phrase for K_1.

9.4.8 Step 6: Wrapping-Up – Finding the Last Key Phrase

A first attempt to find the key phrase for K_1 using the reduced 1 000 000 entry database was not successful. Either a bigger database was needed or we needed outside help. In *Decoding the IRA* [61], Jim Gillogly describes a similar challenge he faced while solving the IRA's single transposition messages. While he could find numerical keys for the ciphertexts, he still needed to determine the original key phrases. This would provide a better understanding of the IRA communications procedures, such as whether certain books were used, as well as helping in solving other ciphertexts. The problem of recovering a key phrase from a numerical key is similar to recovering a "hat" in the famous NSA "headline puzzles" [141]. In his book, Jim Gillogly mentions the name of Jude Patterson as an expert in those kinds of puzzles. We contacted her, and she provided useful tips on how "hats" can be solved. Based on this information, we applied a combination of manual and computerized methods, which produced the matching sentence, "TOSTAYYOUFROMELECTION". The phrase "TOSTAYYOUFROMEJECTION" also matched.

We were not sure if this might really be the correct phrase. We turned for more help to Jim Gillogly and he applied the powerful tool which he developed for *Decoding the IRA*. One day later, he came up with the same and unique solution "TOSTAYYOUFROMELECTION" and he also was able to trace its source. This was the last line of the opening of scene II of the "*Merchant of Venice*" from Shakespeare. The last question was: What may Shakespeare/The Merchant of Venice, and Preponderance of Evidence have in common? They seemed totally unrelated at first glance. Further research on Google produced an interesting document named "*Who Wrote Shakespeare? The Preponderance of Evidence*" by Richard F. Whalen. This is a paper discussing the controversy about whether Shakespeare really wrote Shakespeare. With this, we

could confirm that we had the final missing element, and that the solution was complete. Klaus Schmeh later confirmed that during the period of time when he created the double columnar challenge, he was studying the Shakespeare controversy.

9.5 Epilogue

On December 25, 2014, a few weeks after the solution of the Double Transposition Challenge, we received an email from Otto Leiberich (see Section 9.3), with congratulations, as well as new details about his experience with the double transposition cipher. Leiberich had originally suggested publishing a challenge for the double transposition [133].

> Dear George Lasry
>
> What exciting news, deciphering the "Doppelwürfel"! Without knowing the details of your method, I congratulate you for this great success, this will find its place in the history of "Entzifferung" (codebreaking).
>
> When I was responsible for "deciphering" in the German Code Authority, the "Zentralstelle für Chiffrierwesen", we had not yet been confronted with it. The East German Intelligence-Service suddenly introduced it, to encrypt the messages to their spies in the Bundesrepublik. Hundreds of messages were transmitted every month. All the messages were received and stored by our Security Agencies. They contacted us and asked for our support. Now it was our problem. We observed the frequency of the "Geheimzeichen" (symbols) which corresponded to German plaintext. That meant a transposition method was used, and, because we could not find any "Parallelstellen"! (repetitions), we assumed a combination of basic transposition methods. But which? Then suddenly we got support from the police, they succeeded in catching and arresting a spy. And in his papers were found the complete Crypto instructions for the "Doppelwürfel" (double transposition cipher).
>
> And more than that: The "Schlüssel-Losungen" (key phrases) for K_1 and K_2 were based on parts of German literature. But how to find them? But we had an important advantage, the selected texts seemed to obey a certain structure. In a huge effort, visiting many books and libraries, we found finally more than 70% of all "Schlüssel-Losungen" used. Many messages could be "entziffert" i.e. reconstructed into the original plaintext, with considerable political consequences. This success was one of the most important of the young "Zentralstelle für Chiffrierwesen".
>
> But very soon the game ended, in 1965. In a trial against a spy, our expert was forced to reveal all crypto details. A short time later the "Doppelwürfel" was replaced by the unsolvable "i-Wurm" (OTP). This is the short history of Doppelwürfel.
>
> Now, dear George Lasry, I must apologize for my age. I am 86 years old now, and have forgotten. Later, I was promoted to head of the mathematical department and later to head of the "Zentralstelle" and finally, after additional tasks, to President of BSI, the German Agency for IT security. And, you will understand, very seldom, I found time for discussions with my old mathematical colleagues.
>
> Now, once more, I congratulate you for your great success and wish you a good year 2014
>
> Yours

Otto Leiberich

9.6 Summary

The main goal of this work was to develop new methods with the purpose of solving the double columnar challenge from Klaus Schmeh. We were able to identify several new general vulnerabilities of this classical cipher. The primary finding is that is possible to find one of the two keys (K_2) independently from the other (K_1). We achieved this with our new scoring function, the *Index of Digraphic Potential* (IDP). With this function we can evaluate K_2 without the knowledge of K_1, in a deterministic way.

This has the practical effect of almost nullifying the effect of the second transposition. The IDP allows for a highly efficient K_2 *key-phrase dictionary attack*. This dictionary attack is not dependent on the key lengths, and its complexity is only $O(n)$ for a dictionary size of n instead of $O(n^2)$. Thus, any text encrypted using key phrases from books, websites, or common expressions, may be susceptible to this attack and potentially easily decrypted. Clearly, randomly chosen numerical keys prevent this attack, but they are also harder to memorize.

The IDP also allows for an improved *hill-climbing ciphertext-only attack*. This method raises the bar for the current lower limit of key lengths required to achieve security. Until now the longest solvable key length was 13-15. With our new algorithm we can now solve messages encrypted with key lengths – either random or from key phrases – up to a length of 20.

Finally, we identified additional cases of *weak transposition rectangles*. For instance, the case of an almost perfect rectangle should be avoided in addition to those already mentioned in the classical literature.

Even though the identified vulnerabilities are significant, the double transposition still can still be considered secure and its cryptanalysis challenging when it is used with the right parameters. The team from MysteryTwister C3 already published a new double transposition challenge series with additional requirements designed to overcome the vulnerabilities exposed in this work [142]. Another possible follow-up from this work could be to try and apply the techniques presented here on historical messages, which are unsolved if such messages can be found in libraries or archives.

The work on solving the Double Transposition Challenge was an early work as part of the research covered in this thesis (together with the work on the single Columnar Transposition cipher described in Chapter 5). The insights from this work significantly contributed to building the foundations for the new methodology, and the formulation of its five principles, as described in Chapter 4.

Hill climbing, combined with multiple restarts, and specialized scoring functions and transformations, proved to be a highly effective tool for a hard cryptanalytic problem. The combined keyspace of two transpositions keys longer than 20, proved to be too large for a simple hill-climbing search for both keys at the same time, and a divide-and-conquer approach was required. Developing such an approach proved to be the most challenging part of the project. A divide-and-conquer approach was finally made possible with a highly specialized scoring function, the IDP.

The development of the IDP required a deep understanding of how columnar transposition affects the placement of the original plaintext characters, gained from working on single columnar

transposition ciphers. When implementing and testing the IDP, a better understanding of the attributes that make a scoring function effective was acquired. In the work described here, we refer to *selectivity* and *monotonicity*. We further refined those attributes, and the new methodology also refers to selectivity but with an extended scope, and to *resilience to key errors*. We extended the definition of selectivity to address the issue of spurious high scores, and instead of only requiring monotonicity, we emphasized the requirement for resilience to key errors, that is, the ability for a scoring function to stay monotonic up to a moderate number of key errors. The power of the IDP as a specialized scoring method, was further illustrated by enabling a second attack (a dictionary attack), in addition to the hill-climbing attack.

Also, the importance of implementing high-coverage but non-disruptive sets of key transformations was made evident, by the need to include segment-based transformations together with simple swaps.

Finally, the benefits of using multiple restarts with high-quality initial keys were also made clear.

All those insights played a major role in the formulation of the methodology and its principles.

In Table 9.1, we summarize how those methodology principles are reflected in our new attack on the double transposition cipher.

Principle	Application of the methodology principle
GP1	Sequential search with three phases, one phase to generate optimal initial K_2 keys, the second phase to search for K_2, and the last phase for K_1.
GP2	Divide-and-conquer, search for K_2 then for K_1
GP3	Specialized IDP score for K_2 allowing for the divide-and-conquer attack IDP also applicable to dictionary attack
GP4	Non-disruptive transformations applied on key segments Variable neighborhood search
GP5	Multiple restarts, with a first phase to generate optimal initial keys, using a reduced version of the main HC (the “left-to-right” optimizations)

TABLE 9.1: Double transposition – applying the methodology

10

Case Study – Cryptanalysis of Enigma Double Indicators

The Enigma machines were a series of electro-mechanical rotor cipher machines developed in Germany and used in first half of the twentieth century to protect commercial, diplomatic and military communications.

Until 1938, the German Army used the so-called "double indicator" procedure to transmit Enigma-encoded messages. It was replaced in September 1938, by a new procedure, also involving double indicators. Both procedures enabled a team of mathematicians from the Polish Cipher Bureau, to recover the wiring of rotors, and to develop cryptanalytic methods for the recovery of the daily keys. The double indicator procedure was discontinued in May 1940, and new methods were developed by the British in Bletchley Park, assisted by the knowledge transferred to them by the Polish cryptanalysts.

In Section 3.5.1, we presented historical attacks, as well as modern attacks, based on hill climbing, which do not take advantage of the key settings procedures and the double indicators in use until May 1940. In this chapter, we introduce two new algorithms, which extend historical cryptanalytic attacks on the two variants of the double indicator procedures. To develop those new attacks, we applied the principles of our methodology, described in Chapter 4. Those attacks are based on hill climbing, divide-and-conquer, and specialized scoring functions.

This chapter is structured as follows: In Section 10.1, we describe the functioning of the Enigma, focusing on Enigma I, the model employed by the German Army (Heeres) and Air Force (Luftwaffe). In Section 10.2, we present the size of the keyspace for those models. In Section 10.3, we describe the procedure in place until 1938, and in Section 10.4, we present a historical attack, developed by the Polish Cipher Bureau in the 1930s, which takes advantage of this procedure. After that, we present the procedure in place between 1938 and 1940 (in Section 10.5), and an historical attack for this procedure (Section 10.6). In Section 10.7, we present our two new attacks, compare them to the historical methods, and describe how we employed those attacks to win an international Enigma contest organized in 2015 by the City of Poznan, in memory of the Polish Bureau mathematicians and their exploits (Section 10.8). We summarize our results in Section 10.9.

10.1 Functional Description of the Enigma

In this section, we describe the functioning of the Enigma I, the standard 3-rotor Enigma model used by the German Army and Air Force, shown in Figure 10.1.

FIGURE 10.1: Enigma – Model I (courtesy of www.cryptomuseum.com)

Encryption by Enigma is based on substitution. Each input letter is replaced by another one. But unlike a simple monoalphabetic substitution, the mapping to the letters change after each letter has been encrypted, using a set of rotatable wheels, or simply, *rotors*. Each rotor has two sides, with 26 electrical contacts on either side. Each contact on one side is internally wired to one of the contacts on the other side. The wiring is such that if an input letter x is wired to an output letter y, then the input letter y is also wired to output letter x. However, no input letter is wired to itself (to the same letter at the output of the rotor).

Most models have three rotors, left, middle, and right (the exception being the M4 model, used later in the war exclusively for the German U-Boats, which has 4 rotors). Each rotor has a notch (in some models, more than one), which affects the movement of the next rotor on its left side. Each time a key is pressed, the rightmost rotor moves by one step, and based on the position of its notch, the middle rotor may also move by one step. Similarly, based on the position of the notch in the middle rotor, the leftmost rotor may also move by one step. Rotor movement is much like the odometer in a car.

As a result of the movement of the rotors, each new letter is encoded differently. For example, if A was encoded as R, then if A is pressed again, it might be encoded as N or any other letter. This effectively produces a polyalphabetic cipher, but with a large cycle, so that "depths" (messages encrypted with the same settings) can be avoided.

At first, there were three types of rotors, `I`, `II`, and `III`. Their order could be selected, i.e. their assignment to the left, middle, and right positions. On December 15th, 1938, two new rotor types were introduced, `IV` and `V`. The three rotors could now be selected from the five types.

Each rotor has a ring, on which either the letters A to Z, or the numbers 01 to 26 are engraved. When the rotor is in place and the machine is closed, the letter/number at the current position is displayed in a small window. Near this window is a protruding thumbwheel, which can be moved up or down, to set the rotor starting position, as can be seen in Figure 10.2. The movement of the rotor is counter-clockwise, when looking from the right side of the machine. If the letter A was shown in the window, after movement, the letter B will be shown (or similarly, 00 will be followed by 01).

FIGURE 10.2: Enigma – setting the rotors (courtesy of www.cryptomuseum.com)

When a rotor is removed from the machine, its ring can be manually rotated, with respect to the rotor wiring. The effect of rotating the ring is to apply an offset, in the opposite direction of the rotor movement. The turnover notch is also fixed to the ring. Therefore the turnover of the next rotor (on its left, when inserted into the machine), will always happen at the same letter in the window.

We now describe the internal process of encrypting a letter, as described in the simplified diagram, in Figure 10.3. The keyboard consists of 26 keys, marked A-Z. Whenever a key, say Q, is pressed, the rightmost rotor always moves by one step, and depending on the position of the notches, the middle rotor and the left rotor may also move, before encryption. After that, an electrical contact is closed. As a result, current from the battery will flow. The wires from the 26 keys are connected via a plugboard (*Steckerbrett*, see below) to a static wheel called *Eintrittswalze* (ETW). In the Enigma model, the ETW has through connections, that is, input A is wired to output A, B to B, and so forth. In other models, the wiring of the ETW was different.

Leaving the ETW, the current enters the rightmost rotor (1) via one of the contacts on its right-hand side. The internal wiring of that rotor 'translates' this current to one of the contacts on the left side of the rotor. From there the current is 'handed over' to the next rotor, and so on. Left of the rotor is the Reflector, or *Umkehrwalze* (UKW). In the Army version of the Enigma, the reflector (Umkehrwalze) had several versions, first, Umkehrwalze A, superseded by Umkehrwalze B in 1937. This reflector sends the current back into the rotors, but this time the current flows from left to right. Obviously, if a letter x is wired by the reflector to another

letter y, then this letter y is also wired to x. More importantly, no input letter is wired to itself by the reflector, as this would generate a short circuit. From the reflector, the current flows back from left to right through the rotors, until it reaches the ETW again. From the ETW the current goes again to the plugboard, and from the plugboard to the lamp board where the corresponding letter (E in the example) will be lit.

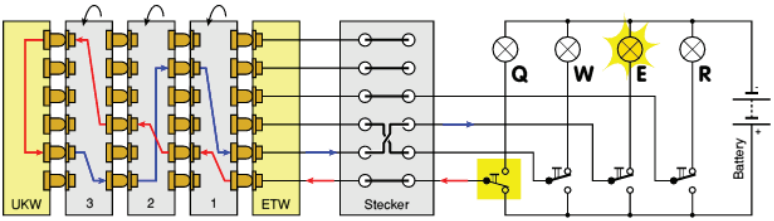


FIGURE 10.3: Enigma – simplified diagram (courtesy of www.cryptomuseum.com)

The plugboard (see Figure 10.4) implements a static substitution, at both the input (after the keyboard), and at the output (before the lamp board). Any number of cables from 0 to 13 may be connected to the Steckerbrett, meaning that between 0 and 13 letter pairs may be swapped. If a letter is not mapped (i.e. no stecker is used for that letter), the letter is known to be *self-steckered*. The wires have two plugs at each end, designed so that the input and output mappings are identical, that is, if an input letter x is mapped by the plugboard to output letter y, then y will also be mapped to x.

FIGURE 10.4: Enigma – plugboard (courtesy of www.cryptomuseum.com)

It is inherent to this design, with the back and forth path via the reflector, that a letter can never be encoded into itself. Moreover, encryption and decryption are always identical, at any given position of the rotor. This is made by design, as the developers of Enigma wanted to simplify its operation, so that a plaintext could be encrypted, and its corresponding ciphertext decrypted, with two machines with exactly the same settings.

Next, we describe the process of setting the Enigma for the transmission of a message. We limit our description to the case of the German Army Enigma, which had three rotors. The settings of the Enigma consist of four parts:

1. The *rotor order* (*Walzenlage*), the selection and ordering of the rotors: Until 1938, the order of the three rotors, `I`, `II`, and `III`, had to be determined from $3! = 6$ possibilities. On December 15th, 1938, two new rotor types were introduced, `IV` and `V`, and three rotors had now to be selected from five, increasing the number of options to $5 \cdot 4 \cdot 3 = 60$.
2. The *ring settings* (*Ringstellung*), the relative position of the ring with respect to the internal wiring of the rotors. It also changes the position of the turnover notch with respect to the internal wiring of the rotors.
3. The *plugboard connections* (*Steckerverbindungen*): Those affect the connections between the input keyboard and the rotors (via the ETW), and between the rotors (also via the ETW) and the output lamps. The plugboard was designed so that the input and output mappings were identical, using special cables with two plugs of different sizes on each side the cable (see Figure 10.5). At first, the German Army used 6 connections, which results in 12 letters being changed, and 14 left unchanged. Later, this number was increased to 10, with only 6 letters left unchanged.
4. The *rotor settings* (*Grundstellung*): The position of each rotor at the beginning of sending or receiving a message, which is set by the operator using the protruding thumbwheel, which can be moved up or down, as can be seen in Figure 10.2.

FIGURE 10.5: Enigma – plugboard cable (courtesy of www.cryptomuseum.com)

Example of Enigma settings:

- Rotor order: `I III II` (left, middle, and right, respectively).
- Ring settings: `ABC` (left, middle, and right, respectively).
- Plugboard connections: `(AR)(BY)(CO)(HX)(IN)(MZ)`.
- Rotor settings (starting position of rotors): `AOR` (left, middle, and right, respectively).

10.2 Keyspace of the 3-Rotor Enigma

With 6 possible rotor orders (three rotor types), there are 6 possible rotor orders. With the five rotor types, there are 60 options for rotor selection and ordering.

There are $26 \cdot 26 \cdot 26 = 17\,576$ possible starting positions (rotor settings) for the rotors.

In theory, there are also $26 \cdot 26 \cdot 26 = 17\,576$ possible ring settings, but as the notch on the leftmost rotor has no effect (the notch of a rotor affects the stepping of the next rotor on its left), only $26 \cdot 26 = 676$ are relevant.

The number of possible plugboard connections depends on the number of connections, as follows:

$$\frac{(26 \cdot 25) \cdot (24 \cdot 23) \cdot (22 \cdot 21) \cdot \ldots \cdot (26 - 2n + 2) \cdot (26 - 2n + 1)}{2^n \cdot n} \tag{10.1}$$

For $n = 6$ connections, their number is 100 391 791 500. For $n = 10$ connections, their number is 150 738 274 937 250.

Therefore, for $n = 6$ and three types of rotor, the size of the keyspace is = 7 156 755 732 750 624 000, or approximately 2^{63}.

For $n = 10$ and five rotor types, the size of the keyspace is approximately $2^{76.5}$.

10.3 Double Indicators – Procedure until 1938

The order of the rotors, the ring settings, and the plugboard settings were changed periodically. Initially, they were kept for months or weeks, but eventually they were replaced every day. They remained constant for all messages sent on a given day. Daily keys were distributed in advance, usually on a monthly basis. Before September 1938, the daily key also included the *base rotor settings*, used to encrypt the per-message rotor settings, as described below.

We illustrate the procedure with the following daily key: `I III II` (rotor order), `ABC` (ring settings), `(AR) (BY) (CO) (HX) (IN) (MZ)` (plugboard connections), and `AOR` (base rotor settings). At the beginning of each day, the operators needed to set up the Enigma machine according to the daily key, that is, for our example:

- Selecting the three rotors. In our example, those are `I`, `III`, and `II`.
- Setting the ring for each rotor. Ring position `A` for rotor `I`, position `B` for rotor `III`, and position `C` for rotor `II`.
- Opening the machine cover and inserting the rotors, `I` as the leftmost, `III` as the middle rotor, and `II` as the rightmost, then closing the cover.
- Connecting the plugs in the plugboard: between `A` and `R`, `B` and `Y`, and so forth.

We describe here the procedure for encrypting an individual message:

- Applying the *base* rotor settings, using the protruding thumbwheels. In our example, they are `AOR`.

- Selecting (randomly) the rotor settings for the specific message, e.g. SSC, *but not yet* setting the machine with it.
- Typing *twice* the message rotor settings, to encrypt them. In our example, SSCSSC, after encryption, gives VHWQUS. This is the *message indicator*.
- Applying the *message* rotor settings (in our example, SSC). The machine is now ready for the encryption of the message.
- Typing the plaintext to obtain the ciphertext.
- Transmitting the indicator (in our case, VHWQUS) as part of the message preamble, and then transmitting the ciphertext.

To decrypt a message:

- Applying the *base* rotor settings. In our example, AOR.
- Extracting the indicator from the received preamble (in our example, the indicator is VHWQUS).
- Typing that indicator (VHWQUS) to decrypt it. In our example, we obtain SSCSSC after decryption, that is, the original message rotor settings, reproduced twice.
- Applying the *message* rotor settings (in our case, SSC). The machine is now ready for the decryption of the ciphertext.
- Typing the ciphertext to obtain the message original plaintext.

Originally, the goal of sending twice the message rotor settings (encrypted), was to cope with possible transmission or reception errors. In practice, this was not necessary, as demonstrated when the procedure was discontinued. In addition, the operators were instructed to select random rotor settings for each message. In practice, non-random settings such as CCC or YXZ, or sequences of neighboring letters in the keyboard, were often used by operators, due to lack of discipline.

10.4 Rejewski's Method

At the beginning of the 1930s, the mathematician M. Rejewski from the Polish Cipher Bureau was assigned with the task of identifying the details of the Enigma system in use by the German Army. He was aided in the process by information obtained from a spy operated by French intelligence. This included manuals, technical information, as well as sets of daily keys. The Polish Cipher Bureau also had access to the commercial version of the Enigma, and hundreds of ciphertexts intercepted daily.

After analyzing the traffic, Rejewski was able to reconstitute the indicator procedure, and he concluded that the double encryption and transmission of message rotor settings, using the same daily key, was a weakness that might be exploited. Rejewski applied the theory of permutations to the problem, and through as a series of ingenious guesses and deductions, he was able to recover the details of the wiring of all the rotors [53].

Still, the Polish Cipher Bureau needed a method to recover the daily keys. As part of his work to recover the rotor wirings, Rejewski discovered that it is possible to map the relationship between letters of the encrypted double indicators, into permutations represented by disjoint cycles. He then developed a method to recover the daily keys [53] [143]. We illustrate Rejewski's method using a sample set of indicators encrypted with the following daily key:

`I III II` (rotor order), `ABC` (ring settings), `(AR)(BY)(CO)(HX)(IN)(MZ)` (plugboard connections), and `AOR` (base rotor settings).

On this particular day, the following indicators have been intercepted:

```
VHWQUS NDLUJK QTKEWU HLOKOG AFRSDQ GYWORS YSMGZW WUMBIW IHHAUV RRYTCA
BYNRRZ MCQHTR PFHPDV CJLCQK IAWAYS DTNIWZ CVJCPF AWFSGJ IKLAMK EPTMAP
RNPTSD KSDWZI TSEZZB MABHYL PMMPEW GNUOSY EGBMNL RJFTQJ NKCUMN XGRDNQ
JKXJME KFLWDK XQKDLU DUXIIE PAIPYO ZFHVDV UXNXVZ FXRFVQ APUSAY BGARNM
OAOLYG BXPRVD EGEMNB GIDOKI UWLXGK IHBAUL PEWPBS HOYKHA JXOJVG OHKLUU
YLNGOZ YLCGON JIYJKA WVCBPN JYQJRR QWLEGK KQCWLN LRDNCI MDDHJI PFZPDH
DBSIFX CNFCSJ UWTXGP EFOMDG JHCJUN MPDHAI JNVJST TEUZBY AZKSXU MUYHIA
NHOUUG WZLBXK ZXHVVV YEFGBJ VMVQET VUFQIJ TISZKX RISTKX RZXTXE ITKAWU
```

Assuming the Enigma machine has been set with the daily key, including the base rotor settings, we define the permutation A_1 as the permutation representing the Enigma encryption at the first position. It is used to encrypt the first letter of the rotor settings (three letters) of each message. Similarly, we define A_2 and A_3 for the encryption permutations at the second and third positions, used to encrypt the second and third letters, respectively, of each one of the message rotor settings. At the fourth position, the first letter of the rotor settings for the message is again encrypted, using A_4, and similarly, the second letter using A_5, and the third letter using A_6.

We illustrate this by encrypting the message rotor settings `SSC` typed twice, that is, `SSCSSC`, using the daily key specified above. We obtain the first indicator, `VHWQUS`. It therefore follows that $A_1(S) = V, A_2(S) = H, A_3(C) = W, A_4(S) = Q, A_5(S) = U$, and $A_6(C) = S$.

However, the cryptanalyst neither knows the message rotor settings (`SSC` in our example), nor the daily key. The cryptanalyst only has in his possession the list of indicators, which are the encrypted versions of various message rotor settings typed twice. He has no way of directly determining any the A_i permutations. The cryptanalyst, however, is able to reconstitute the *product* of A_4 and A_1, that is, $A_4 \cdot A_1$, as described here.

We denote the first letter of the unknown rotor settings as x. From the interception of the encrypted indicators, we can see, for the first indicator and its unknown first letter x, that $A_1(x) = V$, and $A_4(x) = Q$, and therefore $A_1^{-1}(V) = A_4^{-1}(Q)$. By applying A_4 on both sides, we obtain: $(A_4 \cdot A_1^{-1})(V) = Q$. But since encryption and decryption by Enigma are identical, it follows that $A_1^{-1} = A_1$, and therefore, $(A_4 \cdot A_1)(V) = Q$.

Similarly, we can also establish, based on the first indicator `VHWQUS`, that $(A_5 \cdot A_2)(H) = U$, and that $(A_6 \cdot A_3)(W) = S$. Based on the second indicator `NDLUJK`, we are able to establish that $(A_4 \cdot A_1)(N) = U$, $(A_5 \cdot A_2)(D) = J$, and $(A_6 \cdot A_3)(L) = K$. Given enough indicators, the cryptanalyst can fully reconstitute $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$.

For our example, after processing all the indicators, we obtain the following mappings, for $A_4 \cdot A_1$:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
SRCIMFOKAJWNHULPET?ZXQBDGV
```

From the indicators, we could not determine $A_4 \cdot A_1(S)$. But we can deduce that $A_4 \cdot A_1(S) = Y$, since Y is the only letter unused in the second row. We can therefore complete the full mapping for $A_4 \cdot A_1$, as follows, starting with the letters that map to themselves, followed by the disjunctive mappings:

$$A_4 \cdot A_1 = \texttt{(C)(F)(J)(P)(ASYGOLNUXDI)(BRTZVQEMHKW)} \tag{10.2}$$

$A_4 \cdot A_1$ has 4 cycles of length 1, and 2 cycles of length 11. We denote this cycle structure as $(1,1,1,1,11,11)$.

Similarly, the mappings for $A_5 \cdot A_2$ based on the indicators are as follows:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
YFTJBDNUKQMOESHALCZWIPGVRX
```

$$A_5 \cdot A_2 = \texttt{(AYRCTWGNSZXVP)(BFDJQLOHUIKME)} \tag{10.3}$$

The cycle structure for $A_5 \cdot A_2$ is $(13,13)$.

Finally, for $A_6 \cdot A_3$, the mappings based on the indicators are:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
MLNIBJ?VOFUKWZGDRQXPYTSEAH
```

After completing the missing mapping $A_6 \cdot A_3(G) = C$, we obtain:

$$A_6 \cdot A_3 = \texttt{(FJ)(QR)(AMWSXEBLKUY)(CNZHVTPDIOG)} \tag{10.4}$$

The cycle structure for $A_6 \cdot A_3$ is $(2,2,11,11)$.

We could theoretically perform a brute-force search, and survey all possible daily key settings, to find settings with A_1 to A_6 that match our expected $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, which we computed from the indicators. This amounts to a comprehensive survey of the whole keyspace, which is not practical even with today's technology, and certainly not with the technology available in the 1930s.

To simplify the problem, Rejewski ingeniously applied the following proposition from the theory of permutations ([144], page 126, Proposition 11):

*Two elements of S_n are **conjugate** in S_n if and only if they have the same cycle type.*

S_n is the set of permutations of length n. Two permutations K, L in S_n are called *conjugate* if there exists another permutation M in S_n such that:

$$K = M \cdot L \cdot M^{-1} \tag{10.5}$$

The cycle type, also called the *cyclic structure*, of a permutation, is the list of the lengths of all cycles in the permutation (in this chapter, we use the latter term).

The notation of the product of permutations $M \cdot L \cdot M^{-1}$ means that when applied to an argument x, the permutations which compose this product are applied from right to left, that is:

$$M \cdot L \cdot M^{-1}(x) = M(L(M^{-1}(x))) \tag{10.6}$$

M^{-1} is the *inverse* of M, that is, $M^{-1} \cdot M(x) = M \cdot M^{-1}(x) = x$, for any x.

To demonstrate how the proposition stated above applies to the Enigma problem with "double indicators", we denote S as the permutation resulting from the plugboard connections. As stated in Section 10.1, by design, if any input letter x is wired to output letter y, then input y is also wired to output$x$. As a result, the permutation S, representing the plugboard effect on input letters from the keyboard, and its inverse S^{-1}, representing the effect of the plugboard on letters from the rotors, are identical, that is $S^{-1} = S$.

We denote $A_1{}'$ as the permutation produced at the first position, only by the rotors, traversed up to the reflector, and back, without the effect of the plugboard (see Figure 10.6).

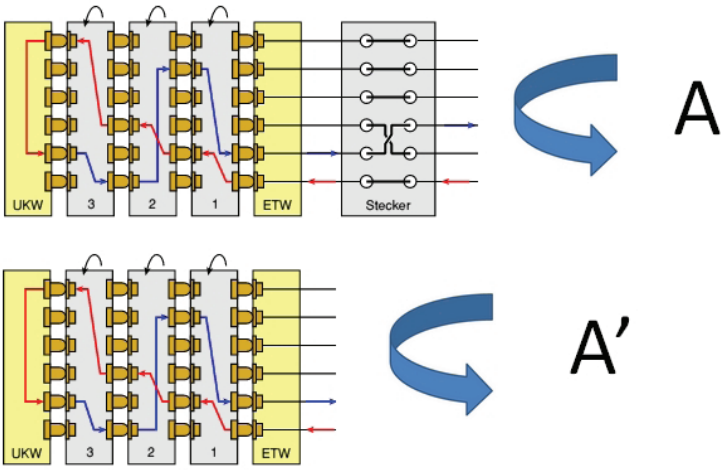


FIGURE 10.6: Enigma – permutations A and A'

After adding the effect of the plugboard at the input and at the output, it follows that:

$$A_1 = S^{-1} \cdot A_1{}' \cdot S \tag{10.7}$$

Similarly, we denote $A_4{}'$ as the permutation at position 4, produced only by the rotors, traversed up to the reflector, and back, without the plugboard:

$$A_4 = S^{-1} \cdot A_4{}' \cdot S. \tag{10.8}$$

It can be seen that $A_4 \cdot A_1$ and $A_4^{'} \cdot A_1^{'}$ are conjugate, since:

$$A_4 \cdot A_1 = (S^{-1} \cdot A_4^{'} \cdot S) \cdot (S^{-1} \cdot A_1^{'} \cdot S) = S^{-1} \cdot A_4^{'} \cdot S \cdot S^{-1} \cdot A_1^{'} \cdot S = S^{-1} \cdot A_4^{'} \cdot A_1^{'} \cdot S \tag{10.9}$$

and therefore:

$$(A_4 \cdot A_1) = S^{-1} \cdot (A_4^{'} \cdot A_1^{'}) \cdot S \tag{10.10}$$

and since $S^{-1} = S$, we obtain

$$(A_4 \cdot A_1) = S \cdot (A_4^{'} \cdot A_1^{'}) \cdot S^{-1} \tag{10.11}$$

According to the proposition stated above, it follows that *the cyclic structures of* $A_4 \cdot A_1$ *(which incorporates the plugboard) and of* $A_4^{'} \cdot A_1^{'}$ *(which does not incorporate the plugboard), are identical*.

With this important result, Rejewski could now reduce the problem of finding the full daily key settings to a much simpler problem. It is enough to find the rotor order, the ring settings, and the base rotor settings, ignoring the plugboard settings, so that the *cycle structure* of $A_4^{'} \cdot A_1^{'}$, $A_5^{'} \cdot A_2^{'}$, and $A_6^{'} \cdot A_3^{'}$ match the corresponding cyclic structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, computed from the encrypted indicators.

To further reduce the scope of the problem, Rejewski also made the assumption that the middle rotor would not step in the process of encrypting the $2 \cdot 3 = 6$ symbols of the rotor settings typed twice. As the middle rotors steps once every 26 times, that assumption was true for most of the daily keys, with $(26-6)/26 = 77\%$ probability. With this assumption, Rejewski could also ignore the ring settings (and assume they are `ZZZ`), and the number of options to be checked was reduced to $3! \cdot 26 \cdot 26 \cdot 26 = 105\,456$, instead of $3! \cdot 26 \cdot 26 \cdot 26 \cdot 26 \cdot 26 = 71\,288\,256$.

Rejewski and his team also discovered that in most cases, for a given rotor order, a combination of cyclic structures for $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$ matches only one possible set of rotor positions, or a small number of them, which could be tested manually. Using a device called "the cyclometer" [143], the Polish Cipher Bureau was able to produce a catalog of cyclic structures and their corresponding rotor settings, for each rotor order. All the elements of this ingenious attack were now ready [53] [143].

It should be noted, however, that this process is not as simple as it sounds. First, there needs to be enough indicators to reproduce the full cyclic structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$. Our simulations show that at least 70 to 90 indicators are needed to completely reconstruct $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, and their cyclic structures.

Furthermore, no stepping of the middle rotor may occur in the first 6 positions.

In addition, the combined set of cyclic structures (for $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$) may often match more than one set of rotor settings. In a recent study [145] using reflector A, Kuhl found that 21 230 different cycle structures may occur. Of these, 11 466 (54.40%) correspond to unique Enigma rotor settings. 20 433 correspond to 10 or fewer positions (or to none). These 20 433 account for 92.34% the possibilities. Rejewski was generally correct that there are very few ground settings that correspond to given cycle structures, but there are some extreme cases. The cycle structure `(13 13)(13 13)(13 13)`, for example, corresponds to 1 771 possible rotor settings.

For each set of rotor position (and given rotor order) which match the expected cycle structure, Rejewski and his team had to manually recover those plugboard settings, which would produce a correct decryption with the form `xyzxyz`, for all the indicators.

In our example, based on our simulations, we found that there are multiple rotor orders and rotor settings, which match the cyclic structure of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$. Only one of them correctly reproduces the original message rotor settings (after recovering the correct plugboard connections). For example, for the (wrong) rotor order `I II III`, there are 12 different rotor settings, such as `BRM`, which match the cycles structures. For the correct order, `I III II`, there are 14 rotors settings that match the cycle structures, 13 of them wrong, and the correct one being `ZMO` (assuming the ring settings are `ZZZ`).

After decrypting the indicators using the correct settings (daily key), that is, `I III II` (rotor order), `ZZZ` (ring settings), `(AR)(BY)(CO)(HX)(IN)(MZ)` (plugboard connections), and `ZMO` (base rotor settings), we obtain the original (doubled) message rotor settings, as follows:

```
VHWQUS ==> SSCSSC,  NDLUJK ==> WAPWAP,  QTKEWU ==> AMTAMT,
HLOKOG ==> XXXXXX,  AFRSDQ ==> QYJQYJ,  GYWORS ==> TFCTFC,
YSMGZW ==> ZHNZHN,  WUMBIW ==> NNNNNN,  IHHAUV ==> ESYESY,
RRYTCA ==> OBHOBH,  BYNRRZ ==> LFMLFM,  MCQHTR ==> DEFDEF,
PFHPDV ==> CYYCYY,  CJLCQK ==> PPPPPP,  IAWAYS ==> EDCEDC,
DTNIWZ ==> MMMMMM,  CVJCPF ==> PQRPQR,  AWFSGJ ==> QKQQKQ,
IKLAMK ==> EWPEWP,  EPTMAP ==> IJKIJK,  RNPTSD ==> OULOUL,
KSDWZI ==> UHBUHB,  TSEZZB ==> GHIGHI,  MABHYL ==> DDDDDD,
PMMPEW ==> CTNCTN,  GNUOSY ==> TUVTUV,  EGBMNL ==> IIDIID,
RJFTQJ ==> OPQOPQ,  NKCUMN ==> WWWWWW,  XGRDNQ ==> HIJHIJ,
JKXJME ==> FWOFWO,  KFLWDK ==> UYPUYP,  XQKDLU ==> HVTHVT,
DUXIIE ==> MNOMNO,  PAIPYO ==> CDECDE,  ZFHVDV ==> YYYYYY,
UXNXVZ ==> KLMKLM,  FXRFVQ ==> JLJJLJ,  APUSAY ==> QJVQJV,
BGARNM ==> LIZLIZ,  OAOLYG ==> RDXRDX,  BXPRVD ==> LLLLLL,
EGEMNB ==> IIIIII,  GIDOKI ==> TGBTGB,  UWLXGK ==> KKPKKP,
IHBAUL ==> ESDESD,  PEWPBS ==> CCCCCC,  HOYKHA ==> XZHXZH,
JXOJVG ==> FLXFLX,  OHKLUU ==> RSTRST,  YLNGOZ ==> ZXMZXM,
YLCGON ==> ZXWZXW,  JIYJKA ==> FGHFGH,  WVCBPN ==> NQWNQW,
JYQJRR ==> FFFFFF,  QWLEGK ==> AKPAKP,  KQCWLN ==> UVWUVW,
LRDNCI ==> BBBBBB,  MDDHJI ==> DABDAB,  PFZPDH ==> CYACYA,
DBSIFX ==> MRGMRG,  CNFCSJ ==> PUQPUQ,  UWTXGP ==> KKKKKK,
EFOMDG ==> IYXIYX,  JHCJUN ==> FSWFSW,  MPDHAI ==> DJBDJB,
JNVJST ==> FUUFUU,  TEUZBY ==> GCVGCV,  AZKSXU ==> QOTQOT,
MUYHIA ==> DNHDNH,  NHOUUG ==> WSXWSX,  WZLBXK ==> NOPNOP,
ZXHVVV ==> YLYYLY,  YEFGBJ ==> ZCQZCQ,  VMVQET ==> STUSTU,
VUFQIJ ==> SNQSNQ,  TISZKX ==> GGGGGG,  RISTKX ==> OGGOGG,
RZXTXE ==> OOOOOO,  ITKAWU ==> EMTEMT
```

Note that the key that was found with this process, `I III II` (rotor order), `ZZZ` (ring settings), `(AR)(BY)(CO)(HX)(IN)(MZ)` (plugboard connections), and `ZMO` (base rotor settings), is different from the original key, `I III II` (rotor order), `ABC` (ring settings), `(AR)(BY)(CO)(HX)(IN)(MZ)` (plugboard connections), and `AOR` (base rotor settings).

This is due to the fact that the combination of the ring settings `ZZZ` and of the rotor settings `ZMO`, is equivalent to `ABC` (ring settings) and `AOR` (rotor settings), since that the middle rotor does not step when encrypting the first 6 letters.

With this method, the Polish Cipher Bureau was able to recover the majority of the daily keys in use by the German Army, until September 1938, when a new procedure was introduced.

10.5 Double Indicators – Procedure from 1938 to 1940

On September 15th, 1938, the German Army changed their keying procedure for Enigma. The German cryptographers might have realized that the encryption and transmission (twice) of hundreds of message rotor settings using the same daily key, could compromise the security of Enigma. Instead of encrypting and transmitting the message rotor settings only once, they continue to do this twice, but with a new procedure. This new procedure was in place from September 1937 until May 1940, right before the major German offensive in the West started. In May 1940, the use of double indicators was terminated altogether.

Starting from September 1938, the daily key does not include any rotor settings. Instead, the daily key includes only the rotor order, the ring settings, and the plugboard connections. At the beginning of each day, the operators had to set up their Enigma machines according to this daily key.

We illustrate the new procedure with the following daily key:

`I III II` (rotor order), `ABC` (ring settings), `(DW) (EN) (GI) (KM) (OV) (UZ)` (plugboard connections).

The procedure for encrypting an individual message consists of:

- Selecting (randomly) some *initial rotor settings*, used only to encrypt the actual message rotor settings. In our example, we use `NDM` as the initial rotor settings.
- Applying those rotor settings (`NDM`).
- Selecting (also randomly) some rotor settings for the message (the *message rotor settings*), but *but not yet* setting the machine with it. In our example, we use `KDL` as the message rotor settings.
- Typing *twice* the message rotor settings, to encrypt it. In our example, `KDLKDL`, after encryption, gives `TOSSUC`.
- Combining the initial rotor settings, *in clear* (`NDM`), with the twice-encrypted message rotor settings (`TOSSUC`), to form the message indicator (`NDM TOSSUC`).
- Applying the message rotor settings (`KDL`). The machine is now ready for the encryption of the message.
- Typing the plaintext to obtain the ciphertext.
- Transmitting the indicator (`NDM TOSSUC`) as part of the preamble, and then transmitting the ciphertext.

To decrypt a message:

- Extracting the first 3 letters of the indicator, and applying them as the initial rotor settings (in our case, `NDM`, the first 3 letters of the indicator `NDM TOSSUC`).

- Typing the remaining 6 letters of the indicator (`TOSSUC`), to obtain the original – doubled – message rotor settings (`KDLKDL`).
- Applying the message rotor settings (`KDL`). The machine is now ready for the decryption of the ciphertext.
- Typing the ciphertext to obtain the message original plaintext.

10.6 The Zygalski Sheets

Rejewski's method is no longer applicable, as the message rotor settings are encrypted using different initial rotor settings. The Polish cryptanalysts needed to develop new methods for this new procedure. Henri Zygalski, together with other members of the team, was able to devise a new method which takes advantage of this new procedure.

We illustrate this method using a set of 80 sample indicators, created with the procedure in place between 1938 and 1940, and intercepted on a specific day:

```
NDM TOSSUC, MDR VBMSQP, GAM GKANNP, PEO PNJONU, HPC GMIWQK,
LRR YUODMG, FKQ FSTJOF, NQS ICBSDX, DXD BGBEFK, IZH RUNJDR,
ELQ SSADPZ, CNZ UJDVCE, IOZ PCUVUJ, SML JNVVRD, FHF JKCVCY,
VFZ QIQCPM, XSE WXQMTT, MVG NXKQUS, VVY GGHFDZ, KGH TEBOKY,
BDR SBKSHQ, TWX KAIXEY, DDV MACFWP, EPQ QSVTST, BIN MPMEWM,
FSY GBZWSH, HCR PAROUR, HKO FDIFAL, IYP RBQHST, VLX AASRFD,
TXP DJUGKN, FLI RDJWXP, POD SMNPOK, WDH YUEMMX, GVE IOLOTX,
CIW WDDDXD, OVA PPEAIV, UMT OSOIWC, BXQ FBBFEU, EIU BDWSTC,
BDW JHSOES, UNY VWZAPZ, VAD FLZNOV, YWX MICKJG, JVF NCYVRG,
TNJ QHEADW, FSF WLENDU, AFE EGZOFJ, YLL BTDKHG, UBD JCLYCE,
QOP QCQCGX, TDH XUXIPS, DUZ ZDFBTI, JDS FBGVDC, MJN EJBKGU,
AVT NFFZAC, TXT KXEXVJ, SCG DTBVLL, XKZ VXANUE, AFZ TNCNWK,
QVG TYMGLA, JKI NDTNUR, ROR UASHOP, QIG EYXCQM, ORL SSHOOH,
LRN NPZMCQ, KKW SZGNFE, YGT BENPHN, RYF LVEHWA, PBL AFRDMT,
TTP TPVKXL, NQB SXIBRB, IZO TXVMZX, WXY AIJPES, ZGF GYPNKY,
LPJ THSRFT, AKN ZXHOBH, OZS FWJAGU, AKU UTCJQF, MCU ZBZZUT,
```

Zygalski's method is also based on the analysis of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, but in a different manner. With the new procedure, for each indicator, we have different initial rotor settings, and therefore A_1 to A_6 are not the same for all the indicators. Instead, the Zygalski method relies on finding indicators with "females". A *female* occurs in an indicator, ignoring the first 3 letters of the 9 in total, when the same letter appears at position i and again at position $i+3$. In our example, we have females in `PEO PNJONU` (`N` appears in positions 2 and 5), or in `BDR SBKSHQ` (`S` appears in positions 1 and 4), for example.

We now consider a single indicator and its corresponding $A_4 \cdot A_1$. This product $A_4 \cdot A_1$ is specific to this particular indicator, and is the result of applying the daily key, and the message initial rotor settings. Those initial rotor settings are used to encrypt the message rotor settings, and they are sent in clear (as the first 3 letters of the indicator). For the combination of the daily key, and of some particular initial rotor settings, a female is *possible* if and only if the cyclic structure of $A_4 \cdot A_1$ with those settings contains at least one fixed point, that is, a cycle with a single element.

In other words, there exists at least one value of x for which $(A_4 \cdot A_1)(x) = x$, and the cycle `(x)` appears in the permutation $A_4 \cdot A_1$.

Zygalski computed the probability for any combination of the rotor order, the rings settings, and the rotor settings, to allow for *no* fixed point in its $A_4 \cdot A_1$, to be 0.5882. Therefore, the probability for at least one fixed point is $1 - 0.5882 = 0.4118$ (about 41%). As we saw in Section 10.4, we can ignore the plugboard settings and instead use $A_4' \cdot A_1'$, which has the same cyclic structure.

Zykalski's attack is performed separately for each rotor order. For each rotor order, we exhaustively check all possible ring settings. For each candidate ring settings, we look for *indicators with females*, either at positions $1-4$, $2-5$ or $3-6$. For each female indicator, we verify whether the candidate rings settings, combined with the initial rotor settings (the first 3 letters, in clear, from the indicator), allow for *any* female (with any input letter, A to Z). If those settings allow for a female, we cannot deduce anything. But if those settings do *not* allow for any female, we are able to rule out the candidate ring settings.

With each additional indicator having a female, we are able to rule out about 59% of the candidate ring settings, leaving only the other 41%. With 11 females, we can reduce the number of candidate ring settings, from $26^3 = 17576$ options, to only one option, as $(26^3) \cdot (0.4118^{11}) \approx 1$. Based on our simulations, the percentage of females in a given random set of indicators may range between 8% to 20%, and usually around 15%. To obtain 11 females, we need about 137 indicators in the worst case, 55 in the best case, or 73 on average.

To make the attack practical with the technology of the 1930s, Zygalski came up with the idea of using perforated sheets – the "Zygalski Sheets". Those can be aligned and superimposed, so it is possible to visually see those rotor positions (combination of the ring settings and the rotor settings), allow for females. The particular ring settings that allow for all the females found in the indicators, is a candidate, that is further tested to try and recover the plugboard settings. A detailed description of the Zygalski sheets and their use can be found in [143].

As with the Rejewski method, the initial Zygalski method also assumed that no turnover of the middle rotor occurred, but the Polish cryptanalysts later devised a method for taking those turnovers into account. The Zygalski sheets were produced with the help of the Cyclometer [143]. In total, $6 \cdot 26 = 156$ sheets were needed, each having $51 \cdot 51 = 2601$ squares. At the end of 1938, when the German Army introduced two new types of rotors (`IV` and `V`), 10 times more sheets were required to perform the process. As the Polish Cipher Bureau was unable to undertake such a massive task, this was one of the reasons they turned for help to their British and French counterparts, disclosing to them their progress on the cryptanalysis of the Enigma [143]. The British produced a new and extended set of sheets, and renamed them as the "Jeffrey Sheets". From the beginning of WWII until May 1940, when the use of double indicators was discontinued altogether, and before the first Turing Bombe was put in service, the Zygalski/Jeffrey sheets were the primary tool for recovering daily keys.

We conclude with our example given above. The only rotor order and ring settings that allow for all the female indicators in the set, are `I III II` and `ABC`, respectively. After recovering the plugboard settings, `(DW)(EN)(GI)(KM)(OV)(UZ)`, we obtain the following decryptions of the indicators:

```
NDM:TOSSUC ==> KDLKDL, MDR:VBMSQP ==> GHIGHI, GAM:GKANNP ==> QRSQRS,
PEO:PNJONU ==> LMNLMN, HPC:GMIWQK ==> VYNVYN, LRR:YUODMG ==> FFFFFF,
FKQ:FSTJOF ==> YYYYYY, NQS:ICBSDX ==> OOOOOO, DXD:BGBEFK ==> IJNIJN,
IZH:RUNJDR ==> AYYAYY, ELQ:SSADPZ ==> TTTTTT, CNZ:UJDVCE ==> KKKKKK,
IOZ:PCUVUJ ==> WWWWWW, SML:JNVVRD ==> OYBOYB, FHF:JKCVCY ==> MEQMEQ,
VFZ:QIQCPM ==> IFCIFC, XSE:WXQMTT ==> IFCIFC, MVG:NXKQUS ==> GEEGEE,
VVY:GGHFDZ ==> PIEPIE, KGH:TEBOKY ==> UUUUUU, BDR:SBKSHQ ==> MMMMMM,
TWX:KAIXEY ==> ZZZZZZ, DDV:MACFWP ==> PQRPQR, EPQ:QSVTST ==> FWGFWG,
BIN:MPMEWM ==> DEFDEF, FSY:GBZWSH ==> AAAAAA, HCR:PAROUR ==> YRFYRF,
HKO:FDIFAL ==> EEEEEE, IYP:RBQHST ==> UHBUHB, VLX:AASRFD ==> QQBQQB,
TXP:DJUGKN ==> SSSSSS, FLI:RDJWXP ==> SPBSPB, POD:SMNPOK ==> LLLLLL,
WDH:YUEMMX ==> QSSQSS, GVE:IOLOTX ==> RSTRST, CIW:WDDDXD ==> VFHVFH,
OVA:PPEAIV ==> IDGIDG, UMT:OSOIWC ==> KLMKLM, BXQ:FBBFEU ==> WSXWSX,
EIU:BDWSTC ==> GJUGJU, BDW:JHSOES ==> VXKVXK, UNY:VWZAPZ ==> EDCEDC,
VAD:FLZNOV ==> CDECDE, YWX:MICKJG ==> NNNNNN, JVF:NCYVRG ==> TJSTJS,
TNJ:QHEADW ==> TEFTEF, FSF:WLENDU ==> UXPUXP, AFE:EGZOFJ ==> TMCTMC,
YLL:BTDKHG ==> IQUIQU, UBD:JCLYCE ==> EFGEFG, QOP:QCQCGX ==> NRENRE,
TDH:XUXIPS ==> VQPVQP, DUZ:ZDFBTI ==> RORROR, JDS:FBGVDC ==> XUYXUY,
MJN:EJBKGU ==> OMXOMX, AVT:NFFZAC ==> TGBTGB, TXT:KXEXVJ ==> STUSTU,
SCG:DTBVLL ==> WOQWOQ, XKZ:VXANUE ==> ABCABC, AFZ:TNCNWK ==> BBBBBB,
QVG:TYMGLA ==> WWEWWE, JKI:NDTNUR ==> MSGMSG, ROR:UASHOP ==> ZGVZGV,
QIG:EYXCQM ==> JKLJKL, ORL:SSHOOH ==> GDGGDG, LRN:NPZMCQ ==> SZESZE,
KKW:SZGNFE ==> GADGAD, YGT:BENPHN ==> AYOAYO, RYF:LVEHWA ==> IPTIPT,
PBL:AFRDMT ==> LUPLUP, TTP:TPVKXL ==> ICNICN, NQB:SXIBRB ==> QNRQNR,
IZO:TXVMZX ==> QMOQMO, WXY:AIJPES ==> OKMOKM, ZGF:GYPNKY ==> UEGUEG,
LPJ:THSRFT ==> PQBPQB, AKN:ZXHOBH ==> RRRRRR, OZS:FWJAGU ==> WAPWAP,
AKU:UTCJQF ==> XODXOD, MCU:ZBZZUT ==> TQJTQJ
```

10.7 New Attacks on Double Indicators

In this section, we describe two new attacks which we developed to recover key settings from sets of indicators, for each double indicator procedure.

10.7.1 New Attack on Double Indicators – Procedure until 1938

We developed a new method for the cryptanalysis of Enigma (double) indicators based on the German procedure in use until 1938. Our new attack was designed under the guidelines of the new methodology from Chapter 4, employing a divide-and-conquer approach, hill climbing, and a specialized, simple but highly effective scoring method.

Our approach is described in Algorithm 10. The input is a set of indicators with 6 letters, each of them is an encryption of message rotor settings (typed twice), using a daily key (rotor order, ring settings, plugboard connections, and base rotor settings), which we need to recover. Similarly to Rejewski's method, we first try to compute the expected cycle structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, as derived from the indicators. If we have enough indicators to fully reproduce the cyclic structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, this will help to make the search faster, by ruling out wrong settings and saving hill-climbing invocations for those settings. But in contrast

with Rejewski's method, if there are not enough indicators to completely compute the cycle structures, we do not give up.

We start by trying to fully reproduce *IndicCycles*, the cyclic structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, based on the indicators. We then exhaustively look for all possible rotor orders, ring settings and (base) rotor settings. In the outer loop, we start by checking with ring settings ZZZ, and we continue by testing all other relevant ring settings. For each option of ring settings, we compute the actual cycle structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, derived from the settings (combined with the candidate rotor order and based rotor settings), and compare them with *IndicCycles*, if available. If they match, or if *IndicCycles* are not available, we invoke the inner hill climbing *HC*, to recover the plugboard connections. The reason we check first with ring settings ZZZ, is that if there was no middle rotor turnover during the encryption of the message rotor settings ($2 \cdot 3 = 6$ letters), the algorithm would succeed early on with ring settings ZZZ. If there was a turnover, then we need to test all relevant ring setting options.

Algorithm 10 Enigma – new attack on double indicators – procedure until 1938

```
1:  procedure FINDDAILYKEY(Indics)                          ▷ 6-letter indicators
2:      IndicCycles ← ComputeCycles(Indics)                 ▷ nil if not enough indicators
3:      for RingSettings ∈ [ZZZ, AAA to AZZ] do             ▷ ZZZ succeeds if no middle rotor move
4:          for RotorOrder ∈ all rotor orders do
5:              for BaseRotorSettings =AAA to ZZZ do
6:                  Cycles ← ComputeCycles(RotorOrder, RingSettings, BaseRotorSettings)
7:                  if IndicCycles = nil or Cycles = IndicCycles then
8:                      Plugboard ← HC(RotorOrder, RingSettings, BaseRotorSettings, Indics)
9:                      if Plugboard ≠ nil then
10:                         return RotorOrder, RingSettings, RotorSettings, Plugboard
```

At the heart of the method, is an inner hill-climbing algorithm *HC*, described in Algorithm 11, that recovers the plugboard connections, given the rotor order, ring settings, and base rotor settings. If those settings are correct, hill climbing will succeed and find the plugboard connections. Otherwise, it will fail, even if the settings match the cyclic structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$ (computed from the indicators). The inner hill-climbing algorithm, has therefore two purposes: the first one is to rule out wrong settings (rotor order, ring settings, and base rotor settings), the second is to find the plugboard connections if the settings are correct. This inner hill climbing is an adaptation of Weierud and Sullivan hill-climbing method, described in Section 3.5.1 and in [21]. Instead of using IC and n-grams, it uses a specialized scoring method, described in Algorithm 12. To compute the score for candidate plugboard connections, we decrypt all indicators, using the base rotor settings, and examine whether they match the expected *xyzxyz* form. We allocate one point for each correct matching, that is if the letter at position $i+3$ is identical to the letter at position i, with $1 \leq i \leq 3$. The *HC* algorithm succeeds if he finds plugboard connections, so that all decryptions are in the form *xyzxyz*.

The *Neighbor* function in *HC* employs transformations applied on pairs of letters in the plugboard, as described in Table 10.1. If both letters a and b are self-steckered and not connected (denoted as $(a)(b)$), we connect them (denoted as (ab)) and then test the resulting new plugboard settings. If they were already connected to each other, we disconnect them and test the new plugboard settings. Special care is needed if either a or b or both were already connected to other letters, in which case we need to check more combinations, as listed in Table 10.1.

Algorithm 11 Enigma – new attack on double indicators – inner hill climbing

```
1:  procedure HC(RotorOrder, RingSettings, BaseRotorSettings, Indics)
2:      R ← [RotorOrder, RingSettings, BaseRotorSettings]
3:      for i = 1 to 5 do
4:          Plugboard ← RandomPlugboard()
5:          repeat
6:              Stuck ← true
7:              for NewPlugboard ∈ Neighbors(Plugboard) do
8:                  NewScore ← Score(R, NewPlugboard, Indics)
9:                  if NewScore = 3 · NumberOf(Indics) then
10:                     return NewPlugboard              ▷ perfect score – return plugboard
11:                 if NewScore > Score(R, Plugboard, Indics) then
12:                     Plugboard ← NewPlugboard         ▷ improved – update
13:                     Stuck ← false
14:                     break
15:         until Stuck
16:     return nil                                       ▷ did not find the correct plugboard
```

Current state of inputs a **and** b	**Transformed states**
(a) (b)	(ab)
(ab)	(a) (b)
(ax) (b)	(ab) (x) (a) (bx)
(ax) (by)	(ab) (x) (y) (ab) (xy) (a) (by) (x) (a) (bx) (y) (a) (b) (xy) (a) (b) (x) (y) (ay) (bx) (ay) (b) (x) (ax) (b) (y)

TABLE 10.1: Enigma double indicators – plugboard transformations for hill climbing

We evaluated the performance of this new attack, designed to overcome the limitations of the original Rejewski's method. We summarize the results in Table 10.2. With 6 plugboard connections, only 6 to 8 indicators are required, and with 10 plugboard connections, between 7 to 9. For comparison, the original algorithm requires between 70 to 90 indicators to succeed. Unlike the original algorithm, it succeeds even if there is a turnover of the middle rotor during the encryption (double) or the message rotor settings. It is able to automatically rule out wrong settings that may nevertheless match the expected cycle structure of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$ (as computed from the indicators), and to recover the plugboard connections if the settings are correct.

The work factor of our new method depends primarily on the number of times the inner hill-climbing process is invoked. If there were not enough indicators to reconstitute the cyclic structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, it needs to be invoked for every rotor order (6 possible orders with 3 types or rotors, or 60, with 5 types), for every possible base rotor settings ($26^3 = 17576$),

Algorithm 12 Enigma – new attack on double indicators – scoring function

```
1: procedure SCORE(RotorOrder, RingSettings, BaseRotorSettings, Plugboard, Indics)
2:     Key ← [RotorOrder, RingSettings, BaseRotorSettings, Plugboard]
3:     Score ← 0
4:     for Indic ∈ Indics do
5:         MessageRotorSettingsTwice ← Decrypt(Key, Indic)
6:         for i = 1 to 3 do
7:             if MessageRotorSettingsTwice[i] = MessageRotorSettingsTwice[i+3] then
8:                 Score ← Score + 1
9:     return Score
```

	Rejewski's method	**New attack**
Number of indicators required	70 to 90	7 to 9
Applicable with turnover of middle rotor	No	Yes
Ruling out wrong settings that match the cycle structures	Manually	By inner hill climbing
Recovery of plugboard settings	Manually	By inner hill climbing

TABLE 10.2: Enigma double indicators – performance of the new attack with 10 plugboard connections

and for all cryptographically unique ring settings. As the ring settings of the leftmost rotor have no effect, we are concerned only with the ring settings of the middle and right rotors, with a total of $26^2 = 676$ unique settings. With a system with 3 types of rotors, the number of times the inner hill climbing is invoked is 71 288 256, and with 5 types of rotors, 712 882 560.

Those are only worst-case numbers, however. If there was no turn-over of the middle rotor during the encryption of the message settings (which are all encrypted twice using the same base rotor settings), then only one ring settings option is tested (`ZZZ`), and the workload is reduced to 105 456 (3 types of rotors) or 1 054 560 (5 types of rotors). Furthermore, if there were enough indicators to reproduce the cycle structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$, in most cases the inner hill climbing needs to be invoked only for a very small number of settings, and very often, only a single one (as for Rejewski's original method, see Section 10.4). On an Intel Core i7 6950x 3.0Ghz PC with 10 cores and multithreading, this attack takes a few seconds if there were enough indicators to reconstitute the cyclic structures of $A_4 \cdot A_1$, $A_5 \cdot A_2$, and $A_6 \cdot A_3$. Otherwise, it will take less than a minute if there was no turnover of the middle rotor, and up to an hour if the middle rotor steps during the encryption of the indicators.

10.7.2 New Attack on Double Indicators – Procedure from 1938 to 1940

We developed a new method for the cryptanalysis of Enigma (double) indicators based on the German procedure in use between September 1938 and May 1940. Our new attack was designed under the guidelines of the new methodology from Chapter 4, employing a divide-and-conquer approach, hill climbing, and a specialized scoring method.

Our algorithm is described in Algorithm 13. The input is a set of indicators with 9 letters, each of them starting with initial rotor settings (in clear), followed by an encryption of the message rotor settings (typed twice), encrypted with the unknown daily key (rotor order, ring settings, plugboard connections) and the initial rotor settings. Similarly to Zygalski's original method,

we first look for females in the indicators. If we have females, this will help make the search faster, by ruling out wrong settings and saving invocations of the inner hill-climbing process. But in contrast with Zygalski's method, females are not mandatory for the attack to succeed.

We exhaustively look at all possible combinations of the rotor order and the ring settings. For each combination, we check whether they allow the set of females we have previously found (see Section 10.6), and if not, we rule out that particular combination or the rotor order and the ring settings. If we cannot rule out the settings, or if there were no females in the indicators, we invoke the inner hill climbing *HC*, to recover the plugboard connections, or rule out the settings.

Algorithm 13 Enigma – new attack on double indicators – procedure from 1938 to 1940

```
1: procedure FINDDAILYKEY1938(Indics)                          ▷ 9-letter indicators
2:     Females ← FemalesFromIndicators(Indics)
3:     for RotorOrder ∈ all rotor orders do
4:         for RingSettings = AAA to ZZZ do                    ▷ full range
5:             Restrictions ← FemalesRestrictions(RotorOrder, RingSettings)
6:             if Females match Restrictions or Females is empty then
7:                 Plugboard ← HC(RotorOrder, RingSettings, RotorSettings, Indics)
8:                 if Plugboard ≠ nil then
9:                     return RotorOrder, RingSettings, Plugboard
```

The inner hill-climbing process *HC*, is described in Algorithm 14. It tries to recover the plugboard connections, for a candidate rotor order and ring settings. If those settings are correct, hill climbing will succeed and find the plugboard connections. Otherwise, it will fail, even if the settings allow for the females in the indicator set. This inner hill climbing is also similar to Weierud and Sullivan hill-climbing method, described in Section 3.5.1 and in [21]. It uses a specialized scoring method, described in Algorithm 15. To compute the score for candidate plugboard connections, we decrypt the last 6 letters of each indicator, using the initial rotor settings given in clear (first 3 letters), and examine whether the decryption matches the expected *xyzxyz* form. We allocate one point for each correct matching, that is if the letter at position $i+3$ is identical to the letter at position i, with $1 \leq i \leq 3$. The *HC* algorithm succeeds if it finds plugboard connections, so that all decryptions are of the form *xyzxyz*. The transformations employed are the same as for our previous attack, as listed in Table 10.1.

	Zygalski's method	**New attack**
Number of indicators required	55 to 137 to obtain 11 females	8 to 12, with or without females
Ruling out wrong settings that match female restrictions	Manually	By inner hill climbing
Recovery of plugboard settings	Manually	By inner hill climbing

TABLE 10.3: Enigma double indicators (1938-1940) – performance of the new attack

We analyzed the performance of our new attack, and compare it to Zygalski's original method, as described in Table 10.3. It needs a much smaller number of intercepted indicators, regardless of whether there are females or not in the set (although females help in speeding up the search). It also recovers the correct plugboard settings, or alternatively, it is able to automatically rule out wrong settings.

The work factor of our new method depends primarily on the number of times the inner hill-climbing process is invoked. If there are no indicators with females, it needs to be invoked for

Algorithm 14 Enigma – new attack on double indicators (1938-1940) – inner hill climbing

```
1:  procedure HC(RotorOrder, RingSettings, Indics)
2:      R ← [RotorOrder, RingSettings]
3:      for i = 1 to 5 do
4:          Plugboard ← RandomPlugboard()
5:          repeat
6:              Stuck ← true
7:              for NewPlugboard ∈ Neighbors(Plugboard) do
8:                  NewScore ← Score(R, NewPlugboard, Indics)
9:                  if NewScore = 3 · NumberOf(Indics) then
10:                     return NewPlugboard                ▷ perfect score – return plugboard
11:                 if NewScore > Score(R, Plugboard, Indics) then      ▷ improved – update
12:                     Plugboard ← NewPlugboard
13:                     Stuck ← false
14:                     break
15:         until Stuck
16:     return nil                                         ▷ did not find the correct plugboard
```

Algorithm 15 Enigma – new attack on double indicators (1938-1940) – scoring function

```
1:  procedure SCORE(RotorOrder, RingSettings, PlugbConnections, Indics)
2:      Score ← 0
3:      for Indic ∈ Indics do
4:          InitialRotorSettings ← Indic[1 : 3]
5:          Key ← [RotorOrder, RingSettings, InitialRotorSettings, PlugbConnections]
6:          MessageRotorSettingsTwice ← Decrypt(Key, Indic[4 : 9])
7:          for i ∈ [1, 2, 3] do
8:              if MessageRotorSettingsTwice[i] = MessageRotorSettingsTwice[i + 3] then
9:                  Score ← Score + 1
10:     return Score
```

every rotor order (6 or 60 orders, for 3 types of rotors, or for 5 types, respectively) and for all possible ring settings ($26^3 = 17\,576$), with a total of 105 456 (3 types of rotors) or 1 054 560 (5 types of rotors). With each indicator with females, this number is reduced by a factor of 0.41. For example, with 5 females and 3 rotor types, the inner hill-climbing process is invoked only $105\,456 \cdot 0.41^5 = 1\,222$ times. Without any females, this attack takes a few minutes on an Intel Core i7 6950x 3.0Ghz PC with 10 cores and multithreading. With 5 females, it requires a few seconds.

10.8 The Enigma Contest – 2015

With our new methods described in this chapter, we won the International Codebreakers.eu Contest in 2015, organized by the City of Poznan, in Poland, in memory of the Polish mathematicians, Rejewski, Zygalski, and Jerzy Różycki, who had studied at the local university. This Enigma contest was held in five stages, the last four including series of indicators produced according to the two procedures described in this chapter.

10.9 Summary

The two attacks we described in the chapter as based on our new methodology, with divide-and-conquer, inner hill climbing, and highly specialized scoring methods, as summarized in Table 10.4.

Principle	**Application of the methodology principle**
GP1	Nested search – outer semi-brute-force loop on ring settings (both attacks) and rotor settings (attack for indicators before 1938), inner HC search for plugboard connections.
GP2	Search only for cryptographically unique ring/rotor settings Prune non-matching settings (given enough indicators)
GP3	Specialized scores based on the analysis of the indicators, with high resilience to errors
GP4	Simple non-disruptive swap transformations
GP5	Multiple restarts

TABLE 10.4: Enigma – applying the methodology – attacks on indicators

Our new attacks take advantage of weaknesses identified by the Polish Cipher Bureau in the 1930s, based on which Rejewski, Zygalski, and their team developed ingenious partially mechanized cryptanalytic attacks, using the technology available at the time. In contrast, our new attacks are computerized, and while they require modest computing power, they could not have been implemented using the technology available in the 1930s or 1940s. Our methods extend the scope of the original attacks, overcoming their limitations, allowing for a more general, robust, and fully automated solution, and further emphasizing the vulnerabilities introduced by the use of the "double indicators".

11

Conclusion

Deciphering is both a science and an art. It is a science because certain definite laws and principles have been established which pertain to it; it is also an art because of the large part played in it by imagination, skill, and experience. Yet it may be said that in no other science are the rules and principles so little followed and so often broken; and in no other art is the part played by reasoning and logic so great. In no other science, not even excepting the science of language itself, grammar, does that statement, "The exception proves the rule," apply so aptly. Indeed it may be said, and still be within the limits of the truth, that in deciphering, "The rule is the exception."

William F. Friedman [6]

This thesis and research are about the intersection of history, cryptography, and computer science. Modern work on the cryptanalysis of classical ciphers contributes to a deeper understanding of the history of the development of ciphers, and of codebreaking techniques. In some cases (including the ADFGVX case study described in Chapter 6), such work led to the decryption of historical messages which could not be read otherwise. The main tool employed so far has been local search metaheuristics, but with little understanding of what makes a specific application of local search effective. The primary purposes of this research are to help to bridge that gap, and to develop effective cryptanalytic attacks for the more challenging cases of historical cipher systems and problems. To achieve those goals, we developed a new methodology for the effective cryptanalysis of classical ciphers using local search metaheuristics, presented in Chapter 4, with the following five main guiding principles:

GP1: Hill climbing or simulated annealing

GP2: Reduction of the search space

GP3: Adaptive scoring

GP4: High-coverage transformations preserving a smooth search landscape

GP5: Multiple restarts with optimal initial keys

Table 11.1 summarizes the application of the principles in case studies. All attacks are ciphertext-only unless stated otherwise. "++" means that the cryptanalytic method developed for the case study fully or significantly applies the guiding principle. "+" means that the principle has been partially applied.

For completeness, prior case studies by other authors, which are state of the art for the specific cipher problem, in terms of performance, are also included. Although they were developed independently of (and before) our new methodology, their implementation is in line, to a significant extent, with its five principles.

Almost all of the case studies covered in Table 11.1 apply all five principles. In some of the case studies, GP5 is only loosely implemented ("+"), using multiple restarts but with simple random initial keys. This is possible because the application of the other principles is powerful enough, so that optimized initial keys are not mandatory. In the Hagelin M-209 known-plaintext attack case study, the scoring function (ADE) is so effective that a divide-and-conquer approach (GP2) is not necessary (the "+" in GP2 reflects here only the pruning of redundant lug settings). Divide-and-conquer approaches (GP2) were not applicable to the (single) columnar transposition cipher, as well as to Playfair, both of which having a single encryption stage.

Case study	Reference	GP1	GP2	G3	GP4	GP5
Own case studies						
Columnar transposition	Chapter 5	++		++	++	++
ADFGVX	Chapter 6	++	++	++	++	++
Hagelin M-209 – known-plaintext	Section 7.4	++	+	++	++	+
Hagelin M-209 – ciphertext-only	Section 7.5	++	++	++	++	+
Chaocipher	Chapter 8	++	++	++	++	+
Double transposition	Chapter 9	++	++	++	++	++
Enigma – double indicators	Chapter 10	++	++	++	++	+
Prior work						
Enigma – ciphertext-only (Weierud and al.)	[21][51]	++	++	++	++	++
Purple (Freeman and al.)	[22]	++	++	++	+	+
Playfair (Cowan)	[24]	++		+	++	+

TABLE 11.1: Conclusion – summary of case studies

In contrast, prior work case studies from other authors (listed in Section 3.5.4) often employed local search metaheuristics other than the ones recommended in GP1, such as genetic algorithms. When hill climbing and simulated annealing were employed, their application was simplistic, with a single process (partial implementation of GP1), simple restarts (GP5, partially), and only one work applied GP4 (segment-based transformations [16]).

Table 11.2 summarizes the performance of the new attacks in case studies described in detail in this thesis. To date, all the attacks are state of the art, in terms of performance, for the specific cryptanalytic problem. With those new attacks, we were able to decipher for the first time a collection of original WWI ADFGVX messages, to solve several cryptographic challenges, never solved before, and to win an international contest on Enigma. To achieve those results, an extensive process of running simulations was conducted for each attack, to experiment with and validate new types of scoring functions, looking for the right balance between selectivity and resilience to key errors. A similar time-consuming process was also required to select the most effective sets of transformations, with an optimal balance between a wide neighborhood and a smooth search landscape, and to test various local search schemes and divide-and-conquer

approaches. The attacks covered in this thesis are integrated, or in the process of being integrated into CrypTool 2, a leading open-source collaborative tool for learning, teaching, and experimenting with cryptographic and cryptanalytic methods (see Appendix A).

Case study	Performance	Comments
Own case studies		
Columnar transposition	State-of-the-art performance Key length up to 1 000 (120 for worst case settings)	
ADFGVX	State-of-the-art performance Transposition keys with length up to 23	First complete decipherment of 600 original cryptograms from WWI (1918)
Hagelin M-209 known-plaintext attack	State-of-the-art performance 50 known-plaintext letters	Solved Morris' challenge (1978)
Hagelin M-209 ciphertext-only attack	State-of-the-art performance 500 letters	Solved the last challenge and won the Hagelin M-209 Contest (2012)
Chaocipher	State-of-the-art performance 80 messages in depth	Solved Chaocipher Exhibit 6 (2010)
Double transposition	State-of-the-art performance Keys with 20 elements (worst-case settings)	Solved the Double Transposition Challenge (2007)
Enigma – double indicators	State-of-the-art performance Less than 10 indicators	Won an international Enigma challenge (2015)
Prior works		
Enigma – ciphertext-only (Weierud and al.)	State-of-the-art performance	Decipherment of hundreds of WWII original messages
Purple (Freeman and al.)	State-of-the-art performance	Recovered key for historical diplomatic message
Playfair (Cowan)	State-of-the-art performance	

TABLE 11.2: Conclusion – performance summary

An obvious area for future research consists of applying the principles of the methodology, to local search metaheuristics other than hill climbing and simulated annealing. Other local search metaheuristics such as genetic algorithms or ant colony optimization, may benefit from divide-and-conquer approaches (GP2), from more effective scoring methods (GP3), and from better transformations (GP4). In addition, the domain of local search is constantly evolving, and new methods applied in other problem domains, may be effective for cryptanalysis as well. For example, the fixed-temperature variant of simulated annealing, used for Playfair, warrants further investigation for other ciphers.

While the focus of this thesis was the application of local search metaheuristics for the cryptanalysis of classical ciphers, the evolving field of machine learning may be highly relevant as well (e.g. Hidden Markov Models for homophonic ciphers [146]). Machine learning models, and deep learning (neural networks) models in particular, can be trained using a multitude of key-plaintext-ciphertext samples which can be easily generated. Those models might be used for scoring putative keys, or to predict the value of the elements of the key.

The case studies in this thesis and in the prior work presented here represent only a subset of the classical cipher methods and cipher systems. Work has started to implement attacks, based on the methodology, for other types of classical ciphers, such as the Lorenz SZ42 ("Tunny"), the Siemens and Halske T52 ("Sturgeon"), homophonic ciphers, and cylinder ciphers (M-94, M-138). Preliminary results are promising, but many other ciphers have not yet been investigated in the context of attacks based on local search. More details are being declassified about sophisticated Cold War cipher systems. One example is the Russian Fialka, for which there is no cryptanalytic method in the public domain [28]. Those systems are obvious candidates for the methods described in this thesis.

One important area which has not been covered in this research, is the question of how to determine – analytically – which of the principles and guidelines are best suited for a particular type of cipher. Some aspects of this question are trivial, such as the need to reduce the search space (GP2), if the cipher has a very large keyspace. Other aspects are less trivial, such as the choice of a hill climbing or simulated annealing scheme (GP1) for a particular cipher. For that purpose, tools developed for the analysis of the search landscape may be applied. In Section 3.2.3, we introduced the concepts of selective scoring functions, and their resilience to key errors, but we did that on an intuitive and less quantitative level. Some more quantifiable measures might be useful in further formalizing the process of selecting and developing effective scoring functions. Our belief, however, is that even with such measures, the process of developing effective cryptanalytic attacks based on local search metaheuristics will persist being both a science and an art.

A

CrypTool 2

CrypTool 2 (CT2) [https://www.cryptool.org/en/cryptool2] is an open-source e-learning tool that helps pupils, students, and crypto enthusiasts to learn cryptology. CT2 is part of the CrypTool project [https://en.wikipedia.org/wiki/CrypTool] which includes widely-used e-learning tools for cryptography and cryptanalysis. CT2 is the successor of CrypTool 1 (CT1), and it is based on a graphical programming language allowing the user to cascade different ciphers and methods and to follow the encryption or decryption steps in real-time.

CT2 is maintained by the research group Applied Information Security (AIS) of the university of Kassel. Contributions and voluntary support to this open-source project come from all over the world. CT2 implements classical and modern cryptographic methods, including cryptanalytic methods. It is also used to implement real-world prototypes of distributed cryptanalysis using the so-called “CrypCloud”. CT2 is maintained in English and German.

Figure A.1 is a screenshot of a typical workflow in CrypTool 2 showing the cryptanalysis of a German ciphertext which was encrypted with Vigenère using a 16-letter key:

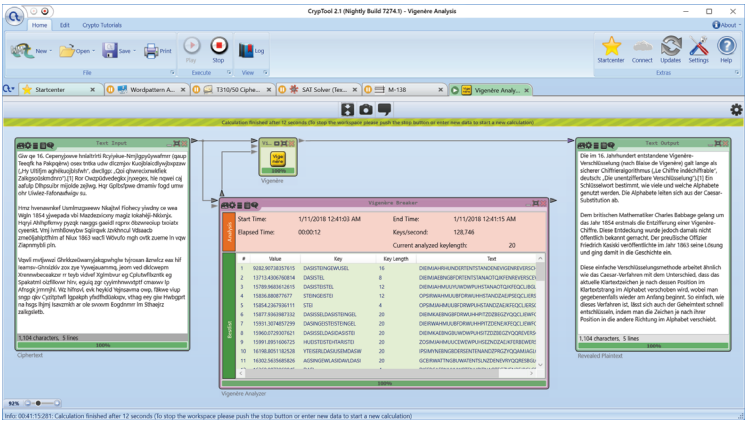


FIGURE A.1: CT2 – cryptanalysis of the Vigenère cipher

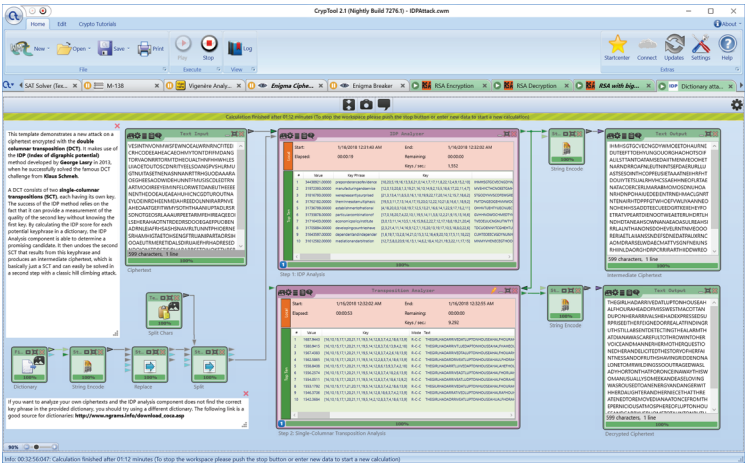


FIGURE A.2: CT2 – cryptanalysis of the double transposition cipher

The attack algorithms against the double transposition cipher (see Chapter 9) are already integrated in CT 2.1, as shown in Figure A.2. Additional cryptanalysis algorithms developed as part of this thesis will be integrated in CT 2.1 during 2018.

Bibliography

[1] David Kahn. *The Codebreakers*. Weidenfeld and Nicolson, 1974.

[2] Craig Bauer. *Secret History: The Story of Cryptology*. CRC Press, 2013.

[3] William F. Friedman. *The Index of Coincidence and its Applications in Cryptanalysis*. Aegean Park Press, 1987.

[4] Cipher A. Deavours and Louis Kruh. *Machine Cryptography and modern cryptanalysis*. Artech House, Inc., 1985.

[5] William F. Friedman. *Military Cryptanalysis*. US Government Printing Office, 1941.

[6] William F. Friedman. *An Introduction to Methods for the Solution of Ciphers*. Riverbank Laboratories, Department of Ciphers, Geneva, IL, 1918.

[7] William F. Friedman. *Elements of Cryptanalysis (Cryptographic Series)*. Aegean Park Press, Laguna Hills, CA, 1976.

[8] Luigi Sacco. *Manual of Cryptography (Cryptographic Series)*. Aegean Park Press, Laguna Hills, CA, 1996.

[9] Stephen Budiansky. *Battle of Wits: The Complete Story of Codebreaking in World War II*. Simon and Schuster, 2000.

[10] Jack B. Copeland. *Colossus: The secrets of Bletchley Park's Codebreaking Computers*. Oxford University Press, 2006.

[11] Colin B. Burke. *It Wasn't All Magic: The Early Struggle to Automate Cryptanalysis, 1930s-1960s*. Fort Meade: Center for Cryptologic History, National Security Agency, 2002.

[12] Auguste Kerckhoffs. La Cryptographie Militaire. *Journal des sciences militaires*, 9:538, 1883.

[13] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell system technical journal*, 28(4):656–715, 1949.

[14] A. Dimovski and D. Gligoroski. Attacks on the Transposition Ciphers Using Optimization Heuristics. *Proceedings of ICEST*, pages 1–4, 2003.

[15] A. Clark. Modern optimisation algorithms for cryptanalysis. *In Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems*, pages 258–262, 1994.

[16] Andrew J. Clark. Optimisation Heuristics for Cryptology. *PhD thesis*, 1998.

[17] M.D. Russell, John A. Clark, and S. Stepney. Making the most of two heuristics: breaking transposition ciphers with ants. *The 2003 Congress on Evolutionary Computation*, 4: 2653–2658, 2003.

[18] Sarab M. Hameed and Dalal N. Hmood. Particles Swarm Optimization for the Cryptanalysis of Transposition Cipher. *Journal of Al-Nahrain University*, 13(4):211–215, 2010.

[19] Jian Chen and Jeffrey S. Rosenthal. Decrypting Classical Cipher Text using Markov Chain Monte Carlo. *Statistics and Computing*, 22(2):397–413, 2012.

[20] James J. Gillogly. Ciphertext-Only Cryptanalysis of Enigma. *Cryptologia*, 19(4):405–413, 1995.

[21] Geoff Sullivan and Frode Weierud. Breaking German Army Ciphers. *Cryptologia*, 29(3): 193–232, 2005.

[22] Wes Freeman, Geoff Sullivan, and Frode Weierud. Purple Revealed: Simulation and Computer-Aided Cryptanalysis of Angooki Taipu B. *Cryptologia*, 27(1):1–43, 2003.

[23] Kelly Chang, Richard M. Low, and Mark Stamp. Cryptanalysis of Typex. *Cryptologia*, 38(2):116–132, 2014.

[24] Michael J. Cowan. Breaking Short Playfair Ciphers with the Simulated Annealing Algorithm. *Cryptologia*, 32(1):71–83, 2008.

[25] Geoff Sullivan and Frode Weierud. The Swiss NEMA Cipher Machine. *Cryptologia*, 23 (4):310–328, 1999. doi: 10.1080/0161-119991887973.

[26] Mark Stamp and Wing On Chan. SIGABA: Cryptanalysis of the Full Keyspace. *Cryptologia*, 31(3):201–222, 2007. doi: 10.1080/01611190701394650.

[27] Cipher Deavours. Helmich and the KL-7. *Cryptologia*, 6(3):283–284, 1982. doi: 10.1080/0161-118291857091.

[28] Eugen Antal and Pavol Zajac. Key Space and Period of Fialka M-125 Cipher Machine. *Cryptologia*, 39(2):126–144, 2015. doi: 10.1080/01611194.2014.915264.

[29] Henry Beker and Fred Piper. *Cipher Systems: The Protection of Communications*. Northwood Books London, 1982.

[30] H. Paul Greenough. Cryptanalysis of the Hagelin C-52 and Similar Machines – a Known-Plaintext Attack. *Cryptologia*, 23(2):139–156, 1999. doi: 10.1080/0161-119991887801.

[31] Wayne G. Barker. *Cryptanalysis of the Double Transposition Cipher: Includes Problems and Computer Programs*. Aegean Park Press, 1995.

[32] George Lasry, Nils Kopal, and Arno Wacker. Solving the Double Transposition Challenge with a Divide-and-Conquer Approach. *Cryptologia*, 38(3):197–214, 2014. doi: 10.1080/01611194.2014.915269.

[33] George Lasry, Nils Kopal, and Arno Wacker. Cryptanalysis of Columnar Transposition Cipher with Long Keys. *Cryptologia*, 40(4):374–398, 2016. doi: 10.1080/01611194.2015.1087074.

[34] George Lasry, Ingo Niebel, Nils Kopal, and Arno Wacker. Deciphering ADFGVX Messages from the Eastern Front of World War I. *Cryptologia*, 41(2):101–136, 2017. doi: 10.1080/01611194.2016.1169461.

[35] George Lasry, Nils Kopal, and Arno Wacker. Automated Known-Plaintext Cryptanalysis of Short Hagelin M-209 Messages. *Cryptologia*, 40(1):49–69, 2016. doi: 10.1080/01611194.2014.988370.

[36] George Lasry, Nils Kopal, and Arno Wacker. Ciphertext-Only Cryptanalysis of Hagelin M-209 Pins and Lugs. *Cryptologia*, 40(2):141–176, 2016. doi: 10.1080/01611194.2015.1028683.

[37] George Lasry, Moshe Rubin, Nils Kopal, and Arno Wacker. Cryptanalysis of Chaocipher and Solution of Exhibit 6. *Cryptologia*, 40(6):487–514, 2016. doi: 10.1080/01611194.2015.1091797.

[38] Klaus Schmeh. *Nicht zu Knacken*. Carl Hanser Verlag GmbH & Co. KG, 2012.

[39] Holger H. Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[40] Adi Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. *IEEE transactions on information theory*, 30(5):699–704, 1984.

[41] Phong Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from crypto'97. In *Annual International Cryptology Conference*, pages 288–304. Springer, 1999.

[42] Melven R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967.

[43] Frank Hutter, Dave AD Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *International Conference on Principles and Practice of Constraint Programming*, pages 233–248. Springer, 2002.

[44] Nenad Mladenović and Pierre Hansen. Variable Neighborhood Search. *Computers & operations research*, 24(11):1097–1100, 1997.

[45] Edward Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological cybernetics*, 63(5):325–336, 1990.

[46] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, 1995.

[47] Donald W. Davies. The bombe a remarkable logic machine. *Cryptologia*, 23(2):108–138, 1999.

[48] Abraham Sinkov. Elementary cryptanalysis: A mathematical approach, mathematical association of america, 1966. *Additional Reading*, 1966.

[49] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[50] C.A. Deavours. Unicity Points in Cryptanalysis. *Cryptologia*, 1(1):46–68, 1977. doi: 10.1080/0161-117791832797.

[51] Olaf Ostwald and Frode Weierud. Modern breaking of Enigma ciphertexts. *Cryptologia*, published online:1–27, 2017. doi: 10.1080/01611194.2016.1238423.

[52] James Reeds. Entropy Calculations and Particular Methods of Cryptanalysis. *Cryptologia*, 1(3):235–254, 1977. doi: 10.1080/0161-117791832977.

[53] Marian Rejewski. Mathematical solution of the enigma cipher. *Cryptologia*, 6(1):1–18, 1982.

[54] Stephan Krah. The M4 Project, 2013. http://distributedcomputinginfo.pbworks.com/w/page/17922396/M4, [Accessed: April, 15th, 2017].

[55] Edwin Olson. Robust Dictionary Attack of Short Simple Substitution Ciphers. *Cryptologia*, 31(4):332–342, 2007. doi: 10.1080/01611190701272369.

[56] Lars R. Knudsen and Willi Meier. Cryptanalysis of an identification scheme based on the permuted perceptron problem. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 363–374. Springer, 1999.

[57] David Pointcheval. A new identification scheme based on the perceptrons problem. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 319–328. Springer, 1995.

[58] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 363–397. Springer, 2010.

[59] Thomas A. Feo and Mauricio G.C. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[60] J. Pirie Hart and Andrew W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3):107–114, 1987.

[61] Thomas G. Mahon and James Gillogly. *Decoding the IRA*. Mercier Press Ltd, 2008.

[62] Robert A.J. Matthews. The use of genetic algorithms in cryptanalysis. *Cryptologia*, 17 (2):187–201, 1993.

[63] J. P. Giddy and Reihaneh Safavi-Naini. Automated cryptanalysis of transposition ciphers. *The Computer Journal*, 37(5):429–436, 1994.

[64] Helen F. Gaines. *Cryptanalysis: A Study of Ciphers and their Solutions*, volume 97. DoverPublications.com, 1956.

[65] Friedrich L. Bauer. *Decrypted Secrets: Methods and Maxims of Cryptology*. Springer, 2007.

[66] Luis Alberto Benthin Sanguino, Gregor Leander, Christof Paar, Bernhard Esslinger, and Ingo Niebel. Analyzing the Spanish strip cipher by combining combinatorial and statistical methods. *Cryptologia*, 40(3):261–284, 2016.

[67] Sophie De Lastours. *La France gagne la guerre des codes secrets: 1914–1918*. Tallandier, 1998.

[68] James R. Childs. *General Solution of the ADFGVX Cipher System*. Aegean Park Press, Laguna Hills, CA, 2000.

[69] James R. Childs. Geschichte und Grundregeln deutscher militärischer Geheimschriften im Ersten Weltkrieg, 1969.

[70] James R. Childs. *The History and Principles of German Military Ciphers, 1914-1918*. Unpublished manuscript, copy available at the National Cryptologic Museum, MD, 1919.

[71] James R. Childs. *German Military Ciphers from February to November 1918*. War Department, Office of the Chief Signal Officer, U.S. Government Printing Office, Washington, DC, 1935. https://www.nsa.gov/public_info/_files/friedmanDocuments/Publications/FOLDER_268/41784789082381.pdf, [Accessed: Feb, 5th, 2016].

[72] John Ferris. The British army and signals intelligence in the field during the first World War. *Intelligence and National Security*, 3(4):23–48, 1988.

[73] Markus Pöhlmann. German Intelligence at War, 1914–1918. *Journal of Intelligence History*, 5(2):25–54, 2005.

[74] William F. Friedman. *Military Cryptanalysis*. Cryptographic Series. Aegean Park Press, Laguna Hills, CA, 1996.

[75] William F. Friedman. *General Solution of the ADFGVX Cipher System. Technical Paper of the Signal Intelligence Section*. War Plans and Training Division. US Printing Office, Washington, DC, 1934. https://www.nsa.gov/public_info/_files/friedmanDocuments/Publications/FOLDER_269/41784769082379.pdf, [Accessed: Feb, 5th, 2016].

[76] Marcel Givierge. *Cours de Cryptographie*. Berger–Levrault, Paris, 1932.

[77] Alan G. Konheim. Cryptanalysis of ADFGVX encipherment systems. In *Advances in Cryptology*, pages 339–341. Springer, 1985.

[78] J. Yi. Cryptanalysis of a Homophonic Substitution-Transposition cipher. *Masters Report, Department of Computer Science, San Jose State University*, 2014.

[79] Thomas Jakobsen. A fast method for cryptanalysis of substitution ciphers. *Cryptologia*, 19(3):265–274, 1995.

[80] William S Forsyth and Reihaneh Safavi-Naini. Automated cryptanalysis of substitution ciphers. *Cryptologia*, 17(4):407–418, 1993.

[81] RS Ramesh, G Athithan, and K Thiruvengadam. An automated approach to solve simple substitution ciphers. *Cryptologia*, 17(2):202–218, 1993.

[82] Andrew Clark and Ed Dawson. A parallel genetic algorithm for cryptanalysis of the polyalphabetic substitution cipher. *Cryptologia*, 21(2):129–138, 1997.

[83] John C. King. An algorithm for the complete automated cryptanalysis of periodic polyalphabetic substitution ciphers. *Cryptologia*, 18(4):332–355, 1994.

[84] Michael Lucks. A constraint satisfaction algorithm for the automated decryption of simple substitution ciphers. In *Advances in Cryptology–CRYPTO'88*, pages 132–144. Springer, 1988.

[85] Ray Smith. An overview of the Tesseract OCR engine. In *12th International Conference on Document Analysis and Recognition*, pages 629–633. IEEE, 2007.

[86] Hermann Stützel. Geheimschrift und Entzifferung im Ersten Weltkrieg. *Truppenpraxis*, 7:541–545, 1969.

[87] Gerald Loftus. J. Rives Childs in Wartime Tangier, 2014. http://www.afsa.org/PublicationsResources/ForeignServiceJournal/FeaturedContent/JanFeb2014JRivesChildsinWartimeTangier.aspx, [Accessed: Feb, 5th, 2016].

[88] Theo Schwarzmüller. Generalfeldmarschall August von Mackensen. *Zwischen Kaiser und "Führer". Paderborn/München/Wien/Zürich*, 1996.

[89] James R. Childs. My recollections of G.2 A.6. *CRYPTOLOGIA*, 2(3):201–214, 1978.

[90] Boris Hagelin and David Kahn. The Story of the Hagelin Cryptos. *Cryptologia*, 18(3): 204–242, 1994.

[91] Crypto Museum. Crypto Museum website, Crypto AG Hagelin cipher machines, 2014. http://www.cryptomuseum.com/crypto/hagelin/index.htm, [Accessed: December, 7th, 2014].

[92] Luigi Donini and Augusto Buonafalce. The Cryptographic Services of the Royal (British) and Italian Navies. *Cryptologia*, 14(2):97–127, 1990.

[93] Jerry Proc. Crypto Machines website, M209(CSP-1500), 2014. http://http://jproc.ca/crypto/m209.html, [Accessed: December, 7th, 2014].

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

[95] War Department. TM-11-380, Technical Manual, Converter M-209, 1942. http://maritime.org/tech/csp1500inst.htm, [Accessed: September, 28th, 2014].

[96] War Department. TM-11-380 B, Technical Manual, Converter M-209B, 1943.

[97] 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].

[98] War Department. TM-11-380, Technical Manual, ConverterM-209, M-209A, M-209 B (Cipher) , 1947.

[99] 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.

[100] Kenneth H. Rosen. Discrete mathematics and its applications. *AMC*, 10:12, 2007.

[101] 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].

[102] 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].

[103] 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].

[104] Robert Morris. The Hagelin Cipher Machine (M-209) Reconstruction of the Internal Settings. *Cryptologia*, 2(3):267–289, 1978.

[105] Ronald L. Rivest. Statistical Analysis of the Hagelin Cryptograph. *Cryptologia*, 5(1): 27–32, 1981.

[106] Geoff Sullivan. Cryptanalysis of Hagelin Machine Pin Wheels. *Cryptologia*, 26(4):257–273, 2002.

[107] Dennis Ritchie. Dabbling in the Cryptographic World - A Story, 2000. http://cm.bell-labs.com/who/dmr/crypt.html, [Accessed: September, 28th, 2014].

[108] Katie Hafner and John Markoff. *Cyberpunk, Outlaws and Hackers on The Computer Frontier*. Simon & Schuster, New York, NY, 1991.

[109] 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.

[110] Jim Reeds. Solved: The Ciphers in Book III of Trithemius Steganographia. *Cryptologia*, 22(4):291–317, 1998.

[111] Jean-François Bouchaudy. The M-209 Challenge, 2014. http://www.jfbouch.fr/crypto/challenge, [Accessed: May, 2nd, 2017].

[112] Jean-François Bouchaudy. George Lasry won the M-209 Challenge on January 14, 2017, 2014. http://www.jfbouch.fr/crypto/challenge/resultats.html, [Accessed: May, 2nd, 2017].

[113] Byrne, John F. *Silent Years: An Autobiography with Memoirs of James Joyce and Our Ireland*. Farrar, Straus and Young, 1953.

[114] The Chaocipher Clearing House. Silent Years, Chapter 21: Chaocipher, 2015. http://www.chaocipher.com/Silent-Years-Chapter-21-Chaocipher.pdf, [Accessed: March, 28th, 2015].

[115] Knight, Gary H. Cryptanalyst's Corner. *Cryptologia*, 2(1):68–74, 1978.

[116] Byrne, John, Deavours, Cipher A. and Kruh, Louis. Chaocipher Enters the Computer Age when its Method is Disclosed to Cryptologia Editors. *Cryptologia*, 14(3):193–198, 1990.

[117] Rubin, Moshe. Chaocipher Revealed: The Algorithm, 2010. http://www.chaocipher.com/ActualChaocipher/Chaocipher-Revealed-Algorithm.pdf, [Accessed: March, 24th, 2015].

[118] Scheffler, Carl. Chaocipher: Cracking Exhibit 1, 2015. http://www.inference.phy.cam.ac.uk/cs482/projects/chaocipher/exhibit1.html, [Accessed: March, 27th, 2015].

[119] The Crypto Forum. Thread entitled "Starting Alphabet Found", 2015. http://s13.zetaboards.com/Crypto/single/?p=8002535&t=6715252, [Accessed: March, 28th, 2015].

[120] Peuha, Esa. Decoding Chaocipher Exhibits 2 & 3, 2015. http://www.chaocipher.com/chaocipher-022.htm, [Accessed: March, 29th, 2015].

[121] The Chaocipher Clearing House. The Chaocipher Clearing House web site, 2015. http://www.chaocipher.com/, [Accessed: March, 28th, 2015].

[122] Calof, Jeff, Hill, Jeff and Rubin, Moshe. Chaocipher Exhibit 5: History, Analysis, and Solution of Cryptologia's 1990 Challenge. *Cryptologia*, 38(1):1–25, 2014.

[123] Calof, Jeff and Rubin, Moshe. Chaocipher: Exhibit 6, 2015. http://www.chaocipher.com/ActualChaocipher/calof-exhibit-6/Chaocipher-Exhibit%206%20article.Calof-Rubin.pdf, [Accessed: March, 27th, 2015].

[124] The Crypto Forum. Attempting to solve Exhibit 6 using hill-climbing/simulated annealing, 2015. http://s13.zetaboards.com/Crypto/topic/7330974, [Accessed: March, 27th, 2015].

[125] Byrne, John Francis. Blueprints for Chaocipher, Unknown Year. https://www.nsa.gov/about/_files/cryptologic_heritage/museum/library/chaocipher_blueprints.pdf, [Accessed: August, 4th, 2015].

[126] Byrne, John Francis. Untitled manuscript describing the encipherment of MacArthur speech with Chaocipher, Unknown Year. https://www.nsa.gov/about/_files/cryptologic_heritage/museum/library/macarthur_speech.pdf, [Accessed: April, 4th, 2015].

[127] Friedman, William F. *Military Cryptanalysis, Part III, Simpler Varieties of Aperiodic Substitution Systems*. Aegean Park Press, 1993.

[128] The Chaocipher Clearing House. The Chaocipher Challenge: Further Work in Progress, 2015. http://www.chaocipher.com/chaocipher-001.htm, [Accessed: March, 29th, 2015].

[129] Rubin, Moshe. John F. Byrne's Chaocipher Revealed: An Historical and Technical Appraisal. *Cryptologia*, 35(4):328–379, 2011.

[130] Babbage, Charles. *Passages from the Life of a Philosopher*. Cambridge University Press, 2011.

[131] Solomon Kullback. *General Solution for the Double Transposition Cipher*, volume 84. Aegean Park Pr, 1934.

[132] Tim Wambach. Kryptanalyse der doppelten Spaltentranspositionschiffre, 2011.

[133] Otto Leiberich. Vom diplomatischen Code zur Falltürfunktion. *Spektrum der Wissenschaft. Dossier Kryptographie*, 2001:12–18, 1999.

[134] Klaus Schmeh. *Codeknacker gegen Codemacher*, volume 2. W3l, 2008.

[135] Klaus Schmeh. MysteryTwister C3, The Crypto Challenge Contest, Double Column Transposition, (No date). https://www.mysterytwisterc3.org/en/challenges/level-x/double-column-transposition, [Accessed: December, 4th, 2013].

[136] Klaus Schmeh. Wettrennen der Codeknacker, 2008. http://www.heise.de/tp/artikel/26/26876/1.html, [Accessed: December, 4th, 2013].

[137] Klaus Schmeh. Top-25 der ungelösten Verschlüsselungen - Platz 5: Die Doppelwürfel-Challenge, 2013. http://scienceblogs.de/klausis-krypto-kolumne/2013/09/13/, [Accessed: December, 4th, 2013].

[138] Craig Bauer. Unsolved: The World Greatest Codes and Ciphers, (No date). http://wdjoyner.org/video/bauer/bauer-Unsolved-ciphers.pdf, [Accessed: December, 4th, 2013].

[139] Mark Davies. N-Grams Data - Corpus of Contemporary American English, Samples, Level 1 (Free), (No date). http://www.ngrams.info/download_coca.asp, [Accessed: December, 4th, 2013].

[140] Thorsten Brants and Alex Franz. Web 1T 5-gram Version 1, 2006. http://catalog.ldc.upenn.edu/LDC2006T13, [Accessed: December, 4th, 2013].

[141] Jude Patterson. The Headline Puzzle, 2013. https://sites.google.com/site/theheadlinepuzzle/home, [Accessed: December, 4th, 2013].

[142] Arno Wacker, Bernhard Esslinger, and Klaus Schmeh. MysteryTwister C3, The Crypto Challenge Contest, Double Columnar Transposition – Reloaded, 2013. https://www.mysterytwisterc3.org/en/challenges/level-iii/double-column-transposition-reloaded-part-1, [Accessed: December, 16th, 2013].

[143] Frank Carter. *The first breaking of Enigma: Some of the pioneering techniques developed by the Polish Cipher Bureau, Number 10*. The Bletchley Park Trust Reports, 2008.

[144] David Steven Dummit and Richard M. Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

[145] Alex Kuhl. Rejewski's catalog. *Cryptologia*, 31(4):326–331, 2007. doi: 10.1080/01611190701299487.

[146] Rohit Vobbilisetty, Fabio Di Troia, Richard M. Low, Corrado Aaron Visaggio, and Mark Stamp. Classic cryptanalysis using hidden markov models. *Cryptologia*, 41(1):1–28, 2017. doi: 10.1080/01611194.2015.1126660.

ISBN 978-3-7376-0458-1