UNIVERSITÄT KASSEL
FACHBEREICH ELEKTROTECHNIK/INFORMATIK
FACHGEBIET ANGEWANDTE INFORMATIONSSICHERHEIT

Bachelorarbeit

Implementierung und Kryptoanalyse der M-138 in CrypTool 2.0

Nils Rehwald
Matrikelnummer: 31202848

UNIKASSEL
VERSITÄT
Fachgebiet Angewandte Informationssicherheit
Fachbereich Elektrotechnik/Informatik
Universität Kassel

29. April 2015

Prüfer:
Prof. Dr. Arno Wacker
Prof. Dr. Sebastian Petersen
Betreuer:
M. Sc. Nils Kopal

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Aufgabenstellung . . . 2
- 1.3 Ziele der Arbeit . . . 4
- 1.4 Aufbau der Arbeit . . . 4

2 Grundlagen **7**
- 2.1 Kryptographie . . . 7
- 2.2 Kryptoanalyse . . . 9
- 2.3 Angriffe . . . 10
  - 2.3.1 Known-Plaintext Angriff . . . 10
  - 2.3.2 Ciphertext-Only Angriff . . . 10
  - 2.3.3 Heuristische Verfahren . . . 11
  - 2.3.4 Brute-Force Angriff . . . 13
  - 2.3.5 Chosen-Plaintext Angriff . . . 14
- 2.4 CrypTool 2 . . . 14

3 M-138 **17**
- 3.1 Geschichte . . . 17
- 3.2 Funktionsweise . . . 18
- 3.3 Schlüsselraumgröße . . . 20
- 3.4 Unizitätslänge . . . 21

4 Kryptoanalyse **23**
- 4.1 Schwächen . . . 23
- 4.2 Angriffe . . . 24
  - 4.2.1 Known-Plaintext Angriff . . . 24
  - 4.2.2 Ciphertext-Only Angriff . . . 25
    - 4.2.2.1 Hill-Climbing . . . 25
    - 4.2.2.2 Brute-Force Angriff . . . 26
  - 4.2.3 Partially Known-Plaintext Angriff . . . 27

5 Konzept **29**
- 5.1 Die M-138 in CrypTool 2 . . . 29
- 5.2 M-138 Analyse in CrypTool 2 . . . 31

6 Design und Implementierung **35**
- 6.1 Die M-138 . . . 35
- 6.2 Das M-138 Analysewerkzeug . . . 39

6.2.1 Known-Plaintext Angriff 40
6.2.2 Ciphertext-Only Angriff 43
6.2.3 Partially Known-Plaintext Angriff 46

7 Evaluation der Analysekomponente 49
7.0.4 Metriken 49
7.0.5 Messungen 49
7.0.6 Diskussion der Ergebnisse 51
7.0.7 Challenges von Klaus Schmeh 55
7.0.7.1 Challenge 1 55
7.0.7.2 Challenge 2 56
7.0.7.3 Challenge 3 56
7.0.7.4 Challenge 4 58
7.0.7.5 Ursprüngliche Challenges 58

8 Ausblick und Zusammenfassung 59
8.1 Zusammenfassung 59
8.1.1 (R1) Verschlüsseln und Entschlüsseln von Texten 59
8.1.2 (R2) Visuelle Darstellung des Werkzeugs 59
8.1.3 (R3) Known-Plaintext Angriff auf ein Paar von Klartext und Geheimtext 60
8.1.4 (R4) Ciphertext-Only Angriff auf einen Geheimtext 60
8.2 Ausblick 61

Literaturverzeichnis 65

Abbildungsverzeichnis

2.1 Beispiel eines Schlüsselraums 12
2.2 Caesar Chiffre in CrypTool 2 15

3.1 Rahmen der M-138 18

5.1 Schema der Komponente M-138 29
5.2 Schema der Visualisierung der M-138 Komponente 30
5.3 Schema des M-138 Analyzers 31

6.1 Die M-138 Komponente in CrypTool 2 36
6.2 Visualisierung der M-138 Komponente 38
6.3 M-138 Analysewerkzeug in CrypTool 2 40
6.4 M-138 Analysewerkzeug in CrypTool 2 42
6.5 Visualisierung des Ciphertext-Only Angriffs 46

7.1 Laufzeit des Hill-Climbing in Abhängigkeit von der Textlänge . . . 51
7.2 Erfolgsrate des Hill-Climbing Algorithmus nach Textlänge 52
7.3 Aufwand des Hill-Climbing Algorithmus 52
7.4 Aufwand des Hill-Climbing Algorithmus 53
7.5 Gefundene Schlüssel bei Known-Plaintext Angriff 54
7.6 Gefundene Schlüssel bei Known-Plaintext Angriff 54

1 Einleitung

Diese Arbeit beschäftigt sich mit der M-138, einem Verschlüsselungswerkzeug, das die amerikanischen Streitkräfte in der Mitte des 20. Jahrhunderts, unter anderem im 2. Weltkrieg, benutzten. Die M-138 schafft es, trotz ihres recht einfachen Aufbaus eine für die Zeit, in der sie eingesetzt wurde starke Verschlüsselung zu ermöglichen. Dies liegt daran, dass das Werkzeug eine grosse Zahl von möglichen Schlüsseln bietet. Dies macht die Verschlüsselung resistent gegen das komplette Durchsuchen des Schlüsselraums. Mit heute bekannten, heuristischen Methoden kann die Verschlüsselung jedoch angegriffen werden. Daher ist eine Implementierung des Werkzeugs in CrypTool 2 interessant, um dessen Vor- und Nachteile zu verdeutlichen.

1.1 Motivation

Auch wenn klassische Verschlüsselungsverfahren im heutigen Alltag nicht mehr eingesetzt werden, kann die Analyse selbiger auch heute noch wertvolle Informationen liefern, die zur Lösung anderer Probleme genutzt werden können. Es gibt zwei Hauptgründe, warum diese nicht mehr genutzt werden. Einer ist, dass viele klassische Verfahren mit der heute zur Verfügung stehenden Rechenleistung zu einfach zu brechen sind. Ein Anderer ist, dass sie durch heuristische Verfahren angreifbar sind. Dies liegt daran, dass bei klassischen Chiffren eine kleine Änderung des Schlüssels auch nur eine kleine Änderung des resultierenden Geheimtextes zur Folge hat. Dadurch kann man sich durch mehrere aufeinander folgende Änderungen des Schlüssels dem korrekten, verwendeten Schlüessel annähern. Weitere, allerdings nicht so schwer wiegende Probleme klassischer Verfahren, sind zum Beispiel der Schlüsselaustausch, die Effizienz der Verschlüsselung oder die praktikablilität in heutiger Zeit, die vor allem bei großen Maschinen gering ist.
Dennoch ist es auch heute noch interessant, sich mit klassischen Verschlüsselungen zu beschäftigen. Ein Grund dafür ist, dass es historische Texte gibt, die bis heute nicht entschlüsselt werden konnten. Beispiele hierfür sind im 2. Weltkrieg mit der Enigma verschlüsselte Nachrichten, die bis heute nicht geknackt werden konnten. [Hoe] Ein Weiteres, etwas moderneres Beispiel, für das allerdings auch klassische Verschlüsselungen verwendet wurden, ist die *Kryptos* Skulptur, die der amerikanische Bildhauer Jim Sanborn erschuf. Diese enthält vier verschlüsselte Absätze. [Lev09] Während die ersten drei Absätze im Laufe der Zeit gelöst werden konnten, gibt es bis heute keine bekannte Lösung des vierten Teils. Die Beschäftigung mit klassischer Kryptologie kann helfen, in Zukunft Lösungen für dieses oder andere

Rätsel zu finden. Die ist insbesondere im Fall verschlüsselter Texte, deren Urheber nicht bekannt sind oder nicht mehr leben, interessant, da diese Texte sonst wahrscheinlich niemals entschlüsselt werden würden.
Andererseits ist die klassische Kryptologie für die Lehre interessant. Die Verfahren sind einfacher nachzuvollziehen als komplexe, moderne Algorithmen. Dadurch kann man Menschen mit Hilfe einfacher Verschlüsselungen, die Teilweise mit Stift und Papier gebrochen werden können, gut an die Kryptologie heranführen.
Diese Ansicht teilt auch der Kryptologe Klaus Schmeh, der als Anreiz, sich mit der M-138 zu beschäftigen, verschiedene Challenges [Sch] veröffentlichte. Er veröffentlichte drei verschieden lange Texte und stellte die Aufgabe, diese zu entschlüsseln. Für diese Challenges entwarf er eine eigene Version der M-138, da über das originale Werkzeug nicht alle notwendigen Informationen bekannt sind. Die von Klaus Schmeh veröfentlichten Geheimtexte sind 50, 75 und 100 Zeichen lang. Später wurden im Rahmen des Mystery Twister C3 Wettbewerbs [BE] vier neue Challenges veröffentlicht. Die Schwierigkeit dieser Challenges variiert stärker als die der originalen Challenges von Klaus Schmeh. Diese Aufgaben bestärkten die Motivation, sich mit der M-138 zu beschäftigen, da das Ziel erreicht werden sollte, im Laufe der Bachelorarbeit zumindest einige dieser Challenges lösen zu können.

1.2 Aufgabenstellung

Die wörtliche Aufgabenstellung lautet wie folgt:
"Die M-138 (auch CSP-845) ist ein Chiffriersystem, das von den US-Streitkräften in der ersten Hälfte des 20. Jahrhunderts eingesetzt wurde. Die M-138 besteht aus einem Aluminumramen, in dem Papierstreifen eingelegt werden können. Auf jedem dieser Papierstreifen ist ein zufälliges Alphabet aufgedruckt. Der Ramen besitzt außerdem eine aufgedruckte Nummerierung von 0 bis 25 (die Offsets). Um einen Text verschlüsseln zu können, müssen Sender und Empfänger sich im Vorfeld auf einen geheimen Schlüssel einigen. Dieser Schlüssel besteht zum einen aus einer Zahl zwischen 0 und 25, dem Schlüssel-Offset, und zum anderen aus der Auswahl von 25 verschiedenen Papierstreifen, welche in die Maschine eingelegt werden. Insgesamt gibt es 100 verschiedene Papierstreifen.[1] Ein Schlüssel könnte somit wie folgt aussehen: (42, 19, 26, 28, 02, 17, 49, 38, 87, 08, 94, 64, 92, 88, 37, 63, 39, 35, 30, 31, 05, 27, 34, 78, 60 / 6). Hier sind die Streifen 42, 19, 26, ... , 60 einlegt und der zu nutzende Offset ist 6. Zum Verschlüsseln einer Nachricht werden die Streifen so verschoben, dass vertikal unter dem Offset 0 die Nachricht steht. Der verschlüsselte Text wird dann unter dem vereinbarten Offset abgelesen und an den Empfänger, z.B. per Morsecode, übertragen. Der Empfänger kann die Nachricht nun entschlüsseln, indem er seine Streifen nun so verschiebt, dass der empfangene Geheimtext unter dem Offset 6 steht. Er kann die

[1] Die in dieser Arbeit beschriebe Variante entspricht nicht der originalen M-138 sondern ist ein fiktives Modell, welches der Kryptologe Klaus Schmeh entwickelt hat. Die echten Parameter, wie z.B. die Alphabete der Streifen, die Anzahl der Streifen, usw. waren in der realen Version anders.

Klartext-Nachricht nun unter dem Offset 0 ablesen. Sollen mehr als 25 Buchstaben verschlüsselt werden, werden für die folgenden Buchstaben die selben Streifen verwendet. Somit handelt es sich bei der M-138 um eine periodische polyalphabetische Chiffre. CrypTool 2.0 —der Nachfolger der bekannten e-Learning Anwendung für Kryptographie und Kryptoanalyse CrypTool —ist ein Open-Source Projekt, welches Lernenden, Lehrenden und Entwicklern mit Interesse an der Kryptographie die Möglichkeiten eröffnet selbst verschiedene kryptographische und kryptoanalytische Verfahren anzuwenden und auszuprobieren. Die moderne Benutzeroberfläche ermöglicht per einfachem und intuitiven Drag & Drop das Erstellen von einfachen bis hin zu sehr komplexen kryptographischen Algorithmen. Dies geschieht dabei mit einer graphischen Programmiersprache, welche speziell für diesen Zweck entwickelt wurde. Der Benutzer kann dabei ohne besondere Programmierkenntnisse die Algorithmen miteinander verbinden und so eigene neue Algorithmen und Abläufe erschaffen und testen. CrypTool 2.0 basiert auf modernsten Techniken wie z.B. das .NET Framework (zurzeit 4.0 SP1) und der Windows Presentation Foundation (WPF). Darüber hinaus ist die Architektur von CrypTool 2.0 vollständig Plug-In- basiert und modular aufgebaut, wodurch die Entwicklung neuer Funktionalitäten stark vereinfacht wird. Im Rahmen dieses Projekts wurden bereits eine Mehrzahl von kryptographischen Algorithmen wie z.B. AES, SHA1 oder die Enigma als Komponenten entwickelt. Hauptgegenstand dieser Bachelorarbeit ist die Implementierung der M-138-Chiffre für Lehr- und Lernzwecke sowie die Implementierung von Kryptoanalysewerkzeugen für die M-138 innerhalb von CrypTool 2.0. Hierfür soll zunächst eine Analyse vorgenommen werden, wie Komponenten für die Visualisierung und die Analyse funktionieren könnten. Anschließend sollen diese Konzepte als CrypTool 2.0 Komponenten in C# mit passender CrypTool 2.0 Visualisierung in WPF umgesetzt werden. Ziel der Komponenten ist es, Benutzern von CrypTool 2.0 die Grundlagen und die Funktionsweise der M-138 sowie deren Kryptoanalyse anschaulich zu vermitteln. Hierfür müssen auch geeignete Vorlagen und Hilfen innerhalb von CrypTool 2.0 umgesetzt werden. Die Analysekomponenten[2] sollen einen Known-Plaintext- Angriff sowie einen Ciphertext-Only-Angriff auf die Chiffre ermöglichen. Die konkreten Kryptoanalyse- Algorithmen, welche auf Hill-Climbing basieren, werden vom Betreuer vorgegeben und in enger Abstimmung mit diesem entwickelt und implementiert. Ein weiteres Ziel der Arbeit ist es, die kryptographische Stärke der M-138 (zusammen mit dem Betreuer) abzuschätzen. Hierfür muss die Größe des Schlüsselraums, die Unizitätslänge der Chiffre, die Komplextität (z.B. Zeit und benötigte Länge des Geheimtextes) der entwickelten Angriffe sowie deren Mächtigkeit evaluiert werden.

[2]Es kann sich um eine oder mehrere Analysekomponenten handeln. Beide Angriffe könnten auch innerhalb einer Komponente implementiert werden.

1.3 Ziele der Arbeit

Ziel der Arbeit ist es, Informationen über die Funktionsweise der M-138 zusammenzutragen und anhand der ermittelten Informationen ein Modell des Werkzeuges in CrypTool 2 zu erstellen. Neben einer schriftlichen Ausarbeitung, die die Funktionsweise der M-138 darstellen und analysieren soll, besteht das Ziel dieser Arbeit darin, Komponenten für CrypTool 2 zu erstellen, mit deren Hilfe man sowohl das Werkzeug simulieren als auch die Verschlüsselung angreifen kann. Im schriftlichen Teil der Arbeit soll die M-138 erklärt und analysiert werden, vor allem im Bezug auf die kryptographische Stärke der Verschlüsselung. Ausgehend davon sollen verschiedene mögliche Angriffe auf die Verschlüsselung analysiert und ihre Verwendbarkeit unter Berücksichtigung der Komplexität der Verschlüsselung abgeschätzt werden. Darüber hinaus sollen die im Programmierteil zu erstellenden Komponenten folgende Anforderungen erfüllen:

- (R1) Verschlüsseln und Entschlüsseln von Texten nach Funktionsweise der M-138.
- (R2) Visuelle Darstellung des Werkzeugs beim Ver- und Entschlüsseln.
- (R3) Möglichkeit, einen Known-Plaintext Angriff auf ein vom Benutzer gegebenes Paar von Klartext und Geheimtext auszuführen.
- (R4) Möglichkeit, einen Ciphertext-Only Angriff auf einen vom Benutzer vorgegebenen Geheimtext auszuführen.

Im Anschluss soll ein rückblickendes Fazit gezogen werden. Außerdem sollen Ideen, die während des Schreibens der Arbeit aufkommen, aber nicht mehr getestet werden können, gesammelt werden.

1.4 Aufbau der Arbeit

Die Arbeit ist in acht Kapitel unterteilt. Im auf die Einleitung folgenden Kapitel 2 werden die zum Verständnis der Arbeit benötigten Grundlagen vermittelt. Dies umfasst neben kryptographischen Grundlagen und Begrifflichkeiten auch eine Erklärung zu CrypTool 2.
Kapitel 3 beschäftigt sich mit der Geschichte des Verschlüsselungswerkzeugs und analysiert dieses anhand kryptographischer Gesichtspunkte. Das darauf folgende Kapitel 4 gibt einen Überblick über die kryptographischen Schwächen der M-138. Mögliche Angriffe auf die Verschlüsselung und Konzepte, wie diese in CrypTool 2 implementiert werden können, werden in Kapitel 5 diskutiert. Kapitel 6 veranschaulicht die Umsetzung dieser Komponenten und erklärt, wie die neuen Komponenten in CrypTool 2 implementiert wurden und funktionieren. Die Evaluation der Arbeit in Kapitel 7 zieht eine Bilanz der getesteten Angriffe. Dabei wird unter anderem die Effizienz und die Verwendbarkeit der Angriffe zum Lösen der bereits

erwähnten Challenges getestet.
Im letzten Kapitel 8 werden die in der Arbeit erreichten Ergebnisse bewertet und die Ideen, die nicht im Rahmen dieser Arbeit getestet wurden, vorgestellt.

2 Grundlagen

Dieses Kapitel gibt einen Überblick über die zum Verständnis dieser Arbeit erforderlichen Grundlagen. Dazu werden zuerst allgemeine Grundlagen der Kryptographie erläutert. Danach werden grundlegende Prinzipien der Kryptoanalyse erläutert und einige verschiedene Arten von Angriffen dargestellt. Zuletzt folgt eine Erklärung zu dem verwendeten Programm CrypTool 2.

2.1 Kryptographie

"Ursprünglich war die Kryptographie die Lehre von der Datenverschlüsselung." [Buc10] Das Hauptziel der Kryptographie ist die Ermöglichung einer Kommunikation zwischen Personen, ohne dass unbeteiligte Dritte die Möglichkeit haben, die ausgetauschten Nachrichten zu lesen, auch wenn sie es geschafft haben, diese während der Übertragung abzufangen. Im Allgemeinen wird dies als Geheimhaltung bezeichnet. Dies unterscheidet die Kryptographie von der Steganographie. Während das Ziel der Steganographie ist, Nachrichten zu verstecken, sodass diese nicht gefunden werden, ist das Ziel der Kryptographie, dass nur berechtigte Personen die Möglichkeit haben, diese zu lesen, auch wenn Dritte Zugriff auf die Nachricht während der Übertragung haben. Dazu wurden Verfahren entwickelt, mit denen Nachrichten verschlüsselt und entschlüsselt werden können. Durch die Benutzung von geheimen Schlüsseln wird sichergestellt, dass nur die Personen, die im Besitz des verwendeten Schlüssels sind, die Nachricht lesen können. Im Laufe der Zeit kamen jedoch neue Ziele hinzu. Heute existieren neben der Geheimhaltung des Nachrichteninhaltes under anderem folgende neue Ziele: [Wac14]

- Datenintegrität, das heißt, dass es für eine dritte Person nicht möglich ist, eine Nachricht zwischen zwei Personen abzufangen und zu verändern, ohne dass dies dem Empfänger auffällt.
- Authenzität, das heißt, dass ein Empfänger einer Nachricht sicher sein kann, von dem die Nachricht stammt, also niemand Drittes im Namen von einer Person eine Nachricht versenden kann.
- Verantwortlichkeit, das heißt, dass man dem Sender einer Nachricht nachweisen kann, dass er und niemand anders diese Nachricht verfasst hat.
- Zugriffskontrolle, das heißt, dass eine Resource abgesichert wird, sodass nur bestimmte Personen Zugriff auf sie haben.

Im Rahmen dieser Arbeit konzentrieren wir uns allerdings auf das Ziel der Geheimhaltung, da die anderen Ziele durch das behandelte Verschlüsselungswerkzeug, die M-138, nicht sichergestellt werden können.

In der Kryptographie werden üblicherweise folgende Abkürzungen verwendet:

- P - Plaintext (Klartext), der Text, der verschlüsselt wird.
- C - Ciphertext (Geheimtext), der verschlüsselte Text, der an den Empfänger übermittelt wird.
- K - Key (Schlüssel), der zur Verschlüsselung verwendet wird.
- S - Schlüsselraum eines Verschlüsselungsverfahren, die Menge aller Schlüssel, die es in diesem gibt.

Kryptographische Verfahren zur Verschlüsselung kann man in zwei verschiedene Arten unterteilen, symmetrische und asymmetrische Verschlüsselungen. Während bei einer symmetrischen Verschlüsselung der Sender den gleichen Schlüssel zum verschlüsseln wie der Empfänger zum Entschlüsseln nutzt, basieren asymmetrische Verschlüsselungen auf unterschiedlichen Schlüsseln für die Kommunikationspartner. Dabei hat jeder Gesprächspartner einen öffentlich bekannten Schlüssel, mit dem jeder, der eine Nachricht an ihn senden will, diese verschlüsseln kann. Zum Entschlüsseln kommt ein geheimer, privater Schlüssel zum Einsatz, den nur der Empfänger kennt. Dies eliminiert das Problem, dass der Schlüssel zwischen den Kommunikationspartnern bei der Verwendung symmetrischer Verschlsselung ausgetauscht werden muss. Verschlüsselungen, die damit arbeiten, Buchstaben durch andere Buchstaben zu ersetzen, nennt man Substitutionschiffren. Weiterführend gibt es Verschlüsselungen, die die Buchstaben des Klartextes nicht ersetzen, sondern ihre Reihenfolge verändern. Diese gehören zur Gruppe der Transpositionschiffren. Allerdings wird in der Arbeit nicht näher auf diese Art der Verschlüsselung eingegangen. In der Gruppe der Substitutionschiffren kann man zwischen monoalphabetischen und polyalphabetischen Substitutionen unterscheiden. Bei einer monoalphabetischen Verschlüsselung gibt es eine feste Zuordnung von Klartextbuchstaben auf Geheimtextbuchstaben. Derselbe Klartextbuchstabe, zum Beispiel A, wird immer demselben Geheimtextbuchstaben, zum Beispiel D, zugeordnet. Dies ist unabhängig von der Position des Buchstabens im Text. Die wahrscheinlich bekannteste Verschlüsselung dieser Art ist die *Caesar Chiffre*. Bei dieser wird einfach jeder Klartextbuchstabe im Geheimtext zu dem Buchstaben, der im Alphabet drei Stellen nach dem Klartextbuchstaben steht.
Bei Polyalphabetischen Verschlüsselungen gibt es so eine eindeutige Zuordnung nicht. Statt nur einer Zuordnung von Klartextalphabet zu Geheimtextalphabet werden hier mehrere verwendet. Welche Zuordnung für einen Buchstaben verwendet wird, hängt in diesem Fall von der Position des Buchstabens im Text ab. Die wohl bekannteste Verschlüsselung dieser Art ist die *Vigenére Verschluesselung*. Diese verwendet verschiedene monoalphabetische Zuordnungen, deren Anwendung abhängig von der Position eines Buchstabens ist. So kann es bei dieser Verschlüsselung passieren, dass der Klartextbuchstabe A an erster Stelle des Textes

dem Geheimtextbuchstaben F und an zweiter Stelle des Textes dem Geheimtextbuchstaben Y zugeordnet wird.

2.2 Kryptoanalyse

Die Kryptoanalyse beschäftigt sich mit der Analyse kryptographischer Verfahren. Dabei wird das Verfahren auf seine Sicherheit und mögliche Angriffe analysiert. Das ursprüngliche Ziel der Kryptoanalyse war es, Geheimtexte abzufangen und zu entschlüsseln, ohne den verwendeten Schlüssel zu kennen. Inzwischen ist die Kryptoanalyse auch ein wichtiges Werkzeug, um Verschlüsselungen auf ihre Sicherheit zu analysieren. Ein wichtiges Prinzip dr Kryptographie ist das Prinzip von Kerckhoff. Dies besagt, dass die Sicherheit einer Verschlüsselung einzig vom verwendeten Schlüssel abhängen darf. Es muss also davon ausgegangen werden, dass dem Angreifer alle Informationen über die Funktion der Verschlüsselung bekannt sind.
Die kryptographische Stärke einer Verschlüsselung ist von mehreren Faktoren abhängig. Ein wichtiger Faktor ist die Schlüsselraumgroesse. Dieser Wert gibt an, wie viele Schluessel es für eine Verschlüsselung gibt. Ein wichtiges Kriterium für die Sicherheit einer Verschlüsselung ist es, dass der Schlüsselraum außreichend groß ist. Es darf nicht möglich sein, den gesamten Schlüsselraum einer Verschlüsselung in einer akzeptabelen Zeit zu durchsuchen. Ist dies nicht der Fall, ist die Verschlüsselung nicht brauchbar, da jede mit ihr verschlüsselte Nachricht einfach entschlüsselt werden kann, indem alle möglichen Schlüssel durchprobiert werden.

Ein wichtiges Werkzeug zum Angriff auf Verschlüsselungen ist der Koinzidenzindex. Dieser analysiert die Wahrscheinlichkeiten des Vorkommens aller Buchstaben und fasst dies zusammen. Dadurch kann man mit Hilfe des Koinzidenzindex bewerten, ob ein Text einer natürlichen Sprache ähnelt oder nicht. Kommt zum Beispiel jeder Buchstabe im Geheimtext gleich oft vor, weicht der Koinzidenzindex sehr stark von dem Wert ab, den er für eine natürliche Sprache hat, in der verschiedene Buchstaben verschieden oft vorkommen. Dadurch kann man feststellen, mit Hilfe was für einer Chiffre ein Text verschlüsselt wurde. In dem Fall, dass eine Transpositionschiffre oder eine monoalphabetische Verschlüsselung genutzt wurden, ändert sich der Koinzidenzindex nur minimal im Vergleich zur natürlichen Sprache. Die Änderung resultiert in diesem Fall lediglich daraus, dass ein Text immer nur einen Teil einer Sprache darstellt und die Häufigkeit, mit der die Buchstaben vorkommen, nicht perfekt dem für die Sprache ermittelten Wert entspricht. Wurde jedoch eine polyalphabetische Verschlüsselung genutzt, so ändert sich der Koinzidenzindex deutlich. Wurde eine periodische polyalphabetische Verschlüsselung genutzt, so kann man mit Hilfe des Koinzidenzindex auch die Länge der Periode abschätzen. Eine polyalphabetische Verschlüsselung bedeutet, dass sich die monoalphabetischen Zuordnungen innerhalb der Verschlüsselung periodisch wiederholen. Dies ist hilfreich, da man die Verschlüsselung somit wieder mit Hilfsmitteln, mit denen man auch monoalphabetische Verschlüsselungen bearbeitet, angreifen kann. Dies

liegt daran, dass in einer polyalphabetischen Verschlüsselung alle N Zeichen dieselbe monoalphabetische Verschlüsselung zum Einsatz kommt. Daraus folgt, dass eine polyalphabetische Verschlüsselung umso sicherer ist, je länger ihr Schlüssel und somit ihre Periode ist.
Die Länge des verschlüsselten Textes ist auch wegen der Unizitätslaenge einer Verschlüsselung wichtig für die Sicherheit dieser. Dieser Wert gibt an, wie lang ein Geheimtext sein muss, damit man genau einen Schlüssel im Schlüsselraum findet, der dazu führt, dass man nach dem Entschlüsseln einen sinnvollen Text erhält. Ist ein Text kürzer als die Unizitätslaenge, kann es passieren, dass man mehrere Schlüssel findet, die den Geheimtext auf verschiedene, sinnvolle Klartexte abbilden.

2.3 Angriffe

Basierend auf diesen Grundlagen gibt es verschiedene Möglichkeiten, kryptographische Verfahren anzugreifen. Diese kann man in unterschiedliche Kategorien einteilen, abhängig davon, welche Informationen dem Angreifer zur Verfügung stehen.

2.3.1 Known-Plaintext Angriff

Ein Known-Plaintext Angriff basiert darauf, dass dem Angreifer ein Stück Geheimtext sowie der dazugehörige Klartext bekannt sind. Das Ziel dieses Angriffs ist es, den verwendeten Schlüssel herauszufinden. Dieser kann nützlich sein, um mit ihm weitere abgefangene Geheimtexte zu entschlüsseln, falls derselbe Schlüssel bereits einmal verwendet wurde. Selbiges ist möglich, falls der Schlüssel zu einem späteren Zeitpunkt wieder verwendet wird. Dies ist zum Beispiel wahrscheinlich, wenn der Schlüssel einen gesamten Tag lang benutzt wird. Außerdem ist es somit möglich, eine Datenbank verwendeter Schlüssel zu erstellen, um Gemeinsamkeiten zu analysieren und vielleicht vorhersagen zu können, wie andere verwendete Schlüssel aussehen könnten. Für diesen Angriff versucht man, mit Hilfe eines Schlüssels den bekannten Klartext auf den bekannten Geheimtext abzubilden. Die Funktionsweise dieses Angriffs ist stark von der Funktion der verwendeten Verschlüsselung abhängig und kann nicht verallgemeinert werden.

2.3.2 Ciphertext-Only Angriff

Das Prinzip des Ciphertext-Only Angriffs basiert darauf, dass dem Angreifer nur der Geheimtext und nach dem Prinzip von Kerkhoff, die Funktionsweise der Verschlüsselung bekannt sind. Der Angreifer hat in diesem Fall zwei Möglichkeiten. Entweder kann er den gesamten Schlüsselraum der Chiffre angreifen, indem er diesen komplett durchsucht oder sich eine Schwäche der verwendeten Verschlüsselung zum Vorteil machen. So können zum Beispiel viele Verfahren mit Hilfe heuristischer

Verfahren angegriffen werden. Außerdem haben verschiedene Verschlüsselungen Einschränkungen, die sie besser angreifbar machen. Ein Beispiel hierfür ist die Enigma, die die Schwäche hat, dass sie beim Verschlüsseln niemals einen Klartextbuchstaben auf sich selbst abbilden kann. Ziel des Angriffs ist, wie schon beim Known-Plaintext Angriff, das Herausfinden des verwendeten Schlüssels. Ein weiteres Ziel des Ciphertext-Only Angriffs ist es allerdings auch, den Geheimtext zu entschlüsseln und somit den dazugehörigen Klartext herauszufinden. Wurde eine periodische, polyalphabetische Verschlüsselung mit variabler Schlüssellänge verwendet, muss man zum Beispiel zuerst die Schlüssellänge abschätzen, bevor man einen Ciphertext-Only Angriff erfolgreich durchführen kann. Dazu kann man die oben erwähnten Hilfsmittel wie den Koinzidenzindex nutzen. Für Ciphertext-Only Attacken werden in der Regel entweder Brute-Froce Angriffe oder heuristische Verfahren verwendet.

2.3.3 Heuristische Verfahren

Heuristische Verfahren basieren auf der Annahme, dass bei der Verschlüsselung und Entschlüsselung von Texten eine kleine Änderung im Schlüssel auch nur eine kleine Änderung im resultierenden Text zur Folge hat. Da diese Annahme nicht auf alle kryptographischen Verfahren zutrifft, ist es nicht möglich, jede Verschlüsselung mit Hilfe heuristischer Verfahren anzugreifen.
Ein Beispiel für ein heuristisches Verfahren ist das Hill-Climbing, das im Deutschen auch als Bergsteigealgorithmus bezeichnet wird. Dies basiert darauf, dass manche Schlüssel besser sind als andere. Um dies zu bewerten, verwendet man sogenannte Kostenfunktionen. Diese analysieren einen Text auf das Vorkommen von Buchstabenkombinationen wie Bigrammen (Buchstabenpaare), Trigramme (drei aufeinander folgende Buchstaben), die Häufigkeit, mit der bestimmt Buchstaben vorkommen, und andere Eigenheiten einer Sprache. Entscheidend dafür ist es, die Sprache, die für den Klartext verwendet wurde, zu kennen, da die Bewertung der Texte von der verwendeten Sprache abhängt. In der englischen Sprache kommt zum Beispiel das Trigram *the* relativ häufig vor, während es in der deutschen Sprache sehr selten vorkommt.
Der Algorithmus funktioniert wie folgt: Zuerst generiert man einen zufälligen Schlüssel K. Mit diesem Schlüssel K entschlüsselt man nun den Geheimtext C. Auf den so erhaltenen Klartext P wendet man nun die vorher definierte Kostenfunktion an und speichert den errechneten Wert. Im nächsten Schritt ändert man einen Teil des Schlüssels und entschlüsselt mit dem so erhaltenen neuen Schlüssel K' erneut den Geheimtext C. Auf den nun erhaltenen Klartext P' wendet man erneut die Kostenfunktion an und prüft, ob diese den Schlüssel K oder K' für besser befindet. Dazu vergleicht man die Werte, die die Kostenfunktion den Klartexten P und P' zuordnet. Bewertet die Kostenfunktion den Schlüssel K' für besser, so wird der Schlüssel K verworfen und der Schlüssel K' wird weiter verwendet. Dieser wird nun wie vorher K modifiziert. Der so erhaltene Schlüssel K'' wird nun ebenfalls wie vorher K und K' getestet. Diese Prozedur wiederholt sich, bis der

modifizierte Schlüssel von der Kostenfunktion nicht mehr als besser bewertet wird. Ist dies der Fall, dass der Schlüssel K für besser als der Schlüssel K' befunden wird, werden zunächste alle möglichen Modifikationen von K durchprobiert. Bewertet die Kostenfunktion alle Modifikationen von K schlechter als K selbst, so hat der Algorithmus ein sogennantes lokales Maximum gefunden. Dies bedeutet, dass es durch eine Modifikation des Schlüssels nicht möglich ist, einen besseren Schlüssel zu generieren. Das Problem hierbei ist, dass es innerhalb des Schlüsselraumes meist mehrere lokale Maxima gibt, wie in 2.1 zu sehen ist. Ein lokales Maximum bedeutet, dass mit Hilfe des Schlüssels ein Klartext entsteht, der von der Kostenfunktion als näher an einer gewählten Sprache angesehen wird als die Texte, die mit den benachbarten, leicht veränderten Schlüsseln, erzeugt werden. Dies bedeutet aber nicht, dass der getestete Schlüssel auch der gesuchte Schlüssel ist, da es sein könnte, dass es mehrere lokale Maxima gibt, und sich der gesuchte Schlüssel ganz woanders im Schlüsselraum befindet.

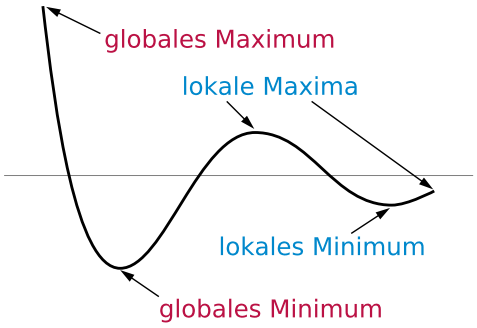

Abbildung 2.1: Beispiel eines Schlüsselraums. Auf der x-Achse befindet sich der Schlüsselraum, die y-Achse entspricht der Bewertung durch eine Kostenfunktion. Quelle: `http://upload.wikimedia.org/wikipedia/commons/f/fd/Extrema_example_de.svg`

Ein anderes Beispiel für heuristische Verfahren sind genetische Algorithmen. Diese hängen wie das Hill-Climbing von einer Kostenfunktion, die den Klartext analysiert, ab. Die Funktionsweise genetischer Algorithmen ist an die Evolutionstheorie angelehnt. Zuerst wird eine Menge von zufälligen Schlüsseln erzeugt, die als Ausgangsmenge verwendet werden. Diese Schlüssel werden nun mit Hilfe der Kostenfunktion auf ihre Qualität analysiert. Anschließend werden, bis eine Abbruchbe-

dingung erreicht wurde, einige Schritte immer wieder durchgeführt. Zuerst werden anhand ausgewählter Merkmale wie den zugeordneten Kostenwerten die Schlüssel, die weiterverwendet werden sollen, ausgewählt. Diese Schlüssel werden im nächsten Schritt miteinander kombiniert, sodass neue Schlüssel entstehen, die aus jeweils zwei Schlüsseln der vorigen *Generation* zusammengesetzt wurden. Im nächsten Schritt kommt es, wie in der Evolutionstheorie, zur Mutation. In diesem Schritt werden einige, zufällig gewählte Schlüssel ausgewählt und zufällig verändert. Im letzten Schritt werden die nun erhaltenen Schlüssel wieder mit Hilfe der Kostenfunktion bewertet und es werden wieder die weiterhin zu benutzenden Schlüssel ausgewählt. Diese Algorithmen basieren auf der Annahme, dass durch die Mischung zweier guter Schlüssel ein noch besserer Schlüssel entsteht.

2.3.4 Brute-Force Angriff

Der Brute Force angriff basiert, wie der Name übersetzt sagt, auf roher Gewalt. Bei diesem Angriff werden einfach alle möglichen Schlüssel durchprobiert. Ob ein Brute-Force Angriff sinnvoll ist, hängt daher von der Größe des Schlüsselraums der verwendeten Verschlüsselung ab. Je grösser der Schlüsselraum ist, desto länger dauert es, diesen zu durchsuchen. Dies lässt sich gut an dem Beispiel eines Passwortes verdeutlichen. Nehmen wir für dieses Beispiel an, dass ein Passwort sieben Zeichen lang ist. Wenn wir davon ausgehen, dass der Ersteller des Passwortes nur Kleinbuchstaben des deutschen Alphabets ohne Umlaute verwendet hat, so hatte er für jede Stelle des Passworts 26 Möglichkeiten, einen Buchstaben zu wählen. In diesem Fall gibt es also

$$S = 26^7 \approx 8.03 \cdot 10^9 \approx 2^{33}$$

Möglichkeiten, wie das Passwort aussehen könnte. Ändern wir die Voraussetzung nun und gehen davon aus, dass zwar immer noch nur Kleinbuchstaben, aber auch Umlaute verwendet worden sein können, so kommen wir auf

$$S = 29^7 \approx 1.72 \cdot 10^{10} \approx 2^{34}$$

Möglichkeiten, was einer Verdoppelung des zu durchsuchenden Raumes bedeutet. Mit heutiger Rechenleistung sind $10^{12} \frac{Schlüssel}{Sekunde}$ [Wac14] ein realistischer Wert, wie viele Schlüssel pro Sekunde durchprobiert werden können. Daher ist es zwingend notwendig, den Schlüsselraum eines Kryptosystems mathematisch zu analysieren, um abschätzen zu können, ob die Anwendung eines Brute-Force Angriffs sinnvoll und erfolgsversprechend ist. Ebenso ist die Abschätzung des Schlüsselraums bei dem Entwurf einer neuen Verschlüsselung zwingend notwendig, da eine Verschlüsselung, deren Schlüsselraum in annehmbarer Zeit durchsuchbar ist, nicht sinnvoll ist.

2.3.5 Chosen-Plaintext Angriff

Das Prinzip des Chosen-Plaintext Angriffs ist es, dass man die Möglichkeit hat, Nachrichten zu generieren, deren Inhalt man kennt. Dies kann bewerkstelligt werden, indem man eine Verschlüsselungsmaschine in seinen Besitz bringt, mit deren Hilfe man, ohne den verwendeten Schlüssel zu kennen, Texte verschlüsseln kann. Eine andere Möglichkeit ist es, den Feind dazu zu bringen, Nachrichten zu verschicken, deren Inhalt man zu kennen glaubt. Dies kann geschehen, indem man den Feind an einer bestimmten Stelle angreift oder einen feindlichen Verband einen gefälschten eigenen Funkspruch absichtlich abhören lässt, in der Hoffnung, dass dieser den abgefangenen Funkspruch meldet und man diese Nachricht abfangen kann. Das Ziel ist es in beiden Fällen, möglichst viele Paare von Klartext und Geheimtext zu generieren. Mit Hilfe dieser versucht man dann, Informationen über die Funktionsweise der Verschlüsselung zu erhalten, um diese mit Hilfe von Reverse Engineering brechen zu können. Des Weiteren erhält man durch die Paare von Klartext und Geheimtext die Möglichkeit, einen Known-Plaintext Angriff auf die Verschlüsselung anzuwenden.

Mit Hilfe einer Chosen-Plaintext Attacke kann man einige Informationen über die Art der Verschlüsselung erfahren. Anhand der Analyse vieler Texte kann man mit Hilfe der Analyse auf das Vorkommen von n-Grammen, also dem in Reihenfolge Vorkommen von n Buchstaben, herausfinden, ob eine Substitutions- oder Transpositionschiffre verwendet wird. Auch kann man mit Hilfe eines Chosen-Plaintext Angriffs gut die Schlüssellänge einer polyalphabetischen Verschlüsselung feststellen, indem man einen Text verschlüsselt, der nur aus demselben Buchstaben besteht. Anhand der Wiederholung des Geheimtextes erkennt man die wahrscheinliche Schlüssellänge.

2.4 CrypTool 2

CrypTool ist ein Lernwerkzeug für Schüler, Studenten und andere interessierte Personen, die die Grundlagen der Kryptografie auf anderem Wege erlernen wollen als aus trockenen Büchern.[Sch14] Die Entwicklung von CrypTool begann 1998 bei der Deutschen Bank unter der Leitung von Professor Berhard Esslinger. [KKWE14] Im Laufe der Zeit wuchs das Projekt und wurde 2003 als Open Source Software veröffentlicht. Später entstand ein Nachfolger, CrypTool 2, an dem heute verschiedene Universitäten und über 60 Personen arbeiten. [Ess08] CrypTool 2 wurde vor allem entwickelt, um Menschen die Möglichkeit zu geben, kryptographische Verfahren und Angriffe auf diese ausprobieren zu können und sie somit an die Kryptographie heranführen zu können.
Der Benutzer kann in CrypTool 2 verschiedene kryptographische Verfahren, vom bereits erwähnten Caesar Chiffre bis zum heute verwendeten *Advanced Encryption Standard* AES, simulieren. Dazu gibt es verschiedene Komponenten. Im Regelfall hat jede Verschlüsselung eine eigene Komponente, in der auch die genau-

en Einstellungen für die Verschlüsselung definiert werden können. Zusätzliche Informationen wie Texteingaben oder den Schlüssel muss der Benutzer, wie in 2.2 zu sehen ist, in der Regel angeben, indem er andere Komponenten mit der Verschlüsselungskomponente verbindet. Die funktioniert in CrypTool 2 per Drag and Drop. Somit ist es auch für unerfahrene Benutzer sehr einfach, Verschlüsselungen

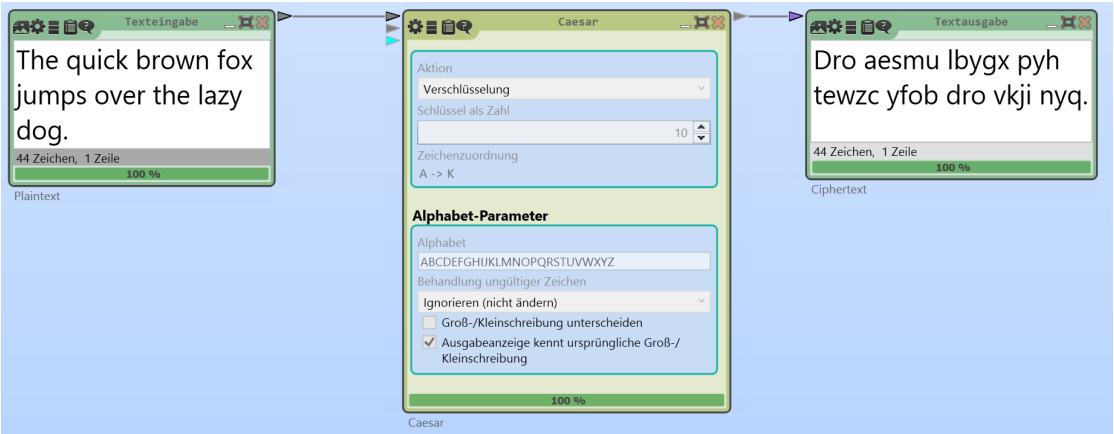

Abbildung 2.2: Beispiel einer Komponente in CrypTool 2. Links der Klartext als Eingabe, in der Mitte die eigentliche Komponente mit der Möglichkeit, Einstellungen zu ändern und rechts der Geheimtext als Ausgabe.
Quelle: Eigener Screenshot

nachzuvollziehen. Viele Komponenten haben zusätzlich Visualisierungen, die dem Benutzer visuell vermitteln, was die Komponente tut. Dadurch kann nachvollzogen werden, wie eine Verschlüsselung funktioniert.
Zum Einstieg gibt es außerdem fertige Vorlagen, die der Benutzer laden und direkt ausführen kann, ohne selber Angaben machen zu müssen. Durch diese Vorlagen werden auch komplexere Vorgänge, wie zum Beispiel ein Angriff auf die WEP Verschlüsselung, die lange Zeit genutzt wurde, um W-LAN Netze abzusichern, einfach dargestellt und teilweise erklärt.
Um die Verschlüsselungen noch besser zu veranschaulichen ist es bei vielen Komponenten in CrypTool 2 möglich, die Einstellungen der Verschlüsselung zu ändern, während diese ausgeführt wird. So kann man zum Beispiel einen Text verschlüsseln lassen und während die Komponente läuft, den Klartext ändern und live sehen, wie sich der Geheimtext ebenfalls verändert. Dadurch kann man zum Beispiel beobachten, was für eine Auswirkung Änderungen am Klartext oder am Schlüssel auf den Geheimtext haben.

3 M-138

Die M-138 ist ein Verschlüsselungswerkzeug, das die amerikanischen Streitkräfte im 20. Jahrhundert benutzten. Sie gehört zur Klasse der symmetrischen Verschlüsselungen und ist eine Substitutionschiffre.

3.1 Geschichte

Erfunden wurde die M-138 von Colonel Parker Hitt im Jahr 1916. [Mar] Ein verwandtes Verschluesselungswerkzeug ist die M-94. Diese funktioniert ähnlich wie die M-138. Sie besteht aus einem Satz von Ringen, auf deren Außenseite das Alphabet in zufälliger Reihenfolge eingraviert ist. Diese Ringe werden auf einen Stab gesteckt und so angeordnet, dass man den Klartext ablesen kann. Dreht man den Zylinder nun ein wenig, kann man in einer anderen Spalte den Geheimtext ablesen. Die Funktionsweise der M-138, die aus einem Aluminiumrahmen und einem Satz Papierstreifen besteht, ist ähnlich. Beide Werkzeuge wurden von Colonel Parker Hitt erfunden. Die Idee hinter den Maschinen ist allerdings älter. Bereits Thomas Jefferson erfand gegen Ende des 18. Jahrunderts ein Gerät, dessen Funktionsweise der der M-94 ähnelte. Dies ist heute als *Jefferson-Zylinder* oder *Jefferson-Walze* bekannt. Der Nachfolger der beiden Werkzeuge ist die M-209. Diese ist im Vergleich zur M-94 und der M-138 eine recht komplexe Maschine, deren Aussehen dem der deutschen *Enigma* ähnelt. Wie diese verwendet auch die M-209 mehrere Walzen, um den Text zu verschlüsseln. Im Gegensatz zur mit Strom arbeitenden Enigma funktioniert die M-209 jedoch rein mechanisch. Erstmals benutzt wurde die M-138 von der US Army im Jahr 1934. [T.12] Wenig später, im Jahr 1939, wurde die M-138 überarbeitet. So wurden in der neuen Version 30 Papierstreifen genutzt, während in der ursprünglichen M-138 nur 25 Streifen genutzt wurden. Diese Version wurde als M-138A von der US Army und als CSP-845 von der Marine genutzt. Während des zweiten Weltkrieges wurden zwar meistens Verschlüsselungsmaschinen wie die M-209 oder die SIGABA genutzt, jedoch wurden die M-94 und die M-138 weiterhin, vor allem als Notfallverschlüsselung oder in Fällen, in denen keine Verschlüsselungsmaschine verfügbar war, verwendet. Dies wurde dadurch begünstigt, dass diese Werkzeuge im Vergleich zu den Maschinen sehr billig und gut transportabel waren. [Sch]

3.2 Funktionsweise

Die M-138 besteht aus einem Aluminium- oder Plastikrahmen und einem Satz Papierstreifen. Auf diesen Streifen ist jeweils zweimal hintereinander das Alphabet in zufälliger Reihenfolge abgedruckt. Die Reihenfolge der Buchstaben des Alphabets ist beide Male gleich. Desweiteren ist am linken Ende des Streifens die Streifennummer aufgedruckt. Außerdem gibt es einen Metallrahmen, in den die Streifen reihenweise eingeschoben werden können. In den Metallrahmen passen untereinander 25 Streifen. Außerdem hat der Rahmen am oberen Rand eine Unterteilung in 25 Spalten und einen Metallstab, den man verschieben kann, um eine Spalte zu markieren.

Abbildung 3.1: M-138 ohne eingeschobene Papierstreifen
Quelle: `http://www.jproc.ca/crypto/venus_01.jpg`

Es existierten mehrere Gruppen von Streifen. Diese waren durch die auf der Rückseite aufgedruckte Gruppennummer erkenntlich. Leider ist nicht bekannt, wie viele Gruppen es gab und wie viele Streifen zu jeder Gruppe gehörten. Jedoch wurden zum Verschlüsseln immer nur Streifen einer Gruppe genutzt. Zum Verschlüsseln gab es einen aus zwei Teilen bestehenden Schlüssel, einen Tagesschlüssel und einen Nachrichtenschlüssel.
Der Tagesschlüssel bestand aus der Auswahl der zu benutzenden Streifen und der Reihenfolge, in der die Streifen in das Werkzeug eingeschoben werden sollten.

Dieser Schlüssel wurde aus einer Tabelle entnommen, in der die Schlüssel einzelnen Tagen zugeordnet waren. Jeder Tagesschlüssel war für 24 Stunden gültig. Der Nachrichtenschlüssel gab an, welche Kanäle, beziehungsweise Einschübe für die Streifen nicht genutzt werden sollten. Dies sollte die Sicherheit der Verschlüsselung erhöhen, da die Schlüssellänge somit variabel war. Des Weiteren enthielt jede Nachricht zwei Indikatoren, einen Systemindikator und einen Nachrichtenindikator. Der Systemindikator bestand aus fünf Buchstaben und wurde an Anfang und Ende einer Nachricht angehängt. Dieser gab an, welche Gruppe Streifen verwendet werden sollte. Auf jeder Tabelle, in der die Tagesschlüssel aufgelistet waren, war ebenfalls der Systemindikator vermerkt. Der Nachrichtenindikator bestand aus zwei Teilen, einem internen und einem externen Indikator. Der externe Indikator enthielt ebenfalls fünf Buchstaben und wurde direkt nach das erste und vor das zweite Vorkommen des Systemindikator gehängt. Der externe Indikator war für jede Nachricht anders. Der interne Indikator hängt von dem externen Indikator ab. Er besteht aus einer bis fünf Zahlen und gibt an, welche Kanäle nicht genutzt werden. Allerdings wurde der interne Nachrichtenindikator niemals übertragen. [Ass96] Da leider nicht alle Informationen über die M-138 bekannt sind, zum Beispiel ist nicht bekannt, wie viele Streifen es gab und wie diese aussahen, wird in dieser Arbeit eine fiktive Version der M-138 genutzt. Diese wurde von Klaus Schmeh zur Verwendung in Bezug auf seine Challenges erstellt. [Sch] Diese Version hat 100 verschiedene Streifen, von denen 25 in das Werkzeug eingeschoben werden. Der Schlüssel besteht aus den 25 benutzten Streifen und dem verwendeten Offset. Zusätzliche Erschwernisse wie Indikatoren oder die Nichtbenutzung bestimmter Kanäle werden in dieser Version des Werkzeuges nicht berücksichtigt. Für die in dieser Arbeit analysierte Version der M-138 existieren 100 verschiedene Streifen. Es ist nicht sichergestellt, dass für die originale M-138 ebenfalls 100 Streifen existierten.
Um eine Nachricht zu verschlüsseln, bildet der Sender zuerst einen Schlüssel. Dieser besteht aus den verwendeten Streifen sowie dem verwendeten Offset. Der Offset entspricht der Anzahl von Spalten, die der Benutzer den Metallstab zum Ablesen nach rechts verschiebt. Zuerst wählt der Sender 25 der 100 Streifen aus, die verwendet werden sollen. Für diese wird eine Reihenfolge festgelegt. Nun beginnt der Sender, die Streifen nacheinander so in das Werkzeug einzuschieben, dass der jeweils zu verschlüsselnde Buchstabe am linken Rand des Rahmens abzulesen ist. Sobald alle 25 Streifen eingeschoben sind, kann der Sender beginnen, den verschlüsselten Text abzulesen. Dies geschieht einfach, indem er am oberen Rand die Spalte, deren Offset er ausgewählt hat, sucht und nun in dieser Spalte von oben nach unten den verschlüsselten Text ablesen kann. Hierfür ist es wichtig, dass das Alphabet zweimal auf jedem Streifen vorhanden ist. Wäre dies nicht der Fall, wäre nicht sichergestellt, dass jeder Buchstabe verschlüsselt werden kann. In diesem Fall könnte es passieren, dass der Streifen in der als Offset verwendeten Spalte bereits zu Ende wäre. Sobald die ersten 25 Buchstaben verschlüsselt sind, werden die Streifen verschoben, sodass die nächsten 25 Buchstaben am linken Rand des Rahmens stehen. Dies wird solange wiederholt, bis der komplette Text verschlüsselt ist. Dabei wird jedoch nie die Reihenfolge der Streifen verändert. Die M-138 ist also eine periodische, polyalphabetische Verschlüsselung, deren Schlüssellänge 25

ist. Der Schlüssel, in dem die Streifen 1, 2, 3,...,25 und der Offset 7 verwendet werden, könnte zum Beispiel so dargestellt werden:

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

3.3 Schlüsselraumgröße

Die Schlüsselraumgröße gibt an, wie viele mögliche Schlüssel es in einem Chiffriersystem gibt. Die Größe des Schlüsselraums der M-138 ist bestimmt durch:

- Die Anzahl der Streifen, die im System verfügbar sind. Aufgrund des Prinzip von Kerkhoff wird davon ausgegangen, dass die Streifen dem Feind bekannt sind. Für die Berechnung des Schlüsselraums ist es also nur wichtig, wie viele Streifen existieren, nicht, wie viele Möglichkeiten es gibt, diese Streifen zu beschriften. Dadurch würde der Schlüsselraum deutlich größer werden.
- Die Anzahl der Streifen, die im Werkzeug verwendet werden.
- Die Anzahl der möglichen Offsets, die der Anzahl der Buchstaben des Alphabets der verwendeten Sprache entspricht.

Die Anzahl der zur Verfügung stehenden Streifen beträgt 100. Aus diesen 100 Streifen werden zufällig 25 ausgewählt. Es gibt also $100 \cdot 99 \cdot 98 \cdot ... \cdot 76 \cdot 75$ Möglichkeiten, die Streifen auszuwählen. Kurz kann man dies als $\frac{100!}{75!}$ Möglichkeiten schreiben. Nun gibt es noch die Möglichkeit, den Offset zu wählen. Ein Offset von 0 ergibt keinen Sinn, da das Werkzeug in diesem Fall den Klartext zu sich selbst verschlüsseln würde. Gehen wir davon aus, dass die deutsche Sprache ohne Umlaute oder die englische Sprache benutzt werden. Diese Alphabete bestehen aus 26 Buchstaben, da die Gross- und Kleinschreibung bei der M-138 ignoriert wird. Der Offset 26 ist mit derselben Begründung wie der Offset 0 zu vernachlässigen. Zu beachtende Offsets wären also 1,2,3,...,24,25. Für den kompletten Schlüsselraum ergibt sich somit folgende Berechnung:

$$\frac{100!}{75!} \cdot 25 \approx 3,76 \cdot 10^{14} \cdot 25 \approx 2^{161,3} \cdot 25 \approx 2^{166}$$

Es ist anzunehmen, dass der Schlüsselraum der originalen M-138 durchaus größer gewesen sein kann, da eventuell mehr Streifen im System verfügbar waren und es sogar mehrere verschiedene Gruppen von Streifen gab. Dies ist jedoch nicht mit Sicherheit bekannt. Aber auch wenn man den errechneten Schlüsselraum zu Grunde legt, sieht man, dass dieser auch im Vergleich zu modernen Verschlüsselungen relativ groß ist. Zum Vergleich kann man die Schlüsselraumgrösse des Triple-DES verwenden, der in den USA vom NIST für die behördliche Nutzung vorgeschlagen wird. [Nat12] Diese entspricht circa 2^{168} für einen Brute-Force Angriff und 2^{112} im Fall eines Known-Plaintext Angriffs. [Wac14] Dies verdeutlicht, dass die M-138

einen auch für heutige Verhältnisse noch sehr großen Schlüsselraum hat und es definitiv nicht möglich war, die Verschlüsselung durch erraten des Schlüssels zu brechen.

3.4 Unizitätslänge

Die Unizitätslänge bezeichnet in der Kryptographie die Länge, die ein verschlüsselter Text haben muss, sodass der dazugehörige Klartext eindeutig ist. Ist der Text kürzer als die Unizitätslänge der verwendeten Verschlüsselung, so ist es möglich, mit Hilfe verschiedener Schlüssel mehrere sinnvolle Klartexte zu konstruieren. Die Unizitätslänge ist nützlich, um zu sehen, ob es sinnvoll ist, auf einen bekannten verschlüsselten Text eine Ciphertext-Only Attacke anzuwenden. Berechnet wird die Unizitätslänge U mit Hilfe der Entropie des Schlüsselraums $H(k)$ und die von der Sprache abhängige Redundanz eines Buchstaben D. Diese beträgt für die englische Sprache $3,2$. [Wac14] Somit ergibt sich folgende Formel für die Unizitätslänge:

$$U = \frac{H(k)}{D} = \frac{166}{3,2} \approx 52,2$$

Für die M-138 bedeutet dies, dass ein Ciphertext-Only Angriff unabhängig von seiner Effizienz nur dann sinnvoll sein kann, wenn die Länge des bekannten verschlüsselten Textes größer als 52 ist, da bei kürzeren Texten mehrere Schlüssel existieren, die den bekannten Text auf verschiedene sinnvolle Klartexte entschlüsseln können.

4 Kryptoanalyse

Dieses Kapitel beschäftigt sich mit der Kryptoanalyse der M-138. Wie im vorigen Kapitel bereits festgestellt wurde, ist der Schlüsselraum der M-138 relativ groß, wodurch sie im ersten Augenblick als sicher erscheinen kann. Dennoch gibt es Wege, die Verschlüsselung zu brechen. Auf einige Möglichkeiten, die Verschlüsselung anzugreifen, wird im Folgenden eingegangen.

4.1 Schwächen

Die wohl gravierendste Schwachstelle der M-138 ist, dass sie mit Hilfe heuristischer Verfahren angreifbar ist. Dies ist ein typisches Problem für klassische Verschlüsselungen. Dies bedeutet, dass ein teilweise korrekter Schlüssel, angewandt auf einen Geheimtext, beim Entschlüsseln auch einen teilweise korrekten Klartext erzeugt. Nutzt man einen festgelegten Klartext, so hat eine kleine Veränderung an dem genutzten Schlüssel nur eine kleine Änderung an dem generierten Geheimtext zur Folge. Dies führt dazu, dass Angriffe, die mit Kostenfunktionen arbeiten, möglich sind. Dazu gehört zum Beispiel das Hill-Climbing, was später noch genauer erklärt wird.

Eine weitere Schwäche der M-138 ist, dass die Verschlüsselung zwar polyalphabetisch, aber periodisch ist. Dadurch, dass bei der M-138 jeder 25. Buchstabe mit demselben Streifen verschlüsselt wird und sich auch der Offset innerhalb einer verschlüsselten Nachricht nicht ändert, ergibt sich alle 25 Stellen eine monoalphabetische Verschlüsselung. Dadurch kann man einen bekannten Geheimtext mit Angriffen, die sich für monoalphabetische Verschlüsselungen eignen, wie zum Beispiel der Häufigkeitsanalyse der Buchstaben, angreifen, wenn der bekannte Geheimtext lang genug ist. Außerdem ermöglicht dies einen gut funktionierenden Known-Plaintext Angriff, wenn der bekannte Textabschnitt lang genug ist.
Eine Gemeinsamkeit mit der deutschen Enigma ist, dass sowohl die M-138 als auch die Enigma in einer Verschlüsselung einen Klartextbuchstaben nicht auf sich selbst abbilden können. Während dies bei der Enigma technische Gründe hatte liegt dies bei der M-138 daran, dass das Abbilden eines Buchstabens auf sich selbst nur unter Verwendung des Offsets 0 möglich ist, welcher nicht sinnvoll einsetzbar ist, da in diesem Fall der komplette Klartext auf sich selbst abgebildet würde, faktisch also keine Verschlüsselung vorhanden wäre.

4.2 Angriffe

Was für ein Angriff auf die M-138 sinnvoll ist, hängt immer davon ab, welche Informationen dem Angreifer bekannt sind. Am einfachsten ist ein Angriff, wenn dem Angreifer ein Paar von Klartext und Geheimtext bekannt ist und das Ziel ist, den verwendeten Schlüssel zu erhalten. Je weniger Informationen bekannt sind, desto schwieriger wird es für den Angreifer, einen erfolgreichen Angriff zu konstruieren.

4.2.1 Known-Plaintext Angriff

Der Known-Plaintext Angriff auf die M-138 nutzt aus, dass es sich um eine periodische, polyalphabetische Verschlüsselung handelt. Dies führt dazu, dass alle 25 Stellen im Text derselbe Streifen zum Verschlüsseln genutzt wird. Dies liegt an dem Design der Maschine. Da in den Metallrahmen nur 25 Streifen eingeschoben werden können, wiederholt sich der Schlüssel alle 25 Stellen. Der Angriff lässt sich in mehrere Schritte unterteilen.
Da man prüfen muss, mit welchen Streifen sich der bekannte Klartext auf den Geheimtext abbilden lässt, muss man diesen Angriff für jeden Offset einmal ausführen. Die folgenden Schritte werden also jeweils 25 mal für die Offsets 1 bis 25 ausgeführt. Zuerst prüft man für jede Stelle des Textes, durch welche Streifen sich der Klartextbuchstabe an dieser Stelle auf den Geheimtextbuchstaben abbilden lässt. Dazu sucht man auf jedem Streifen den Klartextbuchstaben und prüft, welcher Buchstabe mit dem aktuell analysiertem Offset der entsprechende Geheimtextbuchstabe wäre. Stimmt dieser mit dem im Geheimtext stehenden Buchstaben überein, so merkt man sich den geprüften Streifen als einen möglichen Streifen für diese Position. Dies führt man für den gesamten Text aus. Somit erhält man für jede Stelle eine Liste mit möglichen Streifen. Am Ende prüft man, ob es eine Stelle gibt, an der kein Streifen existiert, der den Klartextbuchstaben auf den Geheimtextbuchstaben abbilden kann. Ist dies der Fall, so weiß man, dass der analysierte Offset kein möglicher Schlüssel ist und kann den nächsten Offset prüfen. Ist dies nicht der Fall, kann man mit Schritt zwei fortfahren.
In diesem Schritt nutzt man nun die Tatsache aus, dass sich der Schlüssel alle 25 Zeichen wiederholt. Dazu prüft man die Liste der möglichen Streifen an jeder Stelle mit der Liste der möglichen Streifen 25 Stellen später. Nehmen wir an, wir haben folgende Ausgangssituation:

Position	1	2	3	26	27	28	51	52	52 ...
Streifen	1,3,7	17	21	7	21,77	7,13	7	23	29 ...

Tabelle 4.1: Beispiel für mögliche Streifen an unterschiedlichen Positionen

Vergleicht man nun die erste Stelle mit der 26., so sieht man, dass für die erste Stelle des Schlüssels nur der Streifen 7 infrage kommt, da die Streifen 1 und 3 an

Stelle 26 nicht passen. Geht man nun weiter und schaut an Stelle 51, so sieht man, dass Streifen 7 auch hier passen würde. Die Schnittmenge der möglichen Streifen für Position 1 des Schlüssels ist somit 7. Prüft man nun die zweite Stelle, so sieht man dass an Stelle zwei zwar Streifen 17 passen würde, an Stelle 27 jedoch nur die Streifen 21 und 77. Da es keinen Streifen gibt, der an beiden Stellen passt, weiß man, dass es keinen passenden Schlüssel gibt, der mit dem aktuell getesteten Offset den Klartext in den Geheimtext übersetzt. Je länger der Text ist, umso öfers wiederholt sich der Schlüssel. Dadurch kann der mögliche Schlüssel umso stärker eingeschränkt werden, umso länger der Text ist. Am Ende erhält man, wenn man einen passenden Offset gefunden hat, für jede Stelle eine reduzierte Menge von möglichen, passenden Streifen. Noch weiter reduzieren kann man diese Menge, indem man sich zunutze macht, dass jeder Streifen innerhalb eines Schlüssels nur einmal vorkommen kann, da er nicht mehrfach in das Werkzeug eingeschoben werden kann.
Dazu geht man in Schritt drei erneut über alle 25 Stellen des Schlüssels. Sobald man eine Stelle gefunden hat, an der nur ein Streifen passt, prüft man, ob der Streifen auch an einer anderen Stelle passen würde. An allen anderen Stellen, an denen der Streifen passen würde, kann man diesen nun aus der Menge der passenden Streifen entfernen, da er an der Stelle, an der nur er passt, vorkommen muss. Dies wiederholt man so lange, bis alle Stellen, an denen nur ein Streifen passt, geprüft wurden, und durch diese Reduktion keine neuen Stellen mehr entstanden sind, an denen wiederum nur ein Streifen passt. Gibt es nun immer noch an jeder Stelle des Schlüssels mindestens einen passenden Streifen, kann man eine Menge aller möglicher Schlüssel erstellen.
Dazu muss man im letzten Schritt die Mengen der möglichen Streifen zu allen möglichen Kombinationen verknüpfen. Macht man dies für alle Offsets, so erhält man am Ende eine Menge aller möglicher Schlüssel, die den bekannten Klartext auf den bekannten Geheimtext abbilden.

4.2.2 Ciphertext-Only Angriff

Eine Ciphertext-Only Attacke wird angewandt, wenn der Angreifer nur im Besitz eines Stück Geheimtextes ist, allerdings keine Informationen über den Klartext bekannt sind. Das Ziel des Angriffs ist es, den abgefangenen Geheimtext zu entschlüsseln. Ein Nebeneffekt ist, dass man den verwendeten Schlüssel herausfindet und eventuell bei einem später abgefangenen Geheimtext nochmals verwenden kann. Es gibt mehrere verschiedene Arten von Angriffen.

4.2.2.1 Hill-Climbing

Das Hill-Climbing setzt auf zwei Charakteristiken. Einmal hat jede Sprache charakteristische Besonderheiten, zum Beispiel die Häufigkeit gewisser Buchstaben oder das Vorkommen von Buchstabenkombinationen. Dadurch kann man Texte bewerten, wie sehr sie einer gewählten Sprache ähneln. Deshalb muss man

für diesen Angriff vorher wissen oder zumindest vermuten können, in welcher Sprache der Klartext verfasst wurde. Eine andere Charakteristik, die das Hill-Climbing nutzt, ist, dass bei einem festgelegtem Geheimtext eine kleine Änderung am Schlüssel nur eine kleine Änderung des Geheimtextes zur Folge hat. Ändert man zum Beispiel beim Verschlüsseln nur einen Streifen im Schlüssel, so bleibt der grösste Teil des Geheimtextes gleich. Lediglich an der Stelle, an der der Streifen geändert wurde, ändert sich auch der Geheimtext, also $\frac{1}{25}$ des gesamten Textes. Wie der Known-Plaintext Angriff wird auch das Hill-Climbing für jeden Offset einzeln durchgeführt. In einem ersten Schritt wird ein zufälliger Schlüssel erstellt. Mit Hilfe dieses Schlüssels K wird nun der Geheimtext C entschlüsselt. Der so entstandene Klartext P wird nun von einer Kostenfunktion bewertet. Dieser Wert wird dem genutzten Schlüssel K zugeordnet und gespeichert.
In diesem Schlüssel K wird nun eine Stelle ausgewählt. Der an dieser Stelle vorkommende Streifen wird nun nacheinander mit allen anderen möglichen Streifen ausgetauscht. Die so entstehenden Schlüssel K'^* werden ebenfalls alle benutzt, um den Geheimtext zu einem Klartext P'^* zu entschlüsseln, welcher dann bewertet wird. Am Ende dieser Prozedur wird überprüft, welcher Schlüssel den besten Klartext erstellt hat. Dieser Schlüssel wird nun als neuer K angenommen und die Prozedur des Streifen tauschen wird neu gestartet. Dies wird so lange wiederholt, bis es durch Tauschen der Streifen nicht mehr möglich ist, einen besseren Schlüssel zu finden.
An dieser Stelle hat der Algorithmus ein lokales Maximum gefunden. Da nicht sichergestellt ist, dass dies auch das globale Maximum ist und der korrekte Schlüssel gefunden wurde, startet der Algorithmus mehrmals mit einem neuen zufälligen Schlüssel. Dies erhöht die Chance, dass das globale Maximum gefunden wird. Aus den so erhaltenen jeweils besten Schlüsseln kann man dann eine Bestenliste erstellen, in der man die am besten bewerteten Schlüssel speichert. Dies ist sinnvoll, da ein Mensch letztendlich besser als ein Computer erkennen kann, ob ein so erhaltener Klartext Sinn ergibt oder nicht.

4.2.2.2 Brute-Force Angriff

Mit einem Brute-Force Angriff würde man versuchen, die M-138 zu brechen, indem man für einen Geheimtext alle möglichen Schlüssel ausprobiert, um den dazugehörigen Klartext zu finden. Dies ist aufgrund der bereits errechneten Grösse des Schlüsselraums jedoch nicht sinnvoll. Die Grösse des Schlüsselraums beträgt circa 2^{167}. Gehen wir davon aus, dass bei einem sehr effizienten Brute-Force Angriff 100 Millionen Schlüssel pro Sekunde getestet werden könnten, was ein relativ hoch angesetzter Wert ist. [Joh] Um alle möglichen Schlüssel durchzuprobieren bräuchte man also

$$\frac{2^{167} Schlüssel}{100.000.000 \frac{Schlüssel}{Sekund}} = 1,87 \cdot 10^{42} Sekunden$$

Dies entspricht circa $5 * 10^{34}$ Jahren, was ungefähr $4 \cdot 10^{24}$ mal der Lebensdauer des Universums entspricht. Selbst wenn man davon ausgeht, dass der richtige Schlüssel

mit fünfzigprozentiger Wahrscheinlichkeit nach der Hälfte der Zeit gefunden wird, ist dies kein sinnvoller Angriff.

4.2.3 Partially Known-Plaintext Angriff

Dieser Angriff kombiniert den Known-Plaintext Angriff mit einem Ciphertext-Only Angriff. Für diesen Angriff muss der Angreifer in Besitz eines Stück Klartext und eines längeren Stück Geheimtext sein. Das Ziel des Angriffs ist es einerseits, den kompletten Geheimtext zu entschlüsseln. Andererseits will man, wie bei den anderen Angriffen, den Schlüssel herausfinden, um dadurch eventuell noch mehr Informationen zu gewinnen. Der Angriff funktioniert in zwei Schritten.
Im ersten Schritt wird ein Known-Plaintext Angriff gestartet. Dazu versucht man herauszufinden, welcher Teil des Geheimtextes dem bekannten Klartext entspricht. Hat man keinerlei Informationen darüber, so muss man den Known-Plaintext Angriff für jede Möglichkeit ausführen. Hat man zum Beispiel einen Klartext der Länge 25 und einen Geheimtext der Länge 35, so könnte der Klartext an den Stellen $1, 2, 3, 4, ..., 10$ des Geheimtextes beginnen. Diese Fälle probiert man nun durch.
Findet man mit Hilfe des Known-Plaintext Angriffs eine Stelle, an der Geheimtext und Klartext zueinander passen, kann man versuchen, mit Hilfe der so gewonnen Informationen den kompletten Geheimtext zu entschlüsseln. Wie gut dies funktioniert hängt davon ab, wie viele mögliche Schlüssel durch den Known-Plaintext Angriff gefunden werden. Sind es unter 1000 mögliche Schlüssel, so kann man einfach jeden Schlüssel nehmen, den kompletten Geheimtext mit ihm entschlüsseln, auf den so entstanden Klartext eine Kostenfunktion anwenden und eine Bestenliste der verschiedenen Schlüssel erstellen. Schwieriger wird dies, wenn der Known-Plaintext Angriff mehr Schlüssel liefert, was umso schneller passiert, je kürzer der bekannte Klartext ist.
Eine Möglichkeit wäre, die Stellen, an denen nur ein möglicher Streifen im Schlüssel passt, festzulegen und auf die restlichen Positionen des Schlüssels einen Hill-Climbing Algorithmus laufen zu lassen. Eine andere Möglichkeit wäre es, den Hill-Climbing Algorithmus auf einem reduzierten Streifensatz, nämlich nur den im Schlüssel vorkommenden Streifen, laufen zu lassen.

5 Konzept

Dieses Kapitel behandelt die Thematik, wie man die an die Komponenten gestellten Anforderungen in CrypTool 2 umsetzen kann. Dem Benutzer sollen hierzu zwei Komponenten zur Verfügung gestellt werden. Eine, mit deren Hilfe er Texte ver- und entschlüsseln kann und eine, mit der kryptographische Angriffe auf mit der M-138 verschlüsselte Texte ausgeführt werden können.

5.1 Die M-138 in CrypTool 2

Diese Komponente soll dem Benutzer die Möglichkeit geben, einen Text auszuwählen und diesem unter Angabe eines Schlüssels zu ver- oder entschlüsseln. Dabei soll der Benutzer die Möglichkeit haben, einige Einstellungen selbst zu definieren. So soll es zum Beispiel möglich sein, die Schlüssellänge zu variieren, um auszuprobieren, wie sich dies auf die Verschlüsselung auswirkt. Der vom Benutzer gewählte Text und der zu benutzende Schlüssel können am besten mit Hilfe zweier Texteingabekomponenten zur Verfügung gestellt werden. Als Ausgabe sollte die Komponente den resultierenden Klar- oder Geheimtext liefern. Innerhalb der Komponente soll es die Möglichkeit geben, einige Einstellungen zu verändern. Schematisch kann man dies wie in 5.1 darstellen.

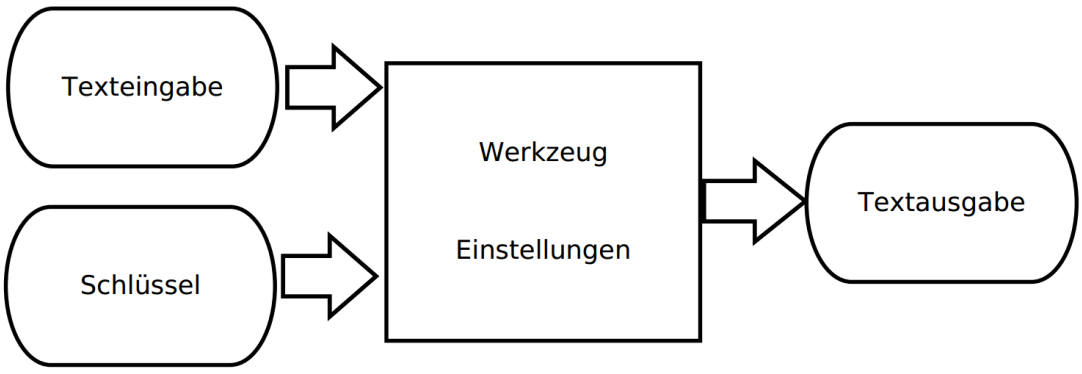

Abbildung 5.1: Schematische Darstellung der Komponente mit Ein- und Ausgängen.

Gibt der Benutzer nun einen Text und einen Schlüssel ein, soll die Komponente den Text, je nach Einstellung, ver- oder entschlüsseln und ausgeben. Hierfür soll auf die von Klaus Schmeh zur Verfügung gestellten Streifen zurückgegriffen werden. In

den Einstellungen der Komponente soll es zum Beispiel möglich sein, festzulegen, was mit nicht unterstützten Zeichen passieren soll. Da die Streifen zum Beispiel keine Umlaute beinhalten, müssen diese entweder ersetzt oder gelöscht werden. Auch die Behandlung von Groß- und Kleinschreibung sollte einstellbar sein. Standartmäßig macht es Sinn, den kompletten Text in Großbuchstaben umzuwandeln, da dies wahrscheinlich auch bei der originalen M-138 so gehandhabt wurde. Die Angabe der Schlüssellänge ist nicht nötig, da diese einfach anhand des eingegebenen Schlüssels ermittelt werden kann. Wichtig ist jedoch, dass der Benutzer die Syntax des von ihm angegebenen Schlüssels definieren kann. Dazu muss der Benutzer die Trennzeichen, die er benutzt hat, um die einzelnen Streifen des Schlüssels voneinander und vom Offset zu trennen, auswählen können, damit die Komponente später den Schlüssel richtig einlesen kann.
Zusätzlich soll eine Visualisierung implementiert werden, die dem Benutzer veranschaulicht, wie die Verschlüsselung mit der M-138 funktioniert. Diese sollte sowohl die genutzten Streifen als auch den gewählten Offset visualisieren. Dies ist realisierbar, indem man eine Tabelle anlegt, die mit den genutzten Streifen befüllt wird. Diese werden so angeordnet, dass man in einer Spalte den Eingabetext und in der um den Offset nach rechts gelegenen Spalte den Ausgabetext ablesen kann. Die Teile der Streifen, die in dem echten Werkzeug links und rechts über den Rand des Rahmens herausragen würden kann man in diesem Fall ignorieren. Zusätzlich sollten die Position des Buchstabens im Text und die Nummer des verwendeten Streifens visualisiert werden. Die Spalten können mit den entsprechenden Offsets überschrieben werden. Zur Verdeutlichung sollten die Spalten, in denen Eingabe- und Ausgabetext farblich hervorgehoben werden. Schematisch könnte dies wie in 5.2 umgesetzt werden.

Position	0	1	2	3	4	5	Streifen
1	H	E	P	Q	L	Z	17
2	A	W	B	Y	F	A	3
3	L	O	X	T	A	H	12
4	L	I	Y	V	M	L	76
5	O	E	T	V	Y	I	14

Abbildung 5.2: Schematische Darstellung der Visualisierung: Verschlüsselung des Wortes "HALLO" mit Hilfe der Streifen 17,3,12,76,14 und dem Offset 4.

Diese Visualisierung sollte ebenfalls die Möglichkeit bieten, während dem Eingeben von Text live zu verfolgen, wie dieser verschlüsselt wird. Dazu ist es nötig, dass der Eingabetext während der Ausführung der Komponente geändert werden kann. Selbiges ist auch für den Schlüssel sinnvoll, da auf diese Weise gut die Funktion

eines heuristischen Algorithmus erklärt werden kann.
Zuletzt sollte auch ein Template erstellt werden, das einen vorgegebenen Text verschlüsselt. Zum aufzeigen der Einstellungsmöglichkeiten ist es sinnvoll, zum Beispiel die Groß- und Kleinschreibung des Textes beizubehalten, da der Benutzer so, ohne einen eigenen Text eingeben zu müssen, testen kann, welche Auswirkungen es auf den Ausgabetext hat, wenn er diese Option deaktiviert.
Bei der Implementierung der Komponente sollte immer darauf geachtet werden, dass diese in Zukunft einfach anpassbar ist. So sollte es möglich sein, sowohl das verwendete Alphabet als auch die verwendeten Streifen auszutauschen. Eine andere Möglichkeit wäre es, dem Benutzer die Möglichkeit zu geben, diese Einstellungen selbst festzulegen. Daher sollten innerhalb der Algorithmen möglichst selten fest eingetragene Werte verwendet werden, sondern wenn möglich bei Einstellungen wie dem unterstützten Alphabet auf Variablen zurückgegriffen werden.

5.2 M-138 Analyse in CrypTool 2

Die zweite zu implementierende Komponente ist die Analysekomponente. Diese soll dem Benutzer die Möglichkeit geben, einen mit Hilfe der M-138 verschlüsselten Text zu analysieren und zu entschlüsseln. Hierfür muss dem Benutzer die Möglichkeit gegebenwerden, den Geheimtext und, im Fall eines Known-Plaintext Angriffes den Klartext eingeben zu können. Dies ist am einfachsten durch die Verwendung der Texteingabekomponenten möglich. Als Ausgaben soll das Programm den gefundenen Schlüssel sowie den dazugehörigen Klartext liefern. Dies kann durch ein Design wie in 5.3 realisiert werden.

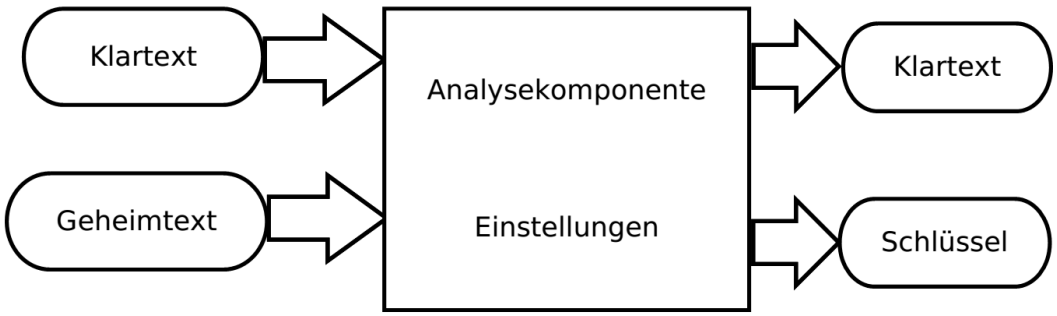

Abbildung 5.3: Schematische Darstellung der Analysekomponente zur M-138.

In den Einstellungen der Komponente muss es möglich sein, folgende Festlegungen zu treffen:

- Art des Angriffs: Soll ein Known-Plaintext, Ciphertext-Only oder Partially Known-Plaintext Angriff ausgeführt werden?

- Schlüssellänge: Die Schlüssellänge, die untersucht werden soll, sollte vom Benutzer einstallbar sein.
- Minimaler / Maximaler Offset: Der Bereich der Offsets, die analysiert werden sollen, sollte vom Benutzer definierbar sein.
- Sprache: Die verwendete Sprache des Klartext ist eine Information, die für Ciphertext-Only und Partially Known-Plaintext Angriff benötigt wird.

Die Implementierung des Known-Plaintext Angriffs soll den Klartext sowie den Geheimtext als Eingabe annehmen. Danach soll für jeden Offset geprüft werden, ob es eine Kombination von Streifen gibt, mit der der Klartext auf den Geheimtext abgebildet werden kann. Am Ende soll die Komponente für jeden Offset, für den ein Schlüssel gefunden wurde, eine Ausgabe liefern, aus der ersichtlich ist, welche Elemente der Schlüssel enthalten kann. Diese soll die Syntax

```
[Streifen an Position 1 | Anderer passender Streifen an P1],
Einzig passender Streifen an P2,
[Streifen P3 | Anderer Streifen P3 |
Noch ein anderer Streifen P3],.../Offset
```

haben. Eine mögliche Ausgabe wäre zum Beispiel

```
[1|3|5], 17,[5|91],[31|17],.../17
```

Der Ciphertext-Only Angriff soll mit Hilfe eines Hill-Climbing Algorithmus implementiert werden. In diesem Fall soll der Geheimtext als Eingabe genommen werden. Auf diesen soll dann für jeden Offset aus dem vom Benutzer gewählten Bereich der Hill-Climbing Algorithmus ausgeführt werden. Dieser soll, ausgehend von einem zufälligen Schlüssel, diesen so lange modifizieren, wie dadurch bessere neue Schlüssel entstehen. Die Ausgabe der Komponente soll aus dem besten gefundenen Schlüssel sowie dem dazugehörigen Klartext bestehen.
Für den Partially Known-Plaintext Angriff werden als Eingabe sowohl ein Klartext als auch ein Geheimtext benötigt. Der Algorithmus soll zuerst eine Menge von Schlüsseln finden, mit deren Hilfe sich der Klartext auf einen Teil des Geheimtextes abbilden lässt. Anschließend soll der Algorithmus prüfen, mit welchem dieser Schlüssel sich der komplette Geheimtext am besten entschlüsseln lässt.
Auch für diese Komponente soll eine Visualisierung erstellt werden. Diese soll zweigeteilt sein. Ein Teil soll interessante Informationen über den ausgeführten Angriff liefern. Der zweite Teil soll aus einer Liste von Schlüsseln bestehen. Die im ersten Teil gezeigten Informationen sollen Auskunft über die Startzeit und die bei gleichbleibender Rechengeschwindigkeit vermutete Terminationszeit, den zum aktuellen Zeitpunkt analysieren Offset und die pro Sekunde getesteten Schlüssel geben. Die Schlüssel im zweiten Teil sollen in Form einer Liste dargestellt werden. Für den

Ciphertext-Only Angriff ergibt es, ebenso wie für den Partially Known-Plaintext Angriff Sinn, die Liste als eine Art Bestenliste zu designen, in der die am besten bewerteten Schlüssel zusammen mit dem dazugehörigen Klartext und dem berechneten Kostenwert angezeigt werden. Fr̈ den Known-Plaintext Angriff ergibt dies keinen Sinn, da der Klartext immer gleich ist, da dieser ja bekannt ist, und auch nicht bewertet wird. Hier ist eine Möglichkeit, für jeden Offset einen Eintrag zu erzeugen und in diesem den Schlüssel auszugeben.
Um dem Benutzer den Einsteig zu erleichtern sollen drei Templates erstellt werden. Für jeden Angriff soll ein Template erstellt werden, das direkt vom Benutzer ausgeführt werden kann und einen Angriff startet, der möglichst auch erfolgreich sein soll. Während man dies für den Known-Plaintext und den Partially Known-Plaintext testen kann, so ist es nicht möglich, dies für den Ciphertext-Only Angriff vorherzusagen. Allerdings kann man durch Testen der Komponente abschätzen, wie wahrscheinlich es ist, dass für einen bestimmten Geheimtext der korrekte Schlüssel gefunden wird. Desweiteren sollen die Templates eine Erklärung zu dem jeweiligen Angriff erhalten und dem Benutzer zeigen, was er verändern kann.

6 Design und Implementierung

Dieses Kapitel erklärt die Umsetzung der Implementierung in CrypTool 2. Dabei wird vor allem auf die Funktionsweise und Implementierung der verwendeten Algorithmen eingegangen. In CrypTool 2 wurden zwei verschiedene Komponenten implementiert, die *M-138* und das *M-138 Analysewerkzeug*.

6.1 Die M-138

Diese Komponente simuliert die M-138 in CrypTool 2. Der Benutzer muss hierfür zwei Eingaben zur Verfügung stellen. Eine ist der Schlüssel, der verwendet werden soll. Dieser sollte im Format *Streifen, Streifen, Streifen,..., Streifen / Offset* eingegeben werden. Die zweite Eingabe enthält den Text, der bearbeitet werden soll. Als Ausgabe gibt die Komponente den bearbeiteten Text zurück. Des Weiteren kann der Benutzer einige Einstellungen in der Komponente vornehmen:

- Entschlüsseln / Verschlüsseln: Der Benutzer hat die Auswahl zu treffen, ob der von ihm eingegebene Text mit Hilfe des angegebenen Schlüssels verschlüsselt oder entschlüsselt werden soll.
- Streifen Trennzeichen: Mit dieser Einstellung kann der Benutzer der Komponente vermitteln, mit Hilfe welches Zeichens er in dem eingegebenen Schlüssel die verschiedenen Streifen voneinander getrennt hat. Zur Auswahl stehen *";"*, *","* und *"/"*.
- Offset Trennzeichen: Mit Hilfe dieser Einstellung kann der Benutzer angeben, womit er im Schlüssel die Liste der zu nutzenden Streifen von dem zu verwendeten Offset getrennt hat. Die Auswahlmöglichkeiten sind dieselben wie in der Einstellung *Streifen Trennzeichen*. Für *Streifen Trennzeichen* sollte niemals die gleiche Einstellung verwendet werden wie für *Offset Trennzeichen*, da die Komponente sonst nicht erkennt, ob ein Teil des Schlüssels zu den Streifen gehört oder den Offset darstellt.
- Behandlung unbekannter Zeichen: Hier kann der Benutzer auswählen, was mit Zeichen im Text, die nicht in dem Alphabet der Streifen enthalten sind, passieren soll. Zur Auswahl stehen die Optionen *"Ignorieren"*, *"Löschen"* und *"Ersetzen durch ?"*.
- Gross-/Kleinschreibung beachten: Der Benutzer hat die Möglichkeit festzulegen, ob die Gross- und Kleinschreibung des eingegebenen Textes beibehalten werden soll oder nicht.

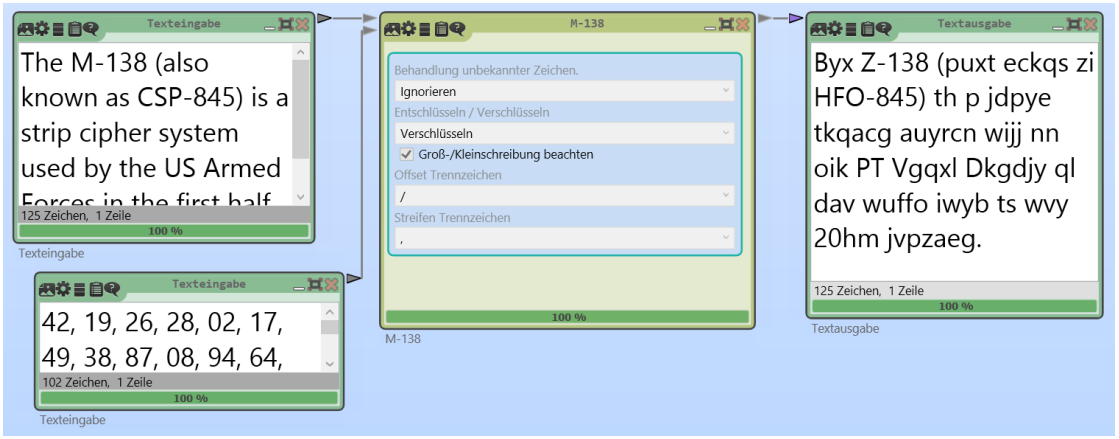

Abbildung 6.1: Beispiel für das Verschlüsseln eines Textes mit der M-138 Komponente. Auf der linken Seite der Klartext und der Schlüssel als Eingabe, auf der rechten Seite der Geheimtext als Ausgabe.

Sobald der Benutzer die Komponente auswählt, werden als erstes die zu verwendeten Streifen eingelesen. Diese werden aus einer Textdatei gelesen, in der in jeder Zeile ein Streifen steht. Am Anfang jedes Streifens steht die Streifennummer, darauf folgt das zufällige Alphabet. Als Beispiel kann man Streifen 17 betrachten, der wie folgt beschriftet ist:

```
17 EFTXLBCAWHUJGVOMYRSNKDQIZP
```

Startet der Benutzer die Komponente, so prüft das Programm zuerst, ob die nötigen Informationen zur Verfügung gestellt wurden. Hat der Benutzer zum Beispiel keinen Schlüssel angegeben, so wird eine Fehlermeldung ausgegeben, dass dieser fehlt, und die Komponente wird beendet. Sind alle nötigen Informationen vorhanden, so beginnt die eigentliche Ausführung der Komponente.
Im ersten Schritt wird geprüft, wie mit unbekannten Zeichen umgegangen werden soll. Hat der Benutzer ausgewählt, dass diese gelöscht werden sollen, so wird für jedes Zeichen des Eingabetextes geprüft, ob dieses in dem Alphabet der Streifen, in diesem Fall A-Z, vorkommt. Ist dies nicht der Fall, zum Beispiel bei Leerzeichen oder Umlauten, so wird dieses Zeichen aus dem Text gelöscht. Aus dem Text *Hallo Welt!* würde somit *HalloWelt*. Hat der Benutzer eine der anderen Varianten ausgewählt, so wird eine Liste angelegt, in der diese Zeichen gespeichert werden können. Später werden die Zeichen in dieser Liste gespeichert und im Text durch Platzhalter ersetzt. Bei der Ausgabe werden diese Zeichen dann entweder wieder in den Text eingefügt oder durch "?" ersetzt. Im nächsten Schritt wird geprüft, ob der Benutzer die Erhaltung der Gross- und Kleinschreibung eingeschaltet hat. Ist dies der Fall, wird eine Liste angelegt, in der Integerwerte gespeichert werden können. Dann wird der komplette Eingabetext durchgegangen. An jeder Position des Textes wird geprüft, ob der Buchstabe oder das Zeichen an dieser Stelle gross oder klein geschrieben ist. Dafür werden die Methoden *Char.isUpper* und *Char.isLower*

verwendet. Im Fall, dass für das Zeichen eine Großschreibung erkannt wurde, wird in der Liste eine 1 gespeichert. Wurde eine Kleinschreibung erkannt, wird eine 0 gespeichert. Wurde aus irgendeinem Grund keins von beidem erkannt, wird eine 2 gespeichert. Dies ist nötig, damit im Falle eines Fehlers die Liste trotzdem genau so lang ist wie der dazugehörige Text und ein Fehler keine Auswirkung auf den Rest des Textes hat. Diese Liste wird gespeichert und erst wieder verwendet, um die Ausgabe richtig zu formatieren. Anschließend wird der gesamte Text mit Hilfe der *ToUpper()* Methode in Großbuchstaben umgewandelt. In einem nächsten Schritt wird der Text in ein Integer Array übersetzt. Dazu wird ein Array erstellt, dessen Größe der Textlänge entspricht. Für jedes Zeichen des Textes wird nun geprüft, an welcher Stelle des Alphabets es sich befindet. Diese Stelle wird in das Array geschrieben. Aus dem Text *HALLO* wird somit zum Beispiel *7,0,11,11,14*.
Im letzten Teil der Vorbereitung wird der Schlüssel eingelesen. Dazu wird auf die vom Benutzer gesetzten Streifen und Offset Trennzeichen zurückgegriffen. Zuerst wird die Benutzereingabe als String eingelesen. Dieser wird mit Hilfe des Offset Trennzeichens einmal geteilt. Der hintere Teil sollte nur aus einer Zahl bestehen, welche als Offset gespeichert wird. Der vordere Teil sollte ein String sein, in dem von Streifen Trennzeichen getrennt die zu verwendeten Streifen aufgelistet sind. Dieser String wird nun mit Hilfe des Streifen Trennzeichens erneut geteilt und die so erhaltenen Zahlen werden nacheinander als Integer in einer Liste gespeichert.
Nachdem alle Vorbereitungen getroffen sind, beginnt die Komponente, den eingegebenen Text je nach Wahl des Benutzers zu ver- oder entschlüsseln. Dazu werden zuerst die Streifen, die verwendet werden, mit derselben Methode wie vorher der Text in Integerarrys umgewandelt. Diese werden in einer Liste abgespeichert. Ebenso wird ein neues Array von Integern erstellt. Dies hat dieselbe Größe wie der zu verschlüsselnde Text und dient zum Speichern des resultierenden Textes. Danach wird mit Hilfe einer For-Schleife über den eingegebenen Text iteriert. In dieser Schleife wird jeder Buchstabe einzeln verschlüsselt. Dazu wird für jeden Buchstaben zuerst geprüft, welcher Streifen an dieser Stelle genutzt wird. Dies funktioniert, indem man die Stelle des Buchstabens im Text modulo der Anzahl der verwendeten Streifen rechnet.
Ist der passende Streifen ausgewählt, so wird nun gesucht, an welcher Stelle des Streifens sich der zu ver- oder entschlüsselnde Buchstabe befindet. Dies funktioniert mit Hilfe der *Array.IndexOf* Funktion. Im nächsten Schritt muss herausgefunden werden, auf welchen Buchstaben dieser abgebildet werden muss. Dazu muss man beim Verschlüsseln den Buchstaben wählen, den man erhält, wenn man auf dem Streifen um den Wert des im Schlüssel gewählten Offsets nach rechts geht. Zum Entschlüsseln muss man um den Wert des Offsets nach links gehen. Dazu merkt man sich den Wert, an welcher Stelle des Streifens der Buchstabe steht, und addiert den Offset zu diesem Wert, beziehungsweise subtrahiert den Offset. An dieser Stelle muss darauf geachtet werden, dass es nicht passiert, dass daraus ein Wert größer 25 oder kleiner 0 resultiert. In der originalen Maschine wurde dies durch das zweifache Aufdrucken des Alphabets auf die Streifen gelöst. In der Implementierung wird das Problem gelöst, indem das Ergebnis zuerst mit 26 addiert und danach modulo 26 gerechnet wird. Somit ist sichergestellt, dass das Ergebnis

eine Zahl zwischen 0 und 25 ist. Nun wird der Buchstabe, der an der errechneten Stelle des aktuell genutzten Streifens steht, genommen, und in dem Array, dass den Ausgabetext beinhaltet, gespeichert.
Ist die Schleife einmal über den kompletten Text gelaufen, muss der so erhaltene Ausgabetext noch von den gespeicherten Integerzahlen zu Buchstaben übersetzt werden. Dies funktioniert wie schon das Übersetzen von Text in Zahlen. Der so erhaltene ver- oder entschlüsselte Text wird nun von der Komponente ausgegeben. Zusätzlich hat die Komponente noch eine Visualisierung, die dem Benutzer zeigt, wie das Verschlüsseln seines Textes in der M-138 aussehen und funktionieren würde. Dazu wird eine Tabelle angezeigt, in der die in die M-138 eingeschobenen Streifen angezeigt werden.

Row	Strip	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	42	T	O	V	H	R	D	B	F	U	C	G	J	X	S	Y	L	M	P	K	Q	I	Z	W	E	A	N
2	19	H	K	S	Z	P	C	Y	A	G	D	U	T	F	I	X	M	B	W	L	O	J	Q	V	N	E	R
3	26	E	W	V	B	Y	Q	X	T	U	R	G	A	N	Z	I	F	P	C	L	O	K	M	H	S	J	D
4	28	M	O	E	Q	J	T	Z	F	Y	S	I	N	A	U	B	W	C	V	D	X	R	K	H	P	L	G
5	2	A	O	V	B	L	D	P	Y	G	S	K	C	J	E	T	R	M	U	W	N	H	Q	X	I	Z	F
6	17	L	B	C	A	W	H	U	J	G	V	O	M	Y	R	S	N	K	D	Q	I	Z	P	E	F	T	X
7	49	S	W	A	G	E	Q	X	T	H	Z	C	F	Y	L	B	D	J	P	K	M	N	U	I	R	O	V
8	38	O	D	X	L	U	V	T	S	Y	H	M	F	G	A	J	N	W	P	Z	Q	B	C	I	K	R	E
9	87	K	M	U	W	F	B	E	T	C	V	S	I	J	N	A	X	H	Z	L	P	O	Q	Y	G	R	D
10	8	N	T	J	Q	R	A	C	Z	L	S	O	G	E	D	P	Y	W	F	X	H	B	V	I	M	U	K
11	94	O	J	N	H	L	S	K	V	U	B	I	D	M	X	A	E	Q	Z	T	G	W	F	Y	C	R	P
12	64	W	K	U	E	F	M	Q	I	A	C	B	Z	G	T	P	N	S	J	Y	D	V	L	X	R	H	O
13	92	N	Q	L	Y	A	D	S	O	V	G	F	U	P	H	K	J	R	Z	E	W	M	C	B	X	I	T
14	88	A	K	Y	T	N	Q	Z	G	X	J	P	L	V	B	R	C	I	U	D	S	H	E	M	W	O	F
15	37	S	N	L	R	G	F	I	T	O	P	A	J	W	H	E	D	Z	K	M	C	V	U	X	B	Y	Q
16	63	C	T	J	F	P	R	H	K	U	W	Q	O	M	X	N	I	B	L	Y	V	D	Z	E	A	G	S
17	39	S	H	Z	A	C	T	F	N	P	I	X	M	Q	R	D	U	B	Y	K	V	J	O	G	W	L	E

Abbildung 6.2: Visualisierung der Verschlüsselung eines Textes mit der M-138.

Wie in 6.3 zu sehen ist, hat die Tabelle eine feste Kopfzeile, in der die den Spalten entsprechenden Offsets angezeigt werden. In der linken Spalte wird die Reihe, die der Position des verschlüsselten Buchstabens im Text entspricht, angezeigt. In der zweiten Spalte wird die Nummer des für diese Position verwendeten Streifens angezeigt. In den folgenden Streifen werden die auf den Streifen aufgedruckten Buchstaben dargestellt. In der mit "0"überschriebenen Spalte wird der Klartextbuchstabe eingetragen. In den darauffolgenden Spalten werden dann die auf den Streifen nach diesem vorkommenden Buchstaben dargestellt. Zur besseren Visualisierung sind die Spalte "0", in der der Klartext steht, und die Spalte, in der der Geheimtext abgelesen werden kann, eingefärbt. Die Visualisierung beim Entschlüsseln eines Textes sieht entsprechend gleich aus.

6.2 Das M-138 Analysewerkzeug

Die zweite implementierte Komponente ermöglicht verschiedene kryptoanalytische Angriffe auf beliebige mit der M-138 verschlüsselten Texte. Der Benutzer kann zwischen einem Known-Plaintext, Partially Known-Plaintext und einem Ciphertext-Only Angriff wählen. In allen Fällen muss der Benutzer einen Geheimtext bereitstellen. Hat er einen Known-Plaintext oder Partially Known-Plaintext Angriff gewählt, so muss er auch den dazugehörigen Klartext angeben. Neben der Art des Angriffe hat der Benutzer die Möglichkeit, mehrere Einstellungen anzupassen:

- Schlüssellänge: Die Komponente bietet die Möglichkeit, einen Text auch mit anderen Schlüssellängen zu analysieren. Dies kann sinnvoll sein, da, wie bereits in Kapitel 3 erwähnt, zum Beispiel die M-138A mit einem Schlüssel der Länge 30 arbeitete.
- Minimaler / Maximaler Offset: Der Benutzer kann mit Hilfe dieser Optionen den möglichen Schlüsselraum eingrenzen, indem er den zu testenden Offset eingrenzt. Dies kann sinnvoll sein, wenn mit Hilfe eines Ciphertext-Only Angriffs erkannt wurde, dass für einen Offset n einige sinnvolle Worte in dem Text auftauchen, jedoch der richtige Schlüssel noch nicht gefunden wurde. In diesem Fall kann der Benutzer den Angriff nochmal laufen lassen, allerdings nur für den Offset n.
- Sprache: Diese Einstellung ist wichtig für den Ciphertext-Only Angriff, da dieser eine Kostenfunktion benutzt, die den Text anhand vorkommender Buchstabenkombinationen bewertet. Da diese Bewertung von der Sprache des Klartextes abhängt muss der Benutzer diese hier angeben.
- Neustarts für Hill-Climbing: Diese Einstellung lässt den Benutzer angeben, wie oft der Hill-Climbing Algorithmus, der für den Ciphertext-Only Angriff genutzt wird, neu gestartet wird. Auf diese Einstellung wird später noch näher eingegangen. Tätigt der Benutzer keine Eingabe, wird der Wert in Abhängigkeit von der Länge des zu analysierenden Textes ausgewählt.
- Schnelle Konvergierung: Diese Einstellung ist nur für den Ciphertext-Only Angriff relevant. Sie verändert das Verhalten des Algorithmus, auf Details wird ebenfalls später eingegangen.

Lädt der Benutzer die Komponente, werden zuerst die verfügbaren Streifen eingelesen. Des Weiteren werden aus zwei Dateien die Wahrscheinlichkeiten für Tri- und Quadgramme geladen. Diese werden in drei- und vierdimensionalen Arrays gespeichert. Dabei wird für jede Kombination, in der drei oder vier Buchstaben hintereinander vorkommen können, die Wahrscheinlichkeit für dieses Vorkommen angegeben. Als Standard werden zuerst die Werte für die englische Sprache geladen, da diese standardmäßig in den Einstellungen der Komponente angegeben ist. Wird nun die Ausführung der Komponente gestartet, wird zuerst geprüft, welchen Angriff der Benutzer ausgewählt hat. Daraufhin wird, abhängig von dem gewählten Angriff, geprüft, ob alle notwendigen Eingaben vorhanden sind. Während für den

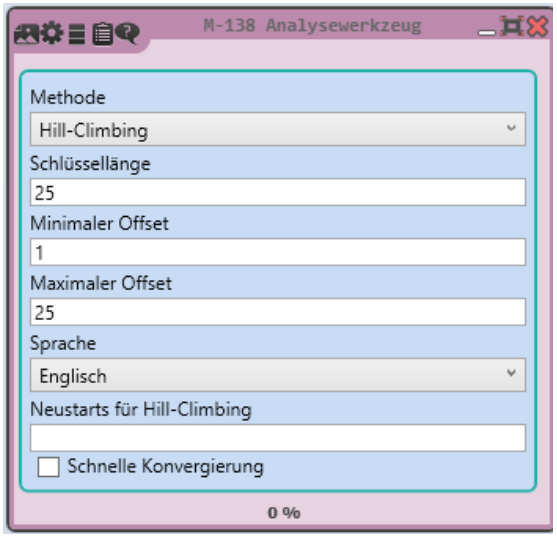

Abbildung 6.3: Das M-138 Analysewerkzeug in CrypTool 2. Auf der linken Seite zwei Eingänge, um Klartext und Geheimtext anzugeben. Auf der rechten Seite zwei Ausgänge, um den berechneten Schlüssel und den dazugehörigen Klartext auszugeben.

Ciphertext-Only Angriff nur Geheimtext angegeben werden muss, müssen für die beiden anderen Angriffe sowohl Geheimtext als auch Klartext angegeben werden. Ist dies nicht der Fall, wird der Benutzer darüber informiert und die Ausführung abgebrochen. Sind alle Informationen vorhanden, beginnt der gewählte Angriff.

6.2.1 Known-Plaintext Angriff

Wurde der Known-Plaintext Angriff gewählt, so werden zuerst der angegebene Klartext und Geheimtext in Großbuchstaben umgewandelt und unbekannte Zeichen, die nicht im Alphabet vorkommen, gelöscht. Des Weiteren wird bereits die erste Ausgabe der Komponente vorbereitet. Da der Klartext durch die Eingabe des Benutzers bereits bekannt ist, da er vom Benutzer eingegeben wurde, wird dieser direkt als durch die Entschlüsselung resultierender Klartext von der Komponente ausgegeben. Im nächsten Schritt werden Klartext und Geheimtext, wie in der Komponente M-138, in Arrays von Integerzahlen umgewandelt.
Nun wird mit Hilfe einer for-Schleife über alle möglichen Offsets iteriert. Dazu wird bei dem vom Benutzer angegebenen minimalen Offset gestartet und beim maximalen Offset gestoppt. Die folgenden Schritte wurden in eine eigene Methode ausgelagert, da dieselbe Vorgehensweise auch beim Partially Known-Planitext Angriff angewandt wird.
Für jeden Offset wird geprüft, ob es eine Kombination von Streifen gibt, die mit diesem Offset zusammen einen gültigen Schlüssel ergeben. Gültig bedeutet in diesem Zusammenhang, dass der Klartext mit Hilfe dieses Schlüssels auf den Geheimtext abgebildet wird und innerhalb des Schlüssels kein Streifen mehrmals verwen-

det wird. Dazu wird, wieder mit Hilfe einer for-Schleife, über den kompletten Text iteriert. Für jeden Buchstaben wird nun jeder vorhandene Streifen darauf getestet, ob der an dieser Stelle stehende Klartextbuchstabe mit dem aktuell analysiertem Offset auf den dazugehörigen Geheimtextbuchstaben abgebildet wird. Für jeden Buchstaben wird so eine Liste erstellt, die alle Streifen beinhaltet, die an dieser Stelle als Schlüssel infrage kommen. Diese Listen werden wiederum in einer Liste gespeichert. Sobald für eine Position kein Streifen infrage kommt, wird die Schleife abgebrochen, da es nicht mehr möglich ist, einen Schlüssel für den analysierten Offset zu finden. In diesem Fall wird mit dem Analysieren des darauffolgenden Offsets fortgefahren.
Wurde für jeden Buchstaben mindestens ein passender Streifen gefunden, existiert nun eine Liste, in der für jede Stelle des Textes eine Liste gespeichert ist, die die an dieser Stelle passenden Streifen beinhaltet. Mit Hilfe einer for-Schleife wird nun über die vom Benutzer eingestellte Länge des Schlüssels iteriert. Bisher gibt es für jede Stelle eine Liste möglicher Streifen. Im Schlüssel allerdings werden, wenn nicht anders eingestellt, nur 25 Streifen genutzt. Das heißt, an Stelle 1 wird derselbe Streifen genutzt wie an Stelle 26. Dazu wird zuerst die Liste, die in der Liste der Listen der möglichen Streifen an der Stelle, an der der Zähler der for-Schleife gerade steht, genommen. Anschließend wird eine neue Integer Variable n auf den Wert $n = Zähler + Schlüssellänge$ gesetzt. Dies entspricht der Stelle der zweiten Liste, die ausgewählt wird. Anschließend werden die Elemente dieser zwei Listen mit Hilfe der *List.Intersect* Funktion verglichen. Alle Elemente, die in beiden Listen vorkommen, werden nun in einer neuen Liste gespeichert. Anschließend wird n erneut um die Schlüssellänge erhöht. Die soeben neu erstellte Liste wird nun wieder mit der Liste an Stelle n verglichen. Dies wird so lange fortgeführt, bis n größer ist als die Länge des Textes. An dieser Stelle wurden alle Listen überprüft, an deren Stelle die aktuell geprüfte Stelle des Schlüssels angewandt wird. Die so erhaltene Liste enthält nur noch Streifen, die an allen diesen Stellen passen. Auch hier wird die Schleife wieder abgebrochen, sobald eine Liste keine Elemente enthält, da somit für den analysierten Offset kein Schlüssel existiert. Wurde diese Schleife erfolgreich durchlaufen, so existiert nun eine Liste, deren Größe der Schlüssellänge entspricht, in der für jede Position des Schlüssels eine Liste gespeichert ist, die alle möglichen Streifen für diese Position des Schlüssels enthält. Im nächsten Schritt wird diese noch weiter reduziert, indem darauf geachtet wird, dass kein Streifen mehrfach in einem Schlüssel vorkommen darf. Dazu wird zuerst eine Boolean Variable angelegt, in der gespeichert wird, ob ein Streifen gelöscht wurde oder nicht. Per default ist diese auf *true* gesetzt. Danach wird eine while-Schleife gestartet, die läuft, solange diese Variable den Wert *true* hat. Innerhalb dieser Schleife wird zuerst der Wert dieser Variable auf *false* gesetzt. Anschließend wird eine for-Schleife gestartet. Diese iteriert über alle Elemente der Liste von Listen möglicher Streifen. Jede Liste in dieser Liste wird geprüft, ob sie nur ein Element beinhaltet. Ist dies der Fall, wird gespeichert, welcher Streifen an dieser Stelle genutzt werden muss. Mit Hilfe einer forall-Schleife wird nun über alle alle anderen Stellen des Schlüssels iteriert und der gespeicherte Streifen, falls er an dieser Stelle ebenfalls in der Liste der möglichen Streifen vorhanden ist, aus dieser gelöscht. Wird ein Streifen gelöscht,

wird zuerst geprüft, ob die Liste der möglichen Streifen an dieser Stelle keine Elemente enthält. Ist dies der Fall, wird der Algorithmus abgebrochen, da somit kein möglicher Schlüssel für diesen Offset mehr existiert. Ist dies nicht der Fall, wird die Variable, die speichert, ob ein Streifen gelöscht wurde, auf *true* gesetzt. Anschließend werden die verbleibenden Listen überprüft. Dies wird aufgrund der while-Schleife so lange wiederholt, bis kein Streifen mehr gelöscht wurde. Wurde auch diese Schleife erfolgreich abgeschlossen, so existiert nun eine Liste, in der für jede Stelle des Schlüssels eine Liste mit den an dieser Stelle passenden Streifen existiert.
Dieses Ergebnis wird von der Methode zurückgegeben, muss jetzt allerdings noch aufbereitet werden, sodass es dem Benutzer als Ausgabe angezeigt werden kann. Dazu wird ein neuer String angelegt, der befüllt wird. Dazu wird mit Hilfe einer for-Schleife über die Länge des Schlüssels iteriert. Für jeden Wert des Zählers der Schleife wird die Liste, die an dieser Stelle steht, ausgewählt. Zuerst wird geprüft, ob die Liste mehr als ein Element enthält. Ist dies nicht der Fall, wird dieses Element in den angelegten String geschrieben. Ist dies jedoch der Fall, wird zuerst ein "[" in den String geschrieben. Anschließend wird mit Hilfe der *string.Join* Methode ein String erstellt, der die Elemente der Liste enthält und zwischen diesen eine "—" als Trennzeichen einfügt. Hinter diesen String wird ein "]" eingefügt. Anschließend wird geprüft, ob das Ende des Schlüssels erreicht ist. Ist dies nicht der Fall, wird ein "," eingefügt, um eine Trennung zum nächsten Element des Schlüssels vorzunehmen. Ist das Ende des Schlüssels erreicht, wird ein "/" angehängt. Danach wird der Offset in den String geschrieben und dieser mit einem Zeilenumbruch abgeschlossen. Dieser so erhaltene String wird nun an einen String angehängt, in den alle möglichen gefundenen Schlüssel geschrieben werden. Getrennt werden diese von Zeilenumbrüchen. Als Ausgabe erhält der Benutzer nun diesen alle möglichen Schlüssel enthaltenden String.

Zusätzlich hat auch diese Komponente eine Visualisierung. Diese ist zweigeteilt. In einem Teil werden Informationen zu dem gerade laufenden Angriff dargestellt. In dem zweiten Teil wird eine Liste der Schlüssel gezeigt. Wie in

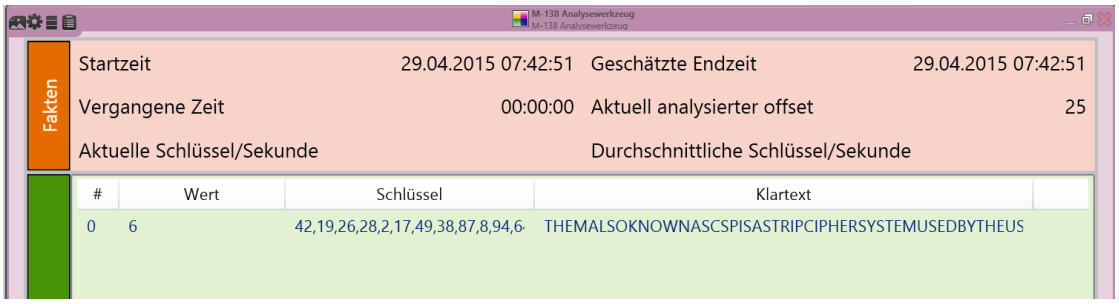

Abbildung 6.4: Das M-138 Analysewerkzeug in CrypTool 2. Auf der linken Seite zwei Eingänge, um Klartext und Geheimtext anzugeben. Auf der rechten Seite zwei Ausgänge, um den berechneten Schlüssel und den dazugehörigen Klartext auszugeben.

6.4 zu sehen ist, werden im oberen Teil der Visualisierung die Startzeit, wann der Algorithmus gestartet wurde, und die Endzeit, wann der Algorithmus terminierte, angezeigt. Während der Algorithmus läuft wird die Endzeit außerdem nach jedem geprüften Offset anhand der bereits vergangenen Zeit und den noch fehlenden, zu analysierenden Offsets, neu abgeschätzt. Weiterhin werden die seit dem Start des Algorithmus vergangene Zeit und der aktuell analysierte Offset angezeigt. Die Felder, in denen die Anzahl der Schlüssel pro Sekunde angezeigt werden, sind für den Known-Plaintext Angriff nicht relevant und werden nur im Fall eines Ciphertext-Only Angriffs verwendet.
Im zweiten, unteren Teil der Visualisierung, werden die gefundenen Schlüssel angezeigt. Dabei wird für jeden Offset, der in der Spalte *Wert* angezeigt wird, eine Zeile erstellt, in der die für diesen Offset passenden Schlüssel und der Klartext angezeigt werden.

6.2.2 Ciphertext-Only Angriff

Hat der Benutzer Ciphertext-Only als Angriffsart ausgewählt, so wird ein Hill-Climbing Algorithmus eingesetzt. Die Idee und ein Großteil des Hill-Climbings stammen von Nils Kopal, im Verlauf der Implementierung in CrypTool 2 wurden jedoch einige Anpassungen vorgenommen. Zuerst wird geprüft, ob ein Geheimtext vom Benutzer angegeben wurde. Ist dies der Fall, wird die ausgewählte Sprache des Klartextes ausgewählt. Abhängig von dieser Auswahl und der Länge des Textes werden die Gewichtungen der Tri- und Quadgramme der Kostenfunktion sowie die Anzahl der Neustarts, die der Hill-Climbing Algorithmus ausführen soll, festgelegt. Eine genauere Beschreibung der Kostenfunktion folgt später. Da für die deutsche Sprache nur eine Statistik über das Vorkommen von Quadgrammen zur Verfügung gestanden hat, werden, wenn Deutsch ausgewählt wurde, nur Quadgramme betrachtet. Für die englische Sprache werden sowohl Tri- als auch Quadgramme betrachtet. Diese Statistiken werden aus Dateien geladen, die vom Betreuer der Arbeit, Nils Kopal, zur Verfügung gestellt wurden. Des Weiteren wird je nach gewählter Sprache eine Statistik über die Häufigkeit des Vorkommens der Buchstaben des Alphabets geladen. Die hierfür verwendeten Daten stammen von Wikipedia. Wurde vom Benutzer ein Wert für die Anzahl der auszuführenden Neustarts eingestellt, so wird der vorher gewählte Wert mit diesem überschrieben. Anschließend wird, wie schon beim Known-Plaintext Angriff, mit Hilfe einer for-Schleife über die zu prüfenden Offsets iteriert. Für jeden Offset wird dann der Hill-Climbing Algorithmus gestartet. Dieser benutzt einige Variablen, über die es sinnvoll ist, einen Überblick zu haben:

- globalBestKeyCost: Der Kostenwert für den besten, bisher für diesen Offset gefundenen Schlüssel.
- globalBestKey: Der zu globalBestKeyCost gehörende Schlüssel.
- localBestKeyCost: Der Kostenwert des besten während eines Neustarts gefundenen Schlüssels.

- localBestKey: Der zu localBestKeyCost gehörende Schlüssel.
- BestCostValueOfAllKeys: Der Kostenwert des besten Schlüssels, der bisher insgesamt gefunden wurde.

Dem Algorithmus wird unter anderem die Anzahl der auszuführenden Neustarts übergeben. Der Großteil des Algorithmus läuft innerhalb einer while-Schleife. Diese Schleife läuft, solange die Anzahl der durchgeführten Neustarts kleiner als die der durchzuführenden Neustarts ist. Innerhalb der Schleife wird zuerst eine Liste von Integern angelegt, in der die Nummern aller verfügbaren Streifen gespeichert werden. Als nächstes wird geprüft, ob dem Algorithmus ein Startschlüssel übergeben wurde. In der Implementierung des Ciphertext-Only Angriffes ist dies jedoch nicht der Fall, weshalb diese Möglichkeit hier vernachlässigt werden kann. Wurde kein Startschlüssel übergeben, so wird ein zufälliger Schlüssel generiert. Dieser Schlüssel hat allerdings nicht die angegebene Schlüssellänge, sondern beinhaltet jeden zur Verfügung stehenden Streifen einmal. Dieser wird befüllt, indem ein zufälliges Element der Liste aller Streifennummern genommen, in den Schlüssel geschrieben und aus der Liste gelöscht wird. Dies wird so lange gemacht, bis die Liste der Streifennummern leer ist. Der so erhaltene Schlüssel wird als *runkey* bezeichnet. Ausgehend von diesem Startschlüssel wird nun mit Hilfe zweier for-Schleifen der Schlüssel stückweise verändert. Dazu wird eine äußere Schleife genutzt, die über die Schlüssellänge läuft. Eine innere Schleife läuft über die Anzahl der verfügbaren Streifen, die der Länge des runkey entspricht. In jedem Schritt wird eine Kopie des Schlüssels, der *copykey* erstellt. Dieser stellt eine exakte Kopie des runkey dar. Dieser copykey wird nun verändert, indem zwei Elemente in ihm getauscht werden. Dazu nimmt man die Elemente, die an den Stellen der Zähler der beiden for-Schleifen stehen. Sind die beiden Werte gleich, wird der Wert übersprungen und die innere Schleife weiter erhöht. Aus diesem Schlüssel wird nun ein der Schlüssellänge entsprechendes Array erstellt, indem die ersten Elemente des copykey in ein *trimkey* genanntes Array kopiert werden, dessen Größe der Schlüssellänge entspricht. Mit Hilfe dieses trimkey wird nun der gegebene Geheimtext entschlüsselt. Der so erhaltene Klartext wird mit Hilfe einer Kostenfunktion analysiert. Die Berechnung des Kostenwertes eines Textes und somit des dazugehörigen Schlüssels gestaltet sich wie folgt:

$$costValue = \frac{TM \cdot Trigramcost + QM \cdot Quadgramcost}{TM + QM} \cdot Unigrams \qquad (6.1)$$

TM und QM sind Faktoren, die, abhängig von der Länge des Textes, die Gewichtung von Trigrammen und Quadgrammen regeln. Während bei kurzen Texten Trigramme stärker gewichtet werden als Quadgramme, ist die Gewichtung bei langen Texten umgekehrt. Für Texte mit unter 100 Zeichen werden Trigramme mit $\frac{4}{5}$ und Quadgramme mit $\frac{1}{5}$ gewichtet. Hat ein Text zwischen 100 und 199 Zeichen, so werden Tri- und Quadgramme jeweils gleichberechtigt berücksichtigt. Bei Texten mit 200 bis 299 Zeichen werden Quadgramme stärker berücksichtigt, hier ist die Verteilung $\frac{1}{3}$ für Trigramme und $\frac{2}{3}$ für Quadgramme. Bei langen Texten mit 300 und mehr Zeichen werden nun zu $\frac{1}{7}$ Trigramme und zu $\frac{6}{7}$ Quadgramme genutzt.

Die Werte Trigramcost und Quadgramcost werden von Funktionen berechnet, die jeweils den Klartext durchlaufen und für jeden Buchstaben die Wahrscheinlichkeit, dass dieser in der Reihenfolge mit den auf ihn folgenden zwei oder drei Buchstaben auftritt, aufaddieren. Der Wert Unigrams wird von einer Funktion ermittelt, die das Vorkommen von Buchstaben im Text analysiert. Diese zählt im gesamten Text, wie oft jeder Buchstabe vorkommt, und berechnet die relative Häufigkeit, mit der jeder Buchstabe vorkommt. Anschließend vergleicht sie für jeden Buchstaben die relative Häufigkeit, mit der er im Klartext vorkommt, mit der relativen Häufigkeit, mit der er in der gewählten Sprache vorkommt. Dazu nutzt die Funktion eine Tabelle, in der die Wahrscheinlichkeiten der Buchstaben im Deutschen und Englischen gespeichert sind. Der Betrag der Differenz zwischen diesen Werten wird aufaddiert. Dieser Wert wird am Ende mit drei potenziert und zurückgegeben. Der so erhaltene Kostenwert wird nun mit der Variable localBestKeyCost verglichen. Wird der Schlüssel als schlechter bewertet, so passiert nichts, und der runkey wird weiter verändert. Wird der Schlüssel als besser bewertet, so wird die Variable localBestKeyCost mit dem aktuellen Wert überschrieben. Außerdem wird der aktuelle Schlüssel in localBestKey gespeichert. Zusätzlich wird in der Variable *foundBetterKey* gespeichert, dass ein besserer Schlüssel gefunden wurde. An dieser Stelle ist auch die Einstellung *Schnelle Konvergierung* wichtig. Ist diese eingeschaltet, so wird an dieser Stelle der runkey durch den localBestKey ersetzt. Ist dies nicht der Fall, so wird der runkey so lange geändet, bis kein besserer Schlüssel mehr gefunden wurde. Erst, wenn alle möglichen Änderungen am runkey getestet wurden, wird der runkey durch den localBestKey ersetzt. Nun wird, ausgehend von dem neuen runkey, dieser erneut geändert. Dies wird solange durchgeführt, bis kein besserer Schlüssel mehr durch eine Änderung des runkey mehr gefunden werden kann. Der Algorithmus erkennt dies, indem er zu Beginn jedes Durchlaufes, sobald ein neuer runkey gesetzt wurde, die Variable foundBetterKey auf *false* setzt. Steht diese auch nachdem alle möglichen Änderungen an runkey getestet wurden noch auf *false*, so konnte kein besserer Schlüssel gefunden werden. In diesem Fall terminiert der Algorithmus und überprüft, ob der localBestKey einen besseren Kostenwert hat als der globalBestKey. Ist dies der Fall, so werden globalBestKey und globalBestKeyCost durch localBestKey und localBestKeyCost ersetzt. Wird ein neuer global bester Schlüssel gefunden, so wird dieser in eine Bestenliste geschrieben. Dazu wird ein neues Objekt vom Typ *ResultEntry* angelegt. In diesem werden der Schlüssel mit dem Offset, der resultierende Klartext und der Kostenwert des Schlüssels gespeichert. Dieser wird dann in einer Bestenliste gespeichert. Näheres zu dieser Liste wird im Abschnitt über die Visualisierung erläutert. Des Weiteren wird geprüft, ob der gefundene Schlüssel auch besser als alle anderen bisher gefundenen Schlüssel, auch die für andere Offsets, ist. Ist dies der Fall, wird auch die Ausgabe der Komponente erneuert. Dazu wird der mit diesem Schlüssel erzeugte Klartext als bester Klartext ausgegeben. Dies führt dazu, dass der Benutzer in der Ausgabe immer den aktuell besten gefundenen Klartext erhält. Wurde kein neuer, global bester Schlüssel gefunden, so wird der Algorithmus neu gestartet und ein neuer zufälliger Startschlüssel erzeugt. Dies geschieht, bis die Anzahl der ausgeführten Neustarts der vorgegebenen Anzahl von Neustarts entspricht. Wurden

alle Neustarts durchgeführt, so endet der Hill-Climbing Algorithmus und wird für den nächsten Offset neu gestartet. Sind alle Offsets geprüft, so wird die Ausgabe noch einmal erneuert. Dazu wird aus der Bestenliste der Schlüssel der ResultEntry mit dem besten Kostenwert genommen. Der in diesem ResultEntry gespeicherte Schlüssel wird von der Komponente als bester Schlüssel ausgegeben. Der mit diesem Schlüssel erzeugte Klartext wird von der Komponente als bester gefundener Klartext ausgegeben.
Wie schon für den Known-Plaintext Angriff gibt es auch für den Ciphertext-Only Angriff eine Visualisierung. Diese enthält neben einigen Werten zum Angriff auch eine sich fortlaufend aktualisierende Bestenliste der besten Schlüssel, die gefunden wurden.

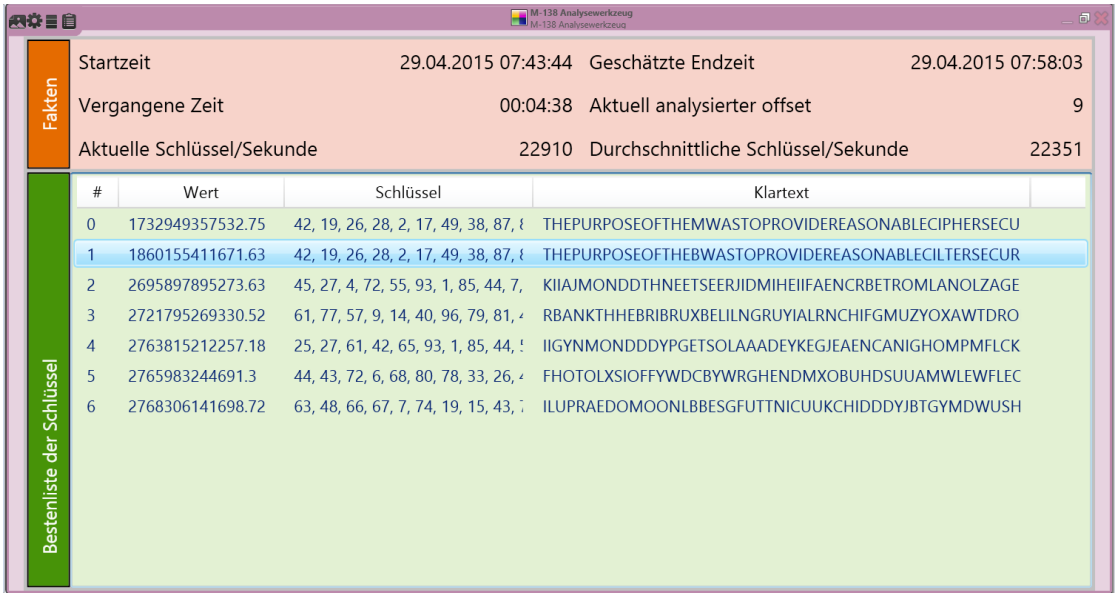

Abbildung 6.5: Visualisierung des Ciphertext-Only Angriffs mit Bestenliste der gefundenen Schlüssel.

6.2.3 Partially Known-Plaintext Angriff

Der Partially Known-Plaintext Angriff vermischt den Known-Plaintext Angriff mit dem Ciphertext-Only Angriff. Dazu wird der gegebene Geheimtext zuerst zurechtgeschnitten, sodass Geheimtext und Klartext gleich lang sind. Dazu wird das Ende des Geheimtextes entfernt und davon ausgegangen, dass der angegebene Klartext zu dem Beginn des Geheimtextes passt. Daraufhin wird auf das Paar von Klartext und beschnittenem Geheimtext der bereits bekannte Known-Plaintext Angriff angewendet. Im Gegensatz zum Known-Plaintext Angriff wird jedoch für jeden Offset nach Ausführung der Attacke geprüft, ob ein oder mehrere Schlüssel gefunden wurden. Ist dies der Fall, muss der gesamte Geheimtext geprüft werden. Wie dies geschieht, hängt von der Anzahl der gefundenen Schlüssel ab. Diese wird berechnet, indem man das Produkt der möglichen Streifen an jeder Stelle bildet.

Passen zum Beispiel an Stelle eins 5 Streifen, an Stelle zwei 3 Streifen und an Stelle drei 4 Streifen, so gibt es insgesamt $5 \cdot 3 \cdot 4 = 60$ Möglichkeiten, diese Streifen zu Schlüsseln zu kombinieren.
Werden weniger als 100.000 mögliche Schlüssel gefunden, so ist es möglich, alle Schlüssel auszuprobieren. Dafür wird mit Hilfe einer Schleife jeder mögliche Schlüssel generiert. Jeder dieser möglichen Schlüssel wird nun genutzt, um den kompletten Geheimtext zu entschlüsseln. Der so erhaltene Klartext wird mit Hilfe derselben Kostenfunktion bewertet, die auch im Hill-Climbing Algorithmus zum Einsatz kommt. Somit kann jeder Schlüssel bewertet und in einer Bestenliste gespeichert werden. Wurden alle möglichen Schlüssel getestet, werden der am besten bewertete Klartext und der dazugehörige Schlüssel von der Komponente ausgegeben.
Werden mehr als 100.000 mögliche Schlüssel gefunden, so wird zuerst geprüft, an wie vielen Stellen des Schlüssels nur ein Streifen passen kann. Sind dies zehn Stellen oder weniger, so bricht der Algorithmus ab, da für diesen Fall keine Implementierung vorgenommen wurde. Der Benutzer erhält in diesem Fall eine Warnung, dass der Angriff nicht erfolgreich ausgeführt werden konnte. Existieren jedoch mehr als zehn Stellen, an denen nur ein Streifen passt, so wird der schon im Ciphertext-Only Angriff verwendete Hill-Climbing Algorithmus genutzt. Allerdings werden in diesem Fall einige Besonderheiten eingestellt. So wird dem Algorithmus ein Startschlüssel übergeben, von dem ausgehend der Algorithmus suchen soll. Dieser Startschlüssel wird erstellt, indem an den Stellen, an denen nur ein Streifen passt, dieser genutzt wird. An den restlichen Stellen wird eine “0” in den Schlüssel geschrieben. Des Weiteren wird ein Array angelegt, das dieselbe Größe hat wie der Schlüssel. In diesem Array wird gespeichert, ob an dieser Stelle des Schlüssels feststeht, dass der im Schlüssel stehende Streifen genutzt wird, oder nicht. Dafür wird an die Stellen, an denen nur ein Streifen passt, eine 1 geschrieben, an die restlichen Stellen eine 0. Dieses Array wird ebenfalls dem Hill-Climbing Algorithmus übergeben.
In dem Algorithmus ändert sich nun unter anderem die Erstellung des Startschlüssels. Da ein Schlüssel übergeben wird, wird kein zufälliger Schlüssel generiert, sondern zuerst der übergebene Schlüssel genutzt. Der runkey wird in diesem Fall gefüllt, indem an den Stellen, an denen ein Eintrag im Schlüssel vorhanden ist, dieser auch übernommen wird. An den restlichen Stellen und vom Ende des übergebenen Schlüssels bis zum Ende des runkey werden zufällige Streifen eingesetzt.
Ist der runkey nun erstellt, werden, wie beim Ciphertext-Only Angriff, immer zwei Elemente des Schlüssels getauscht. Dabei wird hier darauf geachtet, ob eines der Elemente an einer Stelle im Schlüssel steht, für die festgelegt wurde, dass der Streifen an dieser Stelle fest ist. Ist dies der Fall, so wird das Element nicht getauscht. Der Rest des Algorithmus läuft genauso ab wie beim Ciphertext-Only Angriff. Am Ende werden auch hier der beste gefundene Klartext mit dazugehörigem Schlüssel ausgegeben
Auch der Partially Known-Plaintext Angriff hat eine Visualisierung. In dieser werden, wie auch beim Ciphertext-Only Angriff, die besten Schlüssel nach ihren Kos-

tenwerten geordnet angezeigt. Der Unterschied zum Ciphertext-Only Angriff ist lediglich, dass die Schlüssel in der Regel aufgrund der Vorgaben durch den ausgeführten Known-Plaintext Angriff sehr ähnlich sind.

7 Evaluation der Analysekomponente

Dieses Kapitel beschäftigt sich mit der Evaluation der implementierten Algorithmen. Zuletzt wird die Funktion der Algorithmen diskutiert. Dazu wird auf geprüft, ob die für die M-138 existierenden Challenges mit Hilfe der implementierten Algorithmen gelöst werden konnten.

7.0.4 Metriken

Ziel der Evaluation ist es, die Effizienz der implementierten Angriffe zu bewerten. Diese kann je nach Angriff unterschiedlich gemessen werden. Den Known-Plaintext Angriff kann man anhand seiner Laufzeit für Textpaare verschiedener Länge bewerten. Weiterhin kann evaluiert werden, wie viele Schlüssel im Schnitt für ein Textpaar einer bestimmten Länge exitieren. Erwartungsgemäß sollten weniger mögliche Schlüssel existieren, je länger das Textpaar ist, da durch die wiederholte Benutzung der Streifen mehr Möglichkeiten entstehen, den Schlüssel einzugrenzen. Merkmale für die Güte des Ciphertext-Only Angriffs sind sowohl der Aufwand, der zu betreiben ist, bevor der Algorithmus terminiert, als auch die Wahrscheinlichkeit, dass mit Hilfe des Angriffs der richtige Schlüssel gefunden wird. Anhand des Aufwands lässt sich abschätzen, wie lange der Algorithmus läuft, bis er terminiert. Dazu soll gemessen werden, wie viele Schlüssel der Hill-Climbing Algorithmus testen muss, bevor er keinen besseren Schlüssel mehr findet und terminiert. Die Erfolgswahrscheinlichkeit des Algorithmus lässt sich angeben, indem man für verschiedene Textlängen misst, wie wahrscheinlich es ist, dass der Algorithmus den korrekten Schlüssel findet. Zu erwarten ist, dass die Wahrscheinlichkeit, dass der Algorithmus den richtigen Schlüssel findet, mit der Textlänge steigt. Dies ist dadurch begründet, dass die Kostenfunktion mit steigender Textlänge mehr Daten erhält, die sie analysieren kann, und somit weniger stark von Ausreißern beeinflusst wird. Da der Partially Known-Plaintext Angriff eine Kombination der beiden Angriffe darstellt ist es bei der aktuellen Implementierung nicht sinnvoll, diesen gesondert zu analysieren.

7.0.5 Messungen

Zur Durchführung der Messungen wurden ein Testprojekt mit den in CrypTool 2 implementierten Algorithmen erstellt. Innerhalb dieses Projekts wurde eine Testumgebung aufgebaut. Hierzu wurde ein Teil des Romans *"Alice im Wunderland"* in englischer Sprache genutzt, um zufällige Testtexte zu erstellen. Mit Hilfe dieser

Texte wurden anschließend die Algorithmen getestet. Zuerst wurden sowohl der Known-Plaintext Algorithmus als auch der Ciphertext-Only Angorithmus auf ihre Laufzeit getestet. Beide wurden auf einem *Intel Core i5 Prozessor* mit 2.5GHz getestet. Die Messungen wurden jeweils 20 mal wiederholt und die durchschnittliche Laufzeit berechnet. Zum Testen der Qualität des Ciphertext-Only Angriffs wurde gemessen, wie viele Schlüssel der Hill-Climbing Algorithmus testen muss, bevor er terminiert, und wie wahrscheinlich es ist, dass er den richtigen Schlüssel findet. Dies wurde für Textlängen zwischen 125 und 320 Zeichen getestet, wobei die Textlänge in jedem Schritt um 5 erhöht wurde. Dieser Text wurde jeweils mit einem zufällig generierten Schlüssel verschlüsselt. Anschließend wurde jeweils der Hill-Climbing Algorithmus gestartet, wobei der korrekte Offset bereits übergeben wurde. Zusätzlich wurde dem Algorithmus der korrekte Schlüssel übergeben, sodass dieser terminierte, sobald der vom Algorithmus gefundene Schlüssel dem übergebenen, korrekten Schlüssel entsprach. Dies ist damit zu begründen, dass der Schlüssel, sobald er gefunden wurde, in der Bestenliste der Visualisierung sichtbar wäre und der Benutzer an dieser Stelle erkennen könnte, dass der richtige Schlüssel gefunden wurde, sodass er den Algorithmus abbrechen kann. Diese Prozedur wurde für jede Textlänge 100 mal durchgeführt. Während dieser 100 Durchläufe wurde gezählt, wie oft das Hill-Climbing den korrekten Schlüssel finden konnte. Ein Schlüssel wurde als richtig definiert, sobald der durch ihn entstandene Klartext zu mindestens 90% mit dem originalen Klartext übereinstimmte. Dabei wurde für jede Textlänge gezählt, wie viele Schlüssel der Algorithmus analysierte, bevor er abbrach. Dieser Gesamtwert wurde am Ende durch 100 geteilt, um einen Durchschnittswert zu bekommen. Für diese Messungen wurde auẅserdem Multithreading implementiert, sodass verschiedene Threads gleichzeitig verschiedene Textlängen testen konnten. Desweiteren wurden die Messungen dreimal durchgeführt. Dabei wurde der Hill-Climbing Algorithmus einmal mit 10, einmal mit 25 und einmal mit 50 Restarts aufgerufen. Leider war es aus zeitlichen Gründen nicht möglich, die Messung mit 50 Restarts zu Ende zu führen. Dennoch lässt sich die Auswirkung der Erhöhung der Restarts gut erkennen.
Für den Known-Plaintext Angriff wurde gemessen, wie viele mögliche Schlüssel es in Abhängigkeit von der Länge des gegebenen Textpaares gibt. Dies ist keine Eigenschaft des implementierten Algorithmus, sondern eine Eigenschaft des Werkzeugs, die sich allerdings mit dieser Implementierung gut messen lässt. Dazu wurde ein zufälliges Textstück aus Alice im Wunderlang genommen und mit einem zufälligen Schlüssel verschlüsselt. Anschließend wurde der Known-Plaintext Angriff gestartet, dem dieses Textpaar übergeben wurde. Zum zählen der möglichen Schlüssel wurde am Ende des Algorithmus wurde das Produkt der Anzahl der möglichen Streifen an jeder Stelle berechnet, welches der Anzahl der insgesamt möglichen Schlüssel entspricht. Die resultierenden Produkte aller Offsets wurden aufaddiert, um am Ende die insgesamt mögliche Anzahl von Schlüsseln zu erhalten. Dies wurde für jede Textlänge zwischen 25 und 500, die jeweils um 5 erhöht wurden, 500 mal durchgeführt und am Ende der Mittelwert gebildet.

7.0.6 Diskussion der Ergebnisse

Der Known-Plaintext Angriff terminierte bei allen getesteten Textlängen von 25 bis 80.000 Zeichen in unter vier Sekunden. Da sehr lange Textpaare in der Realität extrem unwahrscheinlich zu finden sind wurde es als nicht sinnvoll erachtet, dies genauer zu analysieren. Im Gegensatz dazu ergab sich für den Ciphertext-Only Angriff eine scheinbar lineare Abhängigkeit der Laufzeit von der Textlänge, wie in 7.1 zu sehen ist. Dies ist dadurch zu begrüngden, da der größte Teile der Rechenzeit zum Entschlüsseln der Texte aufgewendet wird. Während zum Ändern des Schlüssels nur zwei Elemente eines Arrays getauscht werden müssen, muss der Geheimtext für jeden Schlüssel entschlüsselt werden, damit die Kostenfunktion angewendet werden kann. Da der Aufwand zum Entschlüsseln des Textes line-

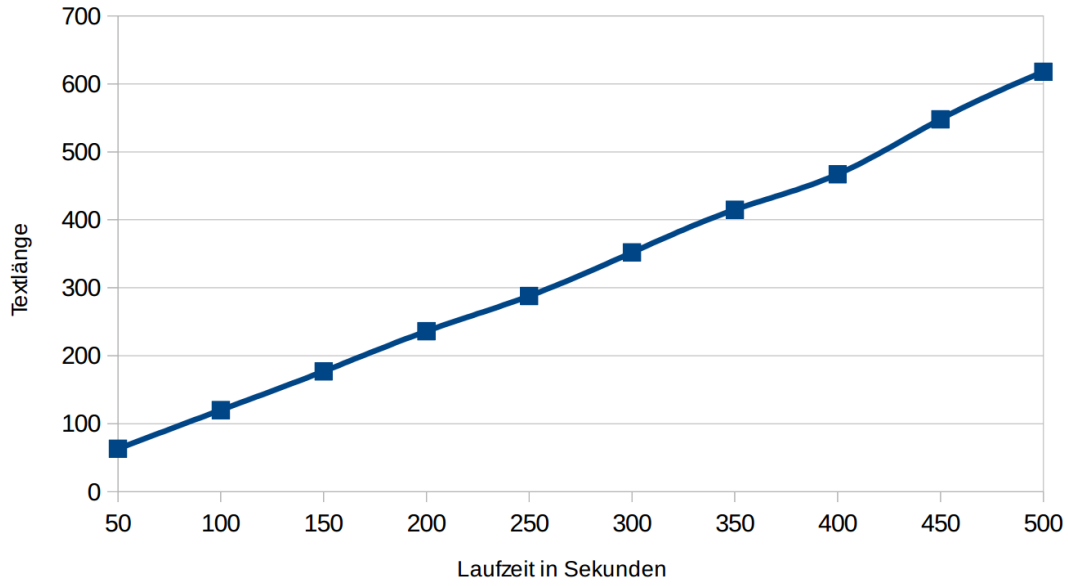

Abbildung 7.1: Laufzeit des Hill-Climbing in Abhängigkeit von der Textlänge.

ar zur Textlänge ist, ist zu vermuten, dass sich deshalb auch der Hill-Climbing Algorithmus ähnlich verhält.

Für den Ciphertext-Only Angriff bestätigt sich die Annahme, dass die Erfolgsrate des Algorithmus mit zunehmender Textlänge steigt. Desweiteren ist die Erfolgswahrscheinlichkeit höher, wenn der Algorithmus mit mehr Restarts gestartet wird. Dies ist dadurch zu erklären, dass mit mehr Restarts die Wahrscheinlichkeit steigt, dass die zufällig generierten Startschlüssel des Hill-Climbing Algorithmus gleichmäßig über den Schlüsselraum verteilt sind, was die Gefahr senkt, bei der Suche nach dem richtigen Schlüssel an lokalen Maxima hängen zu bleiben. In 7.2 ist zu sehen, dass die Erfolgswahrscheinlichkeit bei einer Textlänge von 190 Zeichen im Fall von 10 Restarts nur ca. 20% beträgt, während dieser Wert bei 25 Restarts auf ca. 40% und im Fall von 50 Restarts auf über 50% steigt. Die Wahrscheinlichkeit, mit 90% den richtigen Schlüssel zu finden, wird im Fall von 10 Restarts erst bei

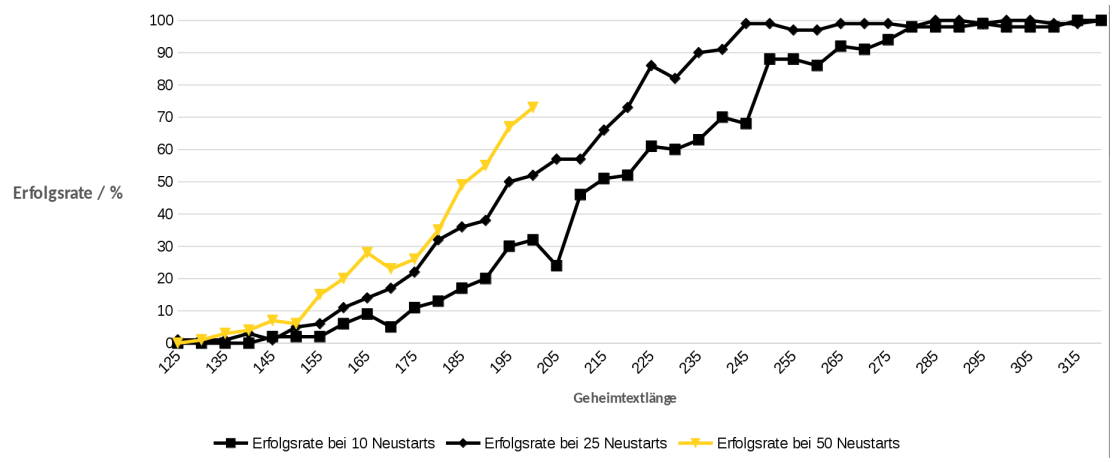

Abbildung 7.2: Erfolgsrate des Hill-Climbing Algorithmus nach Textlänge, getestet mit 10, 25 und 50 Restarts.

einer Textlänge von ca. 260 Zeichen erreicht. Bei 25 Restarts reicht hierfür schon eine Textlänge von 235 Zeichen. Aufgrund des bisherigen Verlauf der Erfolgswahrscheinlichkeit ist zu vermuten, dass bei 50 Restarts eine kürzere Textlänge nötig ist, um dies zu erreichen. Umgekehrt verhält sich die Anzahl der Schlüssel, die der Algorithmus testen muss, um den besten Schlüssel zu finden. Wie in 7.3 zu sehen

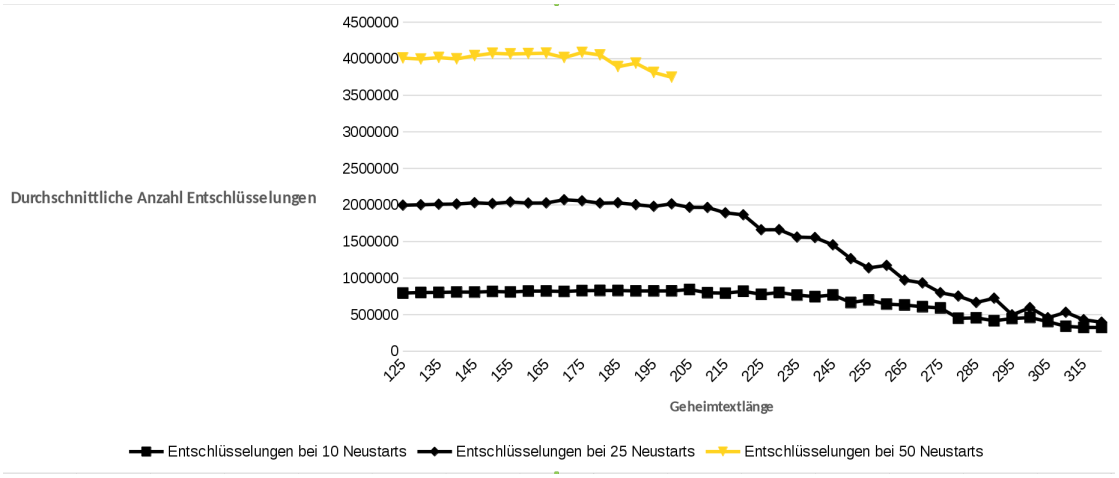

Abbildung 7.3: Aufwand des Hill-Climbing Algorithmus, um einen besten Schlüssel zu finden, in Entschlüsselungen, abhängig von der Textlänge.

ist, sinkt die Anzahl der Schlüssel, die getestet werden müssen, mit der Textlänge. Dies steht nicht im Widerspruch zu der steigenden Laufzeit des Algorithmus, da zwar weniger Entschlüsselungen getätigt werden müssen, dafür aber der Aufwand des Entschlüsselns steigt. Die sinkende Zahl der benötigten Entschlüsselungen ist mit der besseren Funktion der Kostenfunktion bei längeren Texten zu begründen. Dies entspricht auch der Beobachtung der steigenden Erfolgswahrscheinlichkeit.

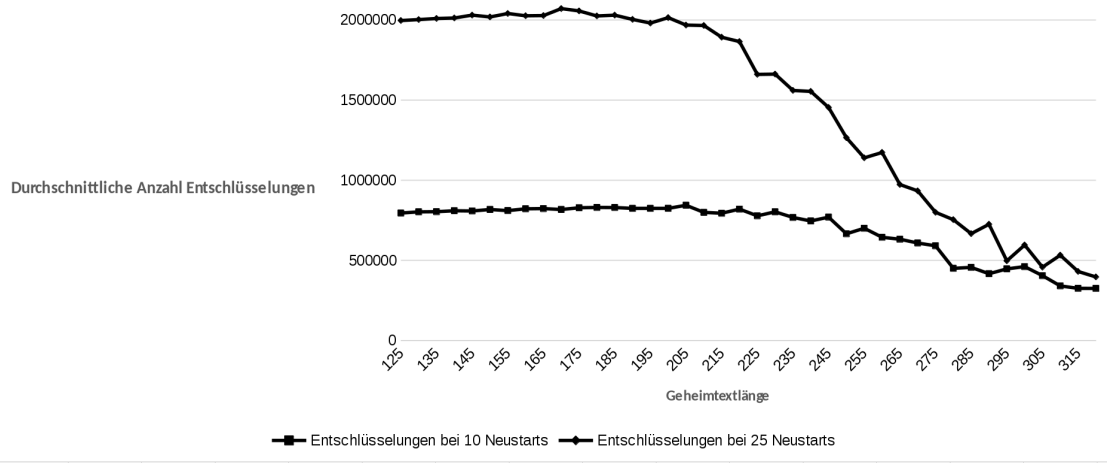

Abbildung 7.4: Aufwand des Hill-Climbing Algorithmus, um einen besten Schlüssel zu finden, in Entschlüsselungen, abhängig von der Textlänge.

Wie ich 7.4 besser zu sehen ist, sinkt die Zahl der zu testenden Schlüssel besonders im Bereich der Textlänge, in der die Erfolgswahrscheinlichkeit auf über 80% steigt. Da die Kostenfunktion bei einer größeren Textlänge mehr Daten zum Auswerten hat, findet der Algorithmus auch schneller einen besten Schlüssel. Jedoch bleibt zu beachten, dass im reellen Einsatz der genutzte Offset in der Regel nicht bekannt wäre, was für die Messungen vorausgesetzt wurde. Daher ist der Aufwand mit 25 zu Multiplizieren, da das Hill-Climbing für jeden möglichen Offset ausgeführt werden muss.
Wie erwartet existieren für sehr kurze Textpaare extrem viele Schlüssel. Da im Fall eines Textes der Länge 25 kein Streifen mehrfach benutzt wird gibt es kaum Möglichkeiten, die Streifenmenge einzuschränken. Mit steigender Textlänge sinkt die Anzahl der möglichen Schlüssel jedoch sehr schnell, sobald mehrere Streifen zweimal genutzt werden. Steigt die Textlänge auf über 50, so werden Streifen bereits dreimal genutzt und die Anzahl der möglichen Schlüssel sinkt stark, bis sie ab ca. 90 Zeichen auf 1 fällt, was bedeutet, dass es ab da möglich ist, den Schlüssel eindeutig festzustellen. Dies ist besonders für die Implementierung eines Partially Known-Plaintext Angriffs wichtig, da somit abgeschätzt werden kann, ab welcher Länge des bekannten Klartextes alle so erhaltenen möglichen Schlüssel ausprobiert werden können. In 7.5 wurden die Textlänge 25 und die Textlängen über 100 ausgelassen. Für die Textlänge 25 war der Wert mit über 300 Millionen möglicher Schlüssel zu groß, da kein Streifen doppelt benutzt wird und somit kaum eine Einschränkung möglich ist. Ab 90 Zeichen Textlänge fällt der Wert auf 1 und steigt auch nicht wieder.

Zur besseren Veranschaulichung wurden in 7.6 ebenfalls die Textlängen unter 50 ausgenommen.

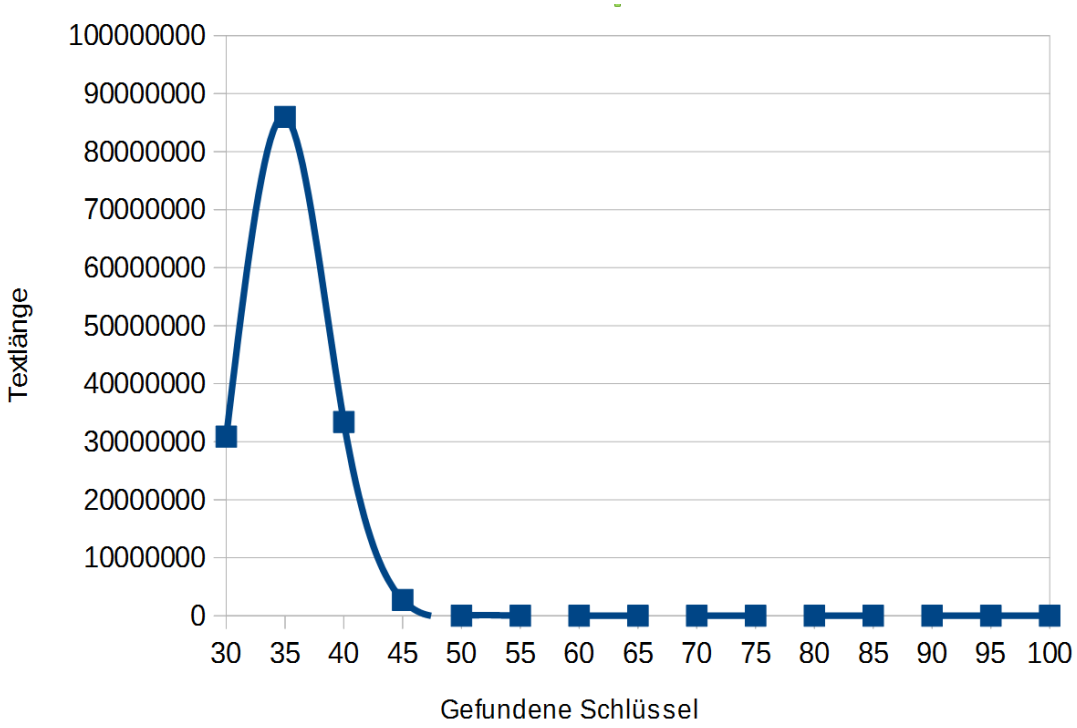

Abbildung 7.5: Anzahl der Schlüssel, die bei einem Known-Plaintext Angriff gefunden wurden, in Abhängigkeit von der Textlänge.

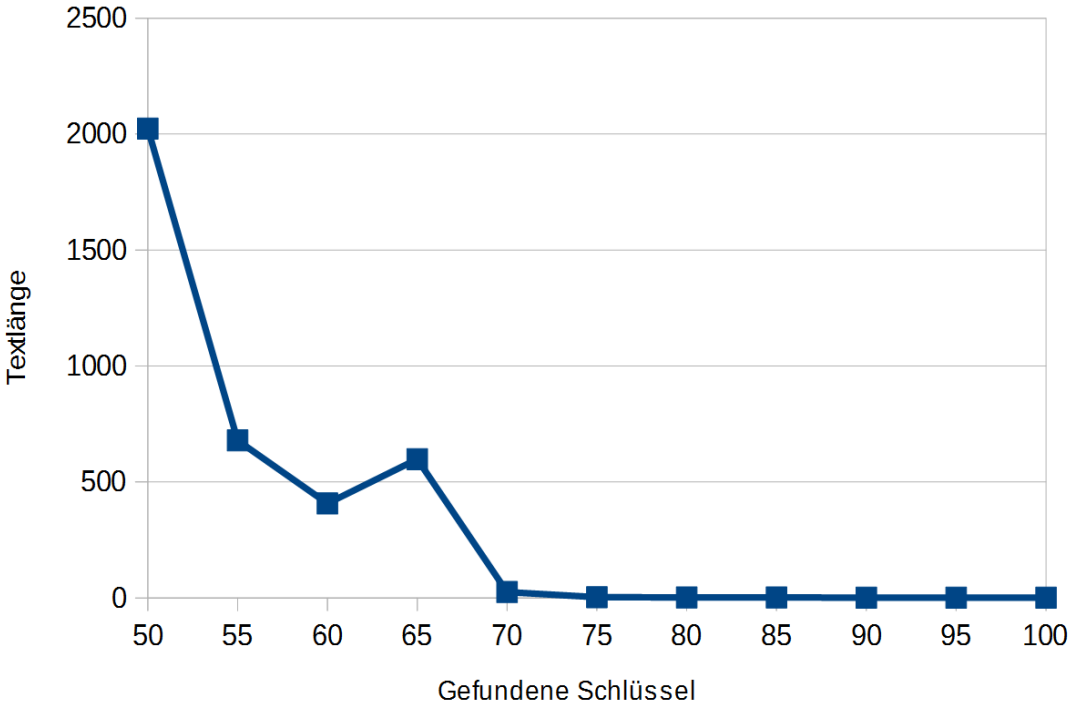

Abbildung 7.6: Anzahl der Schlüssel, die bei einem Known-Plaintext Angriff gefunden wurden, in Abhängigkeit von der Textlänge.

7.0.7 Challenges von Klaus Schmeh

Insgesamt hat der Kryptologe Klaus Schmeh sieben verschiedene Challenges zur M-138 entworfen. Während drei davon in seinem Blog [Sch] zu finden sind, sind vier Challenges auf der Mystery Twister Website [BE] veröffentlicht worden. Da die Challenges der Mystery Twister Seite nach Schwierigkeit geordnet sind, wurde zuerst versucht, diese Challenges zu lösen.

7.0.7.1 Challenge 1

Für den ersten Teil der Challenge ist ein 25 Zeichen langer Geheimtext gegeben. Des Weiteren ist der verwendete Schlüssel angegeben. Allerdings ist der Schlüssel nicht komplett bekannt, sondern nur die Streifen und die Reihenfolge, in der sie verwendet wurden. Der Offset ist nicht bekannt. Als Lösung soll der Klartext angegeben werden. Der vorgegebene Geheimtext lautet:

```
SQRXS VNMWZ PZXSX UPSXJ IYBYA
```

Der vorgegebene Schlüssel lautet:

```
46,59,09,13,08,90,30,34,62,83,51,55,49,88,27,92,69,40,74,17,21,71,45,79,96
```

Diese Challenge kann mit Hilfe der M-138 Komponente gelöst werden. Dazu übergibt man der Komponente den vorgegebenen Geheimtext als Texteingabe. Für den Schlüssel wählt man den Vorgegebenen und fügt einen beliebigen Offset hinzu. Nun stellt man die Komponente auf "Entschlüsseln" und betrachtet die Visualisierung der Komponente. In dieser kann man nun spaltenweise ablesen, welchen Text der Schlüssel unter der Verwendung des jeweiligen Offsets liefert. Betrachtet man diese Spalten genau, so sieht man, dass nur in der Spalte des Offsets 19 ein sinnvoller Text erscheint:

```
A BAD WORK MAN BLAMES HIS TOOLS
```

Die Lösung der Aufgabe ist also "A bad work man blames his tools" und war auffindbar, ohne die Analysekomponente einsetzen zu müssen.

7.0.7.2 Challenge 2

Der zweite Teil der Challenge besteht aus einem 100 Zeichen langen Geheimtext und einem 48 Zeichen langen bekannten Klartext. Es ist bekannt, dass der Klartext zu den ersten 48 Zeichen des Geheimtextes gehört. Somit ist es möglich, einen Partially Known-Plaintext anzuwenden. Als Lösung ist nach den verbleibenden 52 Zeichen des Klartextes gefragt. Dazu wählt man die M-138 Analysekomponente. Dieser übergibt man den Klartext und den Geheimtext. In den Einstellungen wählt man den Partially Known-Plaintext Angriff aus. Startet man die Komponente, so zeigt diese innerhalb weniger Sekunden eine Liste der besten gefundenen Schlüssel an. Diese unterscheiden sich nur an einzelnen Stellen voneinander und erzeugen somit alle sehr ähnliche Klartexte. Der an erster Stelle angezeigte Klartext lautete:

```
TWO THINGS ARE INFINITE
THE UNIVERSE AND HUMAN
STUPIDITY CONGRATULATION
YOU HAVE DECIPHERED
AN ALBERT EINSTEIN QUOTE
```

Somit erkennt man, dass die Lösung der Challenge *Congratulation you have deciphered an Albert Einstein quote* lautet. Diese Challenge konnte relativ einfach unter Einsatz der Analysekomponente gelöst werden.

7.0.7.3 Challenge 3

Für den dritten Teil der Challenge wird nur ein Geheimtext von 125 Zeichen angegeben. Als zusätzliche Information wird festgelegt, dass der Klartext in englischer Sprache verfasst ist. Somit muss ein Ciphertext-Only Angriff verwendet werden, um diese Challenge zu lösen. Der gegebene Geheimtext lautet:

```
RIGVR XIXRH ZQOGD QYIXV HCZKJ
LCDKU SGNDP PIBCL GPZBR UTRFJ
XHTNQ PHWXG QAXPK EEEKM DPWFK
SDTLK PTFIX IRUXN TIMTZ QQCQO
SOPFB XFMMZ PSIGZ SANJK YHWIO
```

In einem ersten Versuch wurde der Geheimtext mehrere Male dem Hill-Climbing Algorithmus übergeben. Dies führte jedoch nie zu einem verwertbaren Ergebnis. Später wurde der Algorithmus angepasst, indem nicht nur Tri- und Quadgramme analysiert wurden, sondern auch das Vorkommen einzelner Buchstaben. Danach wurde ein neuer Versuch gestartet. Nach dieser Änderung fand der Algorithmus

einige gut bewertete Schlüssel. Diese benutzten alle den Offset 6 und die dazugehörigen Klartexte begannen alle mit *"INTHEEARLY"*. Dies legte nahe, dass der gesuchte Schlüssel wirklich den Offset 6 verwendet. Daraufhin wurde das Hill-Climbing erneut gestartet, diesmal allerdings nur auf den Offset 6, dafür mit 1000 Neustarts. Dadurch wurden einige gut passende Schlüssel gefunden, der am besten bewertete Klartext lautete:

```
INTHE EARLY NINET EENTW OOCON
STHEC ABIGB ECAME THEFA FEVEN
DMEAN SOYHO MMUNI CATIO WSEED
OVERS SELYR ATEDB YTHOU INRIL
OFMIL ESTOS EEMED MIRAC TWELL
```

Somit war erkennbar, dass der Text mit *In the early nineteentwenties the cable became* beginnt. Im nächsten Schritt wurde ein Known-Plaintext Angriff gestartet, der testete, mit Hilfe welcher Schlüssel man den Geheimtext auf diesen Klartext abbilden kann. Das Ergebnis war, dass der Schlüssel bis auf eine Stelle festgelegt war:

```
51,23,15,62,14,22,39,21,99,12,19,24,4,73,6,
18,85,20,11,25,42,38,8,26,[9|59|63|87]/6
```

Lediglich an der letzten Stelle des Schlüssels musste also noch herausgefunden werden, ob Streifen 9, 59, 63 oder 87 verwendet werden muss. Dies wurde mit Hilfe der M-138 Komponente getestet. Dazu wurden der Schlüssel und der Geheimtext übergeben. Die Komponente wurde hierfür auf Ëntschlüsseln"gestellt. Durch Ausprobieren aller vier jetzt noch möglichen Schlüssel konnte schnell erkannt werden, dass Streifen 9 an der letzten Stelle genutzt werden muss. Als Klartext ergab sich somit:

```
IN THE EARLY NINETEENTWENTIES
THE CABLE BECAME THE FAVOURED
MEANS OF COMMUNICATION FOR
LOVERS SEPARATED BY THOUSANDS
OF MILES IT SEEMED MIRACULOUS
```

Das Lösen dieser Challenge nahm etwas mehr Zeit in Anspruch als das Lösen der vorhergehenden Challenges. Dies war allerdings auch erwartbar, da die Schwierigkeit der Challenges ansteigend ist. Trotzdem konnte diese gut mit Hilfe der implementierten Komponenten gelöst werden.

7.0.7.4 Challenge 4

Auch für den vierten Teil der Challenge wird nur ein Stück Geheimtext mit dem Hinweis, dass der Klartext in Englisch verfasst ist, zur Verfügung gestellt. Im Vergleich zum dritten Teil der Challenge ist dieser Geheimtext mit 75 Buchstaben jedoch deutlich kürzer. Auch zum Lösen dieser Challenge muss also wieder ein Ciphertext-Only Angriff angewandt werden. Leider war es mit der implementierten Version des Hill-Climbings nicht möglich, diese Challenge zu lösen. Wie in der Diskussion der Ergebnisse bereits dargestellt, findet der Algorithmus erst ab einer Textlänge von ca. 180 Zeichen mit 50% Wahrscheinlichkeit den genutzten Schlüssel, womit das Lösen dieser Challenge aufgrund der zu geringen Länge des Geheimtextes ohne eine Optimierung des Angriffs nicht möglich ist.

7.0.7.5 Ursprüngliche Challenges

Bevor die Challenges bei MysteryTwister veröffentlicht wurden, wurden von Klaus Schmeh bereits Ciphertext-Only Challenges mit 25, 50 und 75 Zeichen länge veröffentlicht. Während die Challenge mit 75 Zeichen Länge aus dem gleichen Grund wie Challenge 4 nicht gelöst werden konnte, können die beiden Challenges mit 50 und 25 Zeichen Länge als nicht eindeutig lösbar angenommen werden. Dies liegt daran, dass die Texte kürzer als die Unizitätslänge sind. Somit sollte es Klartexte geben, anhand derer man sich einen Schlüssel konstruieren kann, der den Geheimtext auf diesen gewählten Klartext abbildet. So lsst sich für die Challenge, bei der 25 Geheimtextbuchstaben gegeben sind, mit Hilfe des Known-Plaintext Angriffs ein Schlüssel

```
36,12,57,16,81,19,
3,10,52,34,78,95,0,
77,51,98,56,4,99,
59,63,3,60,1,84/4
```

finden, der den gegebenen Geheimtext

```
PJVYN PYIXG RPJRW YBMBA FQQPE
```

in den Klartext

```
MY BACHELOR THESIS ROCKS A LOT
```

übersetzt.

8 Ausblick und Zusammenfassung

Das Ziel dieses Kapitel ist es rückblickend, zu überprüfen, ob die gesteckten Ziele erreicht werden konnten. Vorausschauend sollen Ideen, die während der Implementierung der Komponenten entstanden, jedoch nicht mehr umgesetzt werden konnten, zusammenzufassen und mögliche zukünkftige Verbesserungen der Komponente diskutiert werden.

8.1 Zusammenfassung

In diesem Abschnitt wird auf die in der Einleitung genannten Ziele und die Überprüfung, ob diese erreicht wurden, eingegangen. Hierbei soll bewertet werden, inwiefern die an die Arbeit gestellten Anforderungen erfüllt wurden.

8.1.1 (R1) Verschlüsseln und Entschlüsseln von Texten

Das Ziel bestand darin, eine Komponente zu entwerfen, die Texte unter Angabe eines Schlüssels ver- und entschlüsseln kann. Dies ist mit der M-138 Komponente gelungen. Diese bietet die Möglichkeit, nach Angabe eines Klartextes und eines Schlüssels den dazugehörigen Geheimtext auszurechenen und auszugeben. Umgekehrt ist es auch möglich, einen Text zu entschlüsseln. Zusätzlich bietet die Komponente einige Einstellmöglichkeiten, die den Komfort erhöhen. So ist es möglich, die Groß- und Kleinschreibung eines Textes beizubehalten, und auszuwählen, wie Zeichen, die nicht zum Alphabet der verwendeten Streifen gehören, behandelt werden sollen. Die Anforderung **R1** wurde also erfüllt.

8.1.2 (R2) Visuelle Darstellung des Werkzeugs

Das zweite Ziel bestand darin, diese Komponente zu visualisieren. Dazu sollte der Rahmen der M-138 mitsamt den eingeschobenen Streifen visualisiert werden. Dies wurde umgesetzt, indem auf eine Tabelle zurückgegriffen wurde. In dieser Tabelle werden die Position des Buchstabens im Text, die Nummer des verwendeten Streifens und die auf den Streifen aufgedruckten Buchstaben dargestellt. Hierzu wird der Streifen so angepasst, dass der zu verschlüsselnde Buchstabe am linken Rand der Visualisierung der Maschine steht und die auf dem Streifen folgenden

Buchstaben nach rechts fortgeführt werden. Dabei werden nur die Buchstaben, die sich innerhalb des Rahmens befinden würden, angezeigt, die nach links und rechts möglicherweise überhängenden Teile der Streifen, wie sie im originalen Werkzeug vorkommen würden, wurden aus Gründen der Übersichtlichkeit weggelassen. Desweiteren werden die Spalten, in denen Klartext und Geheimtext ablesbar sind, eingefärbt, um das Ablesen dieser Texte zu vereinfachen. Dies simuliert den Schieber, der in dem originalen Werkzeug verwendet wurde, um den Geheimtext zu markieren. Die Anforderung **R2** wurde erfüllt, da das Ver- und Entschlüsseln für den Benutzer gut visualisiert wird.

8.1.3 (R3) Known-Plaintext Angriff auf ein Paar von Klartext und Geheimtext

Anforderung Nummer drei bestand darin, eine Komponente zu entwerfen, die einen Known-Plaintext Angriff auf vom Benutzer vorgegebenes Paar Klartext und Geheimtext durchführt. Für diese Anforderung wurde die Komponente "M-138 Analysewerkzeug" implementiert. Der Known-Plaintext Angriff ist ausgereift und findet für die Eingabe eines Klar- und eines Geheimtextes die passenden Schlüssel, die diese aufeinander abbilden können. Zusätzlich sollte die Komponente eine Visualisierung erhalten. In dieser werden in einem Teil sowohl die Startzeit als auch die bisher vergangene und die daraus berechnete vermutete Endzeit des Angriffs angezeigt Im zweiten Teil der Visualisierung wird eine Liste der gefundenen Schlüssel dargestellt. Der Known-Plaintext Angriff wird visualisiert, indem für jeden Offset, für den ein Schlüssel gefunden wurde, eine Zeile erstellt wird, in der der Offset, der Klartext und ein Ausdruck, der die möglichen Schlüssel darstellt, angezeigt werden. Anforderung **R3** wurde demnach erfüllt.

8.1.4 (R4) Ciphertext-Only Angriff auf einen Geheimtext

Zuletzt sollte ein Ciphertext-Only Angriff implementiert werden. Die wurde ebenfalls in dem Analysewerkzeug realisiert. Der implementierte Ciphertext-Only Angriff funktioniert gut, allerdings findet er erst bei Geheimtexten, die über ca. 180 Zeichen lang sind mit einer Wahrscheinlichkeit von über 50% den richtigen Schlüssel. Vorallem die Kostenfunktion, die der Angriff benutzt, lässt hier wahrscheinlich Spielraum zur Optimierung. Leider war es nicht möglich, mit Hilfe des Ciphertext-Only Angriffes alle Challenges von Klaus Schmeh zu lösen. Auch der Ciphertext-Only Angriff sollte visualisiert werden. Die implementierte Visualisierung ist, wie auch beim Known-Plaintext Angriff, in zwei Teile aufgeteilt. Diese zeigt sowohl die Startzeit als auch die bisher vergangene und die daraus berechnete vermutete Endzeit des Angriffs an. Zusätzlich wird für den Ciphertext-Only Angriff angezeigt, wie viele Schlüssel pro Sekunde im Moment und durchschnittlich getestet werden. Im zweiten Teil wird eine Liste der besten gefundenen Schlüssel angezeigt. In dieser werden live die 20 besten, gefundenen Schlüssel mit ihrem

Kostenwert und dem aus ihnen resultierenden Klartext angezeigt. Anforderung **R4** wurde ebenfalls erfüllt.

Zusätzlich zu den genannten Anforderungen wurde in der Analysekomponente ein Partially Known-Plaintext Angriff implementiert. Dieser funktioniert allerdings nur unter bestimmten Umständen. So setzt die Implementierung voraus, dass der angegebene Klartext an der ersten Stelle des angegebenen Geheimtextes beginnt. Liegt der Klartext an einer anderen Stelle des Geheimtextes, so hat der Benutzer keine Möglichkeit, dies anzugeben, der Angriff funktioniert in diesem Fall also nicht. Desweiteren wurde für den Fall, dass durch den Known-Plaintext Angriff mehr als 100.000 Schlüssel gefunden werden und an weniger als zehn Stellen des Schlüssels eindeutig ist, welcher Streifen genutzt werden muss, kein weiterführender Angriff implementiert, der Benutzer bekommt lediglich ausgegeben, dass kein Partially Known-Plaintext Angriff möglich ist, da die Anzahl der möglichen Schlüssel zu groß ist.
Somit wurden alle für die Arbeit gestellten Anforderungen erfüllt und, in Form des Partially Known-Plaintext Angriffes, ein zusätzlicher Angriff implementiert.

8.2 Ausblick

Eine sehr gute Möglichkeit, die Laufzeit der Angriffe zu verringern, ist die Parallelisierung selbiger. Die ist lediglich für den Ciphertext-Only Angriff interessant, da die anderen beiden Angriffe bereits sehr kurze Laufzeiten haben. Der Ciphertext-Only Angriff könnte relativ gut parallelisiert werden, indem man die Aufrufe des Hill-Climbing Algorithmus für verschiedene Offsets auf unterschiedliche Prozessoren aufteilt. Diese Algorithmen können alle autonom laufen und jeder eine eigene Bestenliste, in der die besten Schlüssel für diesen Offset gespeichert sind, zurückgeben. Somit müssten am Ende nur all diese Bestenlisten zu einer einzigen Liste, die die besten Schlüssel für alle Offsets enthält, zusammengefügt werden. Auf andere globale Variablen muss der Algorithmus nicht zwingend zurückgreifen, somit müssen keine Synchronisationsprobleme beachtet werden. Dadurch würde die Laufzeit des Algorithmus nahezu um den Faktor der Anzahl der eingesetzten Prozessoren gesenkt werden.
Da der Ciphertext-Only Angriff bei Texten mit unter circa 150 Zeichen noch nicht sehr zuverlässig den richtigen Schlüssel findet, wäre es sinnvoll, diesen weiter zu verbessern. Eine Möglichkeit dazu wäre die Verbesserung der verwendeten Kostenfunktion. Für kurze Texte könnte es Sinn ergeben, auch Bigramme in die Analyse mit einzubeziehen. Außerdem ist es vorstellbar, dass die Gewichtung der einzelnen Teile der Kostenfunktion noch optimierbar ist. Hierzu wäre es sinnvoll, eine Testreihe zu starten, in der man verschiedene Klartexte mit verschiedenen Varianten der Kostenfunktion analysiert und überprüft, wie sich die verschiedenen Kostenfunktionen verhalten, wenn ein natürlicher Text durch zufällige Veränderungen künstlich verfälscht wird. Eine andere Möglichkeit, den Hill-climbing Algorithmus zu verbessern, ist es, das Vertauschen der Streifen zu verbessern. Hierfür

müsste analysiert werden, ob es eventuell zu besseren Ergebnissen führt, wenn mehrere Streifen in einem Schritt getauscht werden. Dazu passend wäre es interessant, den Algorithmus um das sogenannte *Simulated Annealing* zu erweitern. Dies ist ein Versuch, zu verhindern, an lokalen Maxima hängen zu bleiben, indem man den Schlüssel nach einer gewissen Zeit an eine zufällige andere Stelle im Schlüsselraum springen lässt. Ein Ähnliches Ziel würde der Ansatz, den zufälligen Startschlüssel des Hill-Climbing Algorithmus besser zu wählen, für Angriffe, bei denen man den verwendeten Offset bereits kennt und sehr viele Neustarts auf einen Offset ausführen möchte, verfolgen. In diesem Fall wäre es sinnvoll, zu versuchen, die Startschlüssel gleichmäßig über den gesamten Schlüsselraum zu Verteilen, um die Wahrscheinlichkeit, das globale Maximum aufgrund umliegender lokaler Maxima nicht zu finden, zu reduzieren. Soll der Benutzer mehr Möglichkeiten haben, den Algorithmus zu kontrollieren, so könnte man eine Option implementieren, dass der Benutzer den Startschlüssen für den Hill-Climbing Algorithmus vorgeben kann. Dies ist auch interessant, da man diese Möglichkeit erweitern könnte, dass der Benutzer einen Teil des Schlüssels festlegen kann. Dann hätte der Benutzer die Möglichkeit, festzulegen, dass der Hill-Climbing Algorithmus nur den Rest des Schlüssels herausfinden soll. Dies wäre in Situationen, in denen der Benutzer einen Teil des Schlüssels bereits kennt, sehr sinnvoll. Um das Problem des Hill-Climbings zu umgehen, dass die Geheimtexte eine gewisse Länge haben müssen, damit der Algorithmus den richtigen Schlüssel findet, kann man versuchen, mehrere kurze Texte zu einem zusammenzufassen. Diese Idee basiert auf der Annahme, dass ein Schlüssel über einen gesamten Tag genutzt werden kann. Ist dies der Fall, kann man die Verschlüsselung angreifen, indem man das Hill-Climbing auf all diese Geheimtexte zusammen anwendet. Dadurch würde die effektive Länge des Textes, auf den man die Kostenfunktion anwendet, verlängern lassen, auch wenn keine einzelnen langen Nachrichten abgefangen werden. Ein Anderer Ansatz, den Ciphertext-Only Angriff zu verbessern, ist der, das Hill-Climbing zuerst auf einem reduzierten Streifensatz auszuführen. Durch das zufällige entfernen von 25 der Streifen würde man den Schlüsselraum deutlich resuzieren. Textbruchstücke könnten jedoch auch mit 75 statt 100 Streifen noch richtig entschlüsselt werden können. Ausgehend von so erhaltenen Teilen des Schlüssels könnte man dann den Angriff erneut starten, diesmal mit allen Streifen, jedoch auch mit schon festgelegten Teilen des Schlüssels. Um die Nutzerfreundlichkeit zu erhöhen wäre es sinnvoll, dem Benutzer die Möglichkeit zu geben, eine Schlüssel aus der Visualisierung zu kopieren. Da nur der beste Schlüssel von der Komponente ausgegeben wird ist es derzeit nicht möglich, einen anderen Schlüssel zu kopieren. Dies könnte gelöst werden, indem in der Visualisierung ein Kontextmenü erstellt wird, sodass der Benutzer durch einen Rechtsklick auf einen Schlüssel diesen kopieren kann.
Es existiert auch Potential, den Partially Known-Plaintext Angriff weiter zu verbessern. Zuerst müsste es eine Möglichkeit für den Benutzer geben, auszuwählen, an welcher Stelle des Geheimtextes der Klartext beginnt. Bisher wird davon ausgegangen, dass der Klartext den Beginn des Geheimtextes darstellt. Weiterhin sollte es eine Möglichkeit geben, nach bestimmten Wörtern im Geheimtext zu suchen. Weiß man zum Beispiel, dass ein gewisses Wort im Geheimtext vorkommt,

allerdings nicht an welcher Stelle dies vorkommt, so sollte es eine Möglichkeit geben, trotzdem einen Partially Known-Plaintext Angriff auszuführen. Dies könnte funktionieren, indem man für jede mögliche Position des Klartextes einen Known-Plaintext Angriff startet und von den so gefundenen Schlüsseln ausgehend weiter sucht. Eine Erweiterung dieses Angriffs wäre die Implementierung eines Angriffs, der ein Wörterbuch benutzt. Hierzu könnte man ein Wörterbuch als Klartexteingabe verbinden. Damit könnte man eine Partially Knwon-Plaintext Attacke mit allen Wörtern, die mindestens eine gewählte Länge haben, starten. So könnte man eine Known-Plaintext Attacke anwenden, indem man testweise das Wort an jeder Stelle des Geheimtextes starten lässt. Wird für einen Fall ein funktionierender Schlüssel gefunden, so wird dieser auf den gesamten Geheimtext angewendet. Dadurch kann es passieren, dass durch einen teilweise richtigen Schlüssel ein Teil eines anderen Wortes gefunden wird. Mit dieser neuen Erkenntnis kann man den Text nun weiter angreifen, darauf basierend, dass man immer neue Wortteile findet und somit den Schlüssel immer weiter eingrenzen kann.
Eine mögliche Einstellung, um die man den Known-Plaintex Angriff erweitern kann, ist die Option, dem Benutzer zu überlassen, ob es möglich sein soll, Streifen in einem Schlüssel mehrfach zu benutzen oder nicht. Während bisher der Schlüssel am Ende darauf überprüft wird, ob ein Streifen zweimal vorkommt und, wenn ja, der Schlüssel verworfen wird, könnte man es einstellbar machen, ob dies geprüft werden soll oder nicht.
Auch die Komponente der M-138 hat noch Optimierungspotenzial. So wäre es sinnvoll, eine gesonderte Behandlung von Umlauten anzubieten. Anstatt unbekannte Zeichen zu löschen oder zu ignorieren könnte man eine Option implementieren, Umlaute umzuwandeln. So könnte man sagen, man bildet diese auf eine Kombination aus zwei Buchstaben ab, zum Beispiel $\ddot{a} \rightarrow ae$. Somit könnten auch Umlaute mit der M-138 verschlüsselt werden.

Literaturverzeichnis

[Ass96] Association, Maritime Park: *Operating Instructions for CSP-845 (a.k.a. M-138A)*. http://maritime.org/tech/csp845inst.htm, 1996. [Online; zugegriffen am 1. April 2015].

[BE] Bernhard Esslinger, Alexander May und Arno Wacker: *Mystery Twister C3 - The Crypto Challenge contest*. https://www.mysterytwisterc3.org.

[Buc10] Buchmann, Johannes: *Einführung in die Kryptographie*. Springer-Verlag, 2010.

[Ess08] Esslinger, Bernhard: *CrypTool*. Available via www.cryptool.de, 2008.

[Hoe] Hoerenberg, Michael: *Breaking German Navy Ciphers*. http://www.enigma.hoerenberg.com/. [Online; zugegriffen am 26. April 2015].

[Joh] John: *Open Security Research*. http://calc.opensecurityresearch.com/.

[KKWE14] Kopal, Nils, Olga Kieselmann, Arno Wacker und Bernhard Esslinger: *CrypTool 2.0*. Datenschutz und Datensicherheit - DuD, 38(10):701–708, 2014.

[Lev09] Levy, Steven: *Mission Impossible: The Code Even the CIA Can't Crack*, 2009.

[Mar] Maritime Park Association: *CSP-845*. http://maritime.org/tech/csp845.htm. [Online; zugegriffen am 17. April 2015].

[Nat12] National Institute of Standards and Technology: *Recommendation for Key Management*. http://csrc.nist.gov/publications/nistpubs, 2012.

[Sch] Schmeh, Klaus: *M-138 Challenges*. http://scienceblogs.de/klausis-krypto-kolumne/m-138-challenge/. [Online; zugegriffen am 23. April 2015].

[Sch14] Scherschel, Fabian: *Krypto-Lernsoftware CrypTool 2 im Neuen Gewand*. http://heise.de/-2303191, August 2014. [Online; zugegriffen am 2. April 2015].

[T.12] T., Christos: *US Military Strip Ciphers.* http://chris-intel-corner.blogspot.de/2012/10/us-military-strip-ciphers.html, Oktober 2012. [Online; zugegriffen am 11. April 2015].

[Wac14] Wacker, Arno: *Introduction to Applied Cryptology*, 2014.

Versicherung an Eides Statt

Ich, Nils Rehwald, Matrikelnummer 31202848, wohnhaft in 34117 Kassel, versichere an Eides Statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen übernommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe.

Ich versichere an Eides Statt, dass ich die vorgenannten Angaben nach bestem Wissen und Gewissen gemacht habe und dass die Angaben der Wahrheit entsprechen und ich nichts verschwiegen habe.

Die Strafbarkeit einer falschen eidesstattlichen Versicherung ist mir bekannt, namentlich die Strafandrohung gemäß § 156 StGB bis zu drei Jahren Freiheitsstrafe oder Geldstrafe bei vorsätzlicher Begehung der Tat bzw. gemäß § 163 StGB bis zu einem Jahr Freiheitsstrafe oder Geldstrafe bei fahrlässiger Begehung.

Kassel, 29. April 2015 — Nils Rehwald