UNIVERSITÄT SIEGEN

Naturwissenschaftlich Technische Fakultät

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science Informatik

Implementierung und Visualisierung verschiedener kryptografischer Handverfahren und Codes in CrypTool 2

Betreuer	Herr Dr. Nils Kopal
Erstprüfer	Herr Prof. Bernhard Esslinger
Zweitprüfer	Herr Prof. Dr. Roland Wismüller

vorgelegt von	Niklas Weimann
Matrikelnummer	1352285
Abgabedatum	1. Juni 2021

Kurzzusammenfassung

Die vorliegende Arbeit beschäftigt sich mit sechs verschiedenen kryptografischen Handverfahren und Codes. Dies sind die Codes T9 und Bacon, und die vier Chiffren Straddling-Checkerboard, Ché Guevara, Josse-Cipher und Chaocipher.

Jedes dieser Verfahren wurde in dieser Arbeit zunächst beschrieben, anhand eines Beispiels durchgeführt und danach kryptografisch analysiert. Es wurde für jedes Verfahren eine Komponente mit Visualisierung in der Lernplattform CrypTool 2 erstellt. Zusätzlich wurde eine Komponente für einen kryptografischen Angriff gegen das Josse-Cipher-Verfahren mittels Simulated Annealing implementiert.

Das Josse-Cipher-Verfahren wurde in dieser Arbeit zum ersten Mal in einer öffentlich zugänglichen Implementierung verfügbar gemacht und erstmals in deutscher Sprache beschrieben.

Diese Arbeit dokumentiert die Funktionsweise einiger ungewöhnlicher Chiffren und bietet durch die Implementierung der Komponenten in CrypTool 2 die Möglichkeit, diese Verfahren leichter zu verstehen und interaktiv zu begreifen.

Abstract

This thesis deals with six different cryptographic hand ciphers and codes. These are the codes T9 and Bacon cipher, and the 4 ciphers Straddling Checkerboard, Ché Guevara, Josse cipher and Chaocipher.

In this thesis, each of the procedures was first described, performed using an example, and then cryptographically analyzed. Afterwards, a component was created for each procedure in the CrypTool 2 learning platform. The components have internal visualizations that illustrate how the procedure works. In addition, a component was implemented for a cryptographic attack against the Josse cipher using simulated annealing.

In this thesis, the Josse cipher was made available for the first time in a publicly accessible implementation and described in German for the first time.

This work documents the functionality of some less common ciphers and offers the possibility to make these methods easier to understand by implementing the components in CrypTool 2.

Inhaltsverzeichnis

1. Einleitung

Kryptografische Verfahren finden bereits seit über 2000 Jahren Anwendung im militärischen Kontext und bei der Kommunikation zwischen Regierungs-Institutionen. Die kryptografischen Verfahren entwickelten sich innerhalb dieser Zeit stetig weiter und haben inzwischen in fast alle Lebensbereiche Einzug gehalten. Dennoch sind auch in der heutigen Zeit historische Handverfahren interessant, denn sie bieten einen Zugang in die Welt der Kryptografie. Dadurch können Lernende selbst erfahren, wie Verschlüsselung funktioniert. Kryptografische Handverfahren sind nicht gleichzusetzen mit Codes, da mit dem Begriff Handverfahren alte Chiffren gemeint sind, die verwendet wurden, um Informationen vor Dritter geheim zu halten. Codes wurden und werden eingesetzt, um Daten in eine andere Form zu bringen, damit sie beispielsweise einfacher übertragen werden können. Im Gegensatz zu Chiffren verwenden Codes also keinen Schlüssel.

Historische Handverfahren werden immer wieder zu Dokumentations- und Analysezwecken betrachtet. So gibt es auch heute noch Verfahren, die nur wenig bis gar nicht wissenschaftlich untersucht oder beschrieben wurden, da sie zum Beispiel lange Zeit in Archiven unter Verschluss gehalten wurden, wie die Chiffre von Josse, zu der die ersten Veröffentlichungen aus den Jahren 2017 [14, S. 415–421], 2018 [2, S. XV] und 2020 [4] stammen. Ebenso gibt es Verfahren, die aus anderen Gründen lange Zeit nicht für die Öffentlichkeit zugänglich waren. Ein Beispiel für ein solches Verfahren ist die Chaocipher von John F. Byrne (1880-1960). Dieses Verfahren wurde erst im Jahre 2010 durch die Erben von Byrne veröffentlicht. [15]

CrypTool 2 (CT2) ist eine Lernplattform, die in der Krypto-Community bereits sehr bekannt ist und in der Lehre weltweit zum Einsatz kommt. Somit stellt CT2 einen idealen Ort dar, um diese Verfahren interaktiv zu demonstrieren und praktisch zu analysieren.

1.1. Motivation

Für das Erlernen von kryptografischen Fähigkeiten ist es wichtig, dass bei den Lerninhalten eine möglichst geringe Hürde hinsichtlich der Komplexität gesetzt wird. CT2 ist ein Computerprogramm, das weltweit an Schulen, Universitäten und in Unternehmen eingesetzt wird. CT2 wurde entwickelt, um Lernenden einen einfachen Einstieg in die Welt der Kryptografie zu ermöglichen. Um den Prozess des Lernens bestmöglich zu unterstützen, bietet CT2 eine visuelle Darstellung für eine Vielzahl von Verfahren. Durch die intuitive Benutzeroberfläche wird der Einstieg in die Kryptografie erleichtert. Die Sammlung an Verfahren, die unterstützt werden, wird stetig erweitert. Somit können sowohl einzelne Verfahren als auch Konzepte mit den angebotenen Komponenten verdeutlicht werden. Durch die visuelle Programmierung kann der individuelle Lernprozess der Anwender unterstützt werden.

CT2 bietet bereits eine große Auswahl von historischen Handverfahren, jedoch werden immer wieder neue Verfahren entdeckt. Beispielsweise wurde im Jahre 2020 zum ersten Mal das Verfahren von Major H. D. Josse in einem wissenschaftlichen Journal beschrieben und analysiert. [4] Im Rahmen dieser Arbeit wird das Verfahren zum ersten Mal in einer öffentlich zugänglichen Implementierung verfügbar gemacht. Um dies zu realisieren muss ein Teil der Funktionsweise aus den originalen Aufzeichnungen von Josse aus dem 19. Jahrhundert hergeleitet werden.

1.2. Ziele

Im Rahmen dieser Bachelorarbeit werden sechs verschiedene kryptografische Handverfahren und Codes beschrieben, analysiert und mittels CT2 implementiert bzw. visualisiert. Die Visualisierung der Verfahren hat zum Ziel, den Zugang zu den Verfahren zu erleichtern. Dabei soll der Fokus auf eine einsteigerfreundliche Beschreibung und Visualisierung gelegt werden, damit auch Nicht-Krypto-Experten die kryptografischen Handverfahren und Codes verstehen können. Das Ziel dieser Arbeit ist es nicht, die Verfahren einer tiefgründigen Kryptoanalyse zu unterziehen, deshalb wird diese im Rahmen dieser Arbeit nur oberflächlich behandelt und soll nur die Idee für entsprechende Angriffe darstellen, um einen Einstieg in die Thematik zu bieten.

Die sechs in dieser Arbeit behandelten Verfahren sind die Chiffren von Ché Guevara und das zugrunde liegende Straddling-Checkerboard, die Chiffre von Josse und das Chaocipher-Verfahren, sowie die beiden Codes, die sogenannte Bacon-Chiffre und der T9-Code (mobile phone code).

1.3. Aufbau der Arbeit

Diese Bachelorarbeit ist in fünf Kapitel unterteilt. Kapitel 2 widmet sich den Grundlagen, die für diese Arbeit nötig sind. In Kapitel 3 wird die Funktionsweise der einzelnen Verfahren anhand von Beispielen erläutert und jedes Verfahren wird kryptoanalytisch betrachtet. Kapitel 4 ist das Hauptkapitel: Darin wird die Vorgehensweise bei der Konzeptionierung und Implementierung der verschiedenen Verfahren erläutert. Der Code wurde für das entsprechende Verfahren zunächst als Konsolenanwendung implementiert und anschließend in das Plugin-Konzept von CT2 migriert. In Kapitel 5 wird überprüft, ob die Ziele der Arbeit erreicht wurden, es werden die wichtigsten Ergebnisse der Arbeit zusammengefasst und es wird ein Ausblick auf mögliche zukünftige Weiterentwicklungen der Komponenten gegeben.

1.4. Verwendete Software

Diese Bachelorarbeit wurde mit der Textverarbeitungs-Software LaTeX in Overleaf erstellt. Die Abbildungen wurden durch die Open-Source-Software Draw.io und mit PowerPoint erstellt. Der Code der Konsolenanwendungen wurde mit der Software Rider von Jetbrains erstellt und die Entwicklung der CT2-Komponenten wurde mit der Entwicklungsumgebung Visual Studio 2019 entwickelt.

2. Grundlagen

Dieses Kapitel erläutert die Grundlagen, die zum Verständnis dieser Arbeit notwendig sind. Dabei wird zunächst der Unterschied zwischen Substitution und Transposition erläutert. Anschließend werden verschiedene Arten von Angriffen gegen Chiffren gezeigt. Außerdem werden die Eigenschaften und Kennzahlen von Chiffren dargestellt, die zur Sicherheit der Verfahren beitragen und für die Kryptoanalyse von Bedeutung sind.

2.1. Kryptografie

Kryptologie umfasst die beiden Bereiche Kryptografie (Erstellen von Chiffren) und Kryptoanalyse (Analysieren der Sicherheit und Brechen von Chiffren). Die klassische Kryptografie betrachtete nur die Verschlüsselung, also wie ein Text so verändert werden kann, dass die Informationen, die er übermittelt, vor Dritten geheimgehalten werden können. Die moderne Kryptografie beschäftigt sich zudem mit der Authentifikation sowie der Integrität von Daten. Diese Arbeit beschäftigt sich jedoch mit der klassischen Kryptografie, daher wird auf die moderne Kryptografie nicht näher eingegangen. [13, S. 2–3]

Es gibt asymmetrische und symmetrische Chiffren. In dieser Arbeit werden nur symmetrische Chiffren diskutiert, daher wird auf eine ausführliche Beschreibung von asymmetrischen Chiffren verzichtet. Eine symmetrische Chiffre ist ein Kryptosystem, das man mit fünf Parametern beschreiben kann: Einem Klartext, einem Geheimtext, einem Schlüssel sowie Transformationen zum Ver- und Entschlüsseln. Dabei kann mit dem Schlüssel und einem Klartext durch eine Verschlüsselungstransformation der Geheimtext erzeugt werden und umgekehrt durch eine Entschlüsselungstransformation der Klartext. Die Zeichen, die für den Klartext verwendet werden dürfen, werden als Klartextalphabet bezeichnet. Entsprechend werden die Zeichen, die für den Geheimtext verwendet werden dürfen, als Geheimtextalphabet bezeichnet. [18, S. 3] Die Transformationen müssen eindeutig umkehrbar sein, damit ein Geheimtext wieder entschlüsselt werden kann.

Die zwei wichtigsten klassischen Transformationen sind die Substitution und die Transposition.

2.1.1. Substitution

Bei der Substitution werden einzelne Zeichen, Zeichengruppen, einzelnen Bits oder Gruppen von Bits ersetzt, um den Klartext in einen Geheimtext umzuwandeln. Die Ersetzung der Zeichen erfolgt dazu nach einem festen Schema. Die Umkehrung der Ersetzung wird

zur Entschlüsselung verwendet und liefert aus dem verschlüsselten Text wieder den ursprünglichen Klartext. [18, S. 17] Ein Beispiel für eine Substitution ist in Abbildung 1 dargestellt.

2.1.2. Transposition

Im Gegensatz zur Substitution steht die Transposition. Statt die Zeichen des Klartextes zu ersetzen, werden bei der Transposition die Zeichen des Klartextes in einer neue Reihenfolge angeordnet. Ein häufig verwendetes Verfahren für die Transposition ist die Spaltentransposition. Dabei wird der Klartext zeilenweise in eine Tabelle eingetragen und spaltenweise wieder ausgelesen, somit ändert sich die Reihenfolge der Zeichen [18, S. 17]. Ein Beispiel einer möglichen Transposition ist in der linken Hälfte der Abbildung 1 abgebildet.

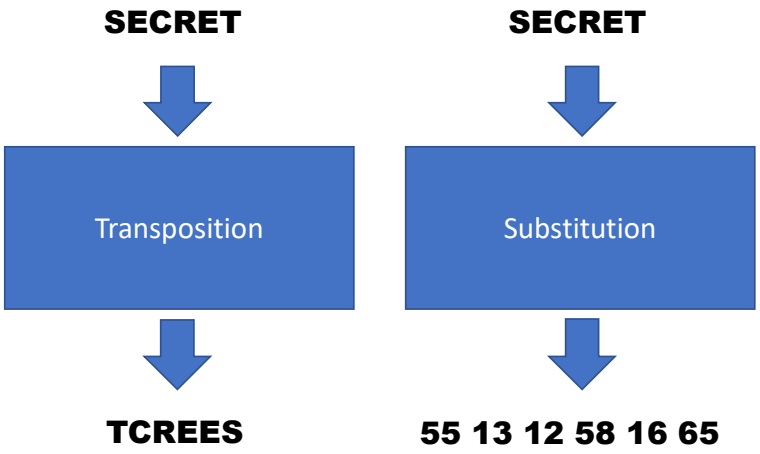

Abbildung 1: Vergleich von Transposition und Substitution

2.1.3. One-Time-Pad

Ein One-Time-Pad (OTP) ist eine zufällige Zeichenfolge, die zum Verschlüsseln verwendet wird. Dabei werden die Zeichen des Klartextes mittels des OTP einzeln in den Geheimtext umgewandelt. Das Besondere an dieser Zeichenfolge ist, dass sie sich niemals wiederholt und nur einmalig verwendet werden darf. Wird ein OTP mehrfach verwendet, kann aus zwei Geheimtexten das OTP errechnet werden und damit die Verschlüsselung gebrochen werden. Aus diesen Bedingungen ergibt sich, dass das OTP mindestens so lang sein muss, wie der Klartext, der mit ihm verschlüsselt werden soll.

Ein mit einem OTP verschlüsselter Klartext kann niemals gebrochen werden, da durch die zufällige Zeichenfolge beim Entschlüsseln jeder Klartext mit der gleichen Wahrschein-

lichkeit erzeugt werden kann. Somit kann keine Aussage über den korrekten Klartext getroffen werden.

Jedoch gibt es auch einen massiven Nachteil: Die zufällige Zeichenfolge für den Schlüssel muss immer über einen zweiten Kanal übertragen werden und kann maximal einmal verwendet werden, was dazu führt, dass dieses Verfahren in der Praxis nicht anwendbar ist. [13, S. 36–38]

2.1.4. Unterschied zwischen Codierungen und Chiffren

Codierungen unterscheiden sich von Chiffren insofern, als dass Chiffren dazu verwendet werden, Informationen vor Dritten geheim zu halten. Bei Codierungen ist das Ziel nicht die Geheimhaltung, sondern nur die Optimierung der Darstellung der Information. So können Daten mit dem Ziel codiert werden, dass sie auf dem Übertragungsweg möglichst robust gegenüber Fehlern sind oder weniger Speicherplatz verbrauchen verwendet wird. Ein weiterer Unterschied zwischen Kodierungen und Chiffren ist, dass Kodierungen ohne einen Schlüssel auskommen und die Daten nach einem festen Schema transformieren.

2.2. Kryptoanalyse

Kryptoanalyse ist der Gegenspieler zur Kryptografie und beschäftigt sich mit dem „Brechen“ von Chiffren. Eine Chiffre wird als gebrochen bezeichnet, wenn es einem Kryptoanalytiker möglich ist, aus dem Geheimtext den ursprünglichen Klartext oder den Schlüssel, der zum Verschlüsseln verwendet wurde, zu rekonstruieren. Ebenso gilt eine Chiffre als gebrochen, wenn aus einem gegebenen Klartext mit dem dazugehörigen Geheimtext der Schlüssel hergeleitet werden kann, der verwendet wurde, um aus dem Klartext den Geheimtext zu erzeugen. Moderne Verfahren gelten auch dann als gebrochen, wenn man Analysemethoden findet, die deutlich schneller sind, als eine Brute-Force-Suche. [18, S. 2–3]

2.2.1. Angriffe

Angriffe auf die Sicherheit von Chiffren können in drei Angriffstypen eingeteilt werden. Man beschreibt damit die Fähigkeiten und das Vorwissen des Angreifers. Diese drei Angriffstypen werden nachfolgend beschrieben.

Ciphertext-only-Angriff Bei dieser Art des Angriffs kennt der Kryptoanalytiker nur Geheimtext, der mit einem bekannten Verfahren verschlüsselt wurde. Aus diesem Geheimtext kann der Kryptoanalytiker den dazugehörigen Klartext bzw. den Schlüssel herleiten. Dieser Angriffstyp ist der mächtigste, weil der Kryptoanalytiker hier das wenigste Vorwissen besitzt.

Known-Plaintext-Angriff Hierbei kennt man neben dem Geheimtext auch Teile des Klartexts oder sogar den vollständigen Klartext. Das Ziel dieses Angriffes ist es, den Schlüssel und gegebenenfalls die restlichen Teile des Klartexts zu finden. Hilfreich sind bestimmte Zeichenfolgen (sogenannte Cribs), die immer wieder vorkommen, zum Beispiel Grußformeln, wie „Sehr geehrte Damen und Herren“ oder „Mit freundlichen Grüßen“. Dieser Angriffstyp ist der zweitmächtigste.

Chosen-Plaintext-Angriff Falls ein Kryptoanalytiker für viele selbst gewählte Klartexte die entsprechenden Geheimtexte erhalten kann, so wird dies als Chosen-Plaintext-Angriff bezeichnet. Die Annahme ist hier, dass bei allen Klartext-/Geheimtext-Paaren der gleiche Schlüssel verwendet wird. Dieser Angriffstyp erscheint abstrakt, er wurde aber in der Historie eingesetzt, indem der Angreifer den Gegner dazu provozierte bestimmte Texte zu verschlüsseln. In der modernen Kryptografie ist dieser Angriffstyp aktueller, da moderne Kommunikationsprotokolle auf Eingaben reagieren müssen. Für klassische Kryptoverfahren ist dieser Angriffstyp eher irrelevant.

2.2.2. Brute-Force-Angriff

Ein Angriff, der bei jeder Chiffre anwendbar ist, ist der Brute-Force-Angriff. Bei Brute-Force-Angriffen wird versucht, den gesamten Schlüsselraum eines Verfahrens zu durchlaufen und alle möglichen Schlüssel des Verfahrens auszuprobieren. Ein solcher Angriff scheitert häufig daran, dass es zu viele mögliche Schlüssel im Schlüsselraum einer Chiffre gibt. Zu viele Schlüssel gibt es dann, wenn nicht alle Möglichkeiten in einer praktikablen Zeit ausprobiert werden können. Daher ist die Größe des Schlüsselraums (Kapitel 2.2.4) wichtig für die Bewertung der Sicherheit eines Verfahrens bezüglich Brute-Force-Angriffen. Damit hat man eine Obergrenze für den Aufwand.

2.2.3. Statistische Angriffe

Statistische Eigenschaften eines Textes können häufig dazu verwendet werden klassische Verfahren zu brechen, da klassische Verfahren diese Eigenschaften nicht verbergen. Es

Text	Wert
hhhhhfhfhhfhfhfhfhfhhahfashfhafhahfhahfh	0.39359
Der Koinzidenzindex ist ein in der Kryptoanalyse verwendeter Indikator	0.07879

Abbildung 2: Koinzidenzindex: Vergleich zwischen zufälligem und deutschem Text

kann beispielsweise die Häufigkeit einzelner Buchstaben oder Buchstabengruppen analysiert werden, um Rückschlüsse auf die angewendete Substitution ziehen zu können. Taucht ein Zeichen besonders häufig im Geheimtext auf, so ist die Wahrscheinlichkeit bei deutschen Texten groß, dass es sich bei diesem Zeichen um den Buchstaben „E“ handelt.

Die Analyse ist nicht auf einzelne Buchstaben beschränkt, auch die Platzierung von Leerzeichen kann Rückschlüsse auf den Klartext zulassen, wenn etwa häufig eine dreier Gruppe von Buchstaben vor einer längeren Buchstabengruppe auftaucht, so ist die Wahrscheinlichkeit groß, dass es sich um „DER“, „DIE“ oder „DAS“ handelt. [13, S. 8–9]

Außerdem existieren Metriken, mit denen ein Text bewertet werden kann, wie sehr er einer natürlichen Sprache ähnelt. Beispiele für solche Metriken sind etwa N-Gramme, die Entropie oder der Koinzidenzindex. Sie können beispielsweise bei Angriffen verwendet werden, um zu untersuchen, ob mit einem zufällig erzeugten Schlüssel aus einem gegebenen Geheimtext ein korrekter Klartext erzeugt werden kann. Jede dieser Metriken berechnet aus einem Text einen Wert. Dieser Wert nähert sich, je nach Verfahren, einem festen Wert an. Für den Koinzidenzindex nähert sich der Wert eines deutschen Textes etwa 0.0762 an. In Abbildung 2 werden beispielhaft die Werte für einen deutschen Satz und eine zufällige Zeichenfolge mittels des Koinzidenzindexes verglichen.

2.2.4. Sicherheitsmetriken von Chiffren

Für Chiffren gibt es verschiedene Eigenschaften (Metriken), die verwendet werden können, um die Sicherheit des Verfahrens zu quantifizieren. Die zwei wichtigsten Eigenschaften dabei sind der Schlüsselraum und die Unizitätslänge. Nachfolgend werden deren Verwendung und Bedeutung kurz erläutert. Außerdem spielt Kerckhoffs' Prinzip eine wichtige Rolle für die Sicherheit von kryptografischen Verfahren.

Wichtig zu beachten ist jedoch, dass es sich bei den hier vorgestellten Eigenschaften nur um notwendige Bedienungen für die Sicherheit eines Verfahrens handelt. Dies bedeutet, dass ein Verfahren nicht sicher ist, nur weil es einen großen Schlüsselraum besitzt. So kann beispielsweise eine monoalphabetische Substitution einen sehr großen Schlüs-

selraum besitzen, jedoch durch eine Häufigkeitsanalyse (Kapitel 2.2.3 vergleichsweise einfach gebrochen werden.

Schlüsselraum Der Schlüsselraum ist eine Angabe dazu, wie viele Schlüssel bei einer Chiffre möglich sind. Die Möglichkeiten leiten sich durch die Bedingungen des Verfahrens an den Schlüssel ab. Wenn zum Beispiel bei einer Chiffre nur das Verschieben des Alphabetes möglich ist und das Alphabet aus 26 Buchstaben besteht, dann hat der Schlüsselraum des Verfahrens eine Größe von 26. Wenn jedoch der Schlüssel aus 4 beliebigen Zeichen aus einer Menge von 26 Buchstaben besteht, dann kann mittels Kombinatorik die Größe des Schlüsselraums auf $26^4 = 456.976$ berechnet werden. Somit ist das zweite Verfahren resistenter gegen einen Brute-Force-Angriff (Kapitel 2.2.2), als das erste Verfahren, da im Durchschnitt wesentlich mehr Möglichkeiten ausprobiert werden müssen.

Unizitätslänge Unizitätslänge (eng. unicity distance) wurde von Claude Shannon im Jahre 1949 in seiner Arbeit „Communication Theory of Secrecy Systems“ [16] beschrieben. Sie gibt an, wie lang ein Geheimtext mindestens sein muss, damit durch einen Brute-Force-Angriff (Kapitel 2.2.2) ein eindeutiger Klartext rekonstruiert werden kann. Das bedeutet, dass bei einem Text, der länger als die Unizitätslänge für die Chriffe ist, eine eindeutige Lösung gefunden werden kann, nachdem jeder mögliche Schlüssel ausprobiert wurde. [13, S. 136] Die Unizitätslänge U wird durch folgende Formel berechnet:

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

Der Wert $H(K)$ gibt dabei die Entropie des Schlüsselraumes K an. Umgangssprachlicher ausgedrückt also: Wie viele Bits sind notwendig, um alle möglichen Nachrichten einer Nachrichtenquelle darstellen zu können, wenn jede Nachricht gleich wahrscheinlich ist. Wenn es eine Nachrichtenquelle K gibt, die $2^k, k \in \mathbb{N}$ gleich wahrscheinliche Nachrichten erzeugt, so gilt offenbar, dass $H(K) = k$. [18, S. 5]

Diese Definition wird verständlicher, wenn man dazu das folgende Beispiel betrachtet: Angenommen eine Chiffre (monoalphabetische Substitution) hat $K = 25! \approx 2^{83,68}$ gleich wahrscheinliche Schlüssel. Der Schlüsselraum hat also eine Größe von $2^{83,68}$, also gilt $H(K) = \log_2 2^{83,68} = 83,68$. Die Entropie des Schlüsselraumes dieser Chiffre beträgt also $83,68$.

Der Wert D gibt die Redundanz des Klartextes bzw. einer Sprache an. Diese Redundanz berechnet sich zum einen aus der maximalen Anzahl Bits an Information R in einem Buchstaben und zum Anderen aus der tatsächlichen Informationsdichte der Sprache r. R wird durch die Formel: $R = \log_2 26 = 4,7$ ausgedrückt, wobei 26 die Anzahl der

Buchstaben des verwendeten Alphabetes ist. R gibt an, wie viele Informationen theoretisch maximal durch einen Buchstaben übertragen werden können, wenn alle Buchstaben gleich wahrscheinlich sind. Jedoch sind in den meisten Sprachen die Buchstaben nicht alle gleich wahrscheinlich. Komprimiert man einen Text in natürlicher Sprache, stellt man fest, dass ein einzelner Buchstabe lediglich etwa $1,5$ Bits an Information enthält und das zuvor berechnete Maximum von $4,7$ nicht erreicht wird. Dies bedeutet für die Redundanz des Klartextes, dass diese bestimmt werden kann, indem R von r abgezogen wird.

In unserer Beispielchiffre und für einen Klartext in deutscher Sprache ergibt sich also: $D = R - r = \log_2 26 - 1,5 \approx 3,2$. [18, S. 5]

Abschließend kann die Unizitätslänge der oben angenommen Chiffre mit dem Schlüsselraum K, der eine Größe von 25! gleich wahrscheinlichen Schlüsseln besitzt, durch

$$U = \frac{H(K)}{D} = \frac{83,68}{3,2} = 27,89$$

berechnet werden. Somit ergibt sich, dass mindestens $27,89$, also aufgerundet 28, Zeichen Geheimtext vorhanden sein müssen, um durch einen Brute-Force-Angriff zu einer eindeutigen Lösung zu gelangen. Diese Angabe garantiert jedoch nicht, dass ein Geheimtext, der 28 oder mehr Zeichen enthält, garantiert eindeutig gebrochen werden kann. Die Unizitätslänge gibt vielmehr nur eine theoretische Mindestgröße an, ab der der Geheimtext mit einer eindeutigen Lösung gebrochen werden kann. Unterhalb dieser Größe ist es möglich, dass mehrere Lösungen als korrekt erscheinen.

OTP (Kapitel 2.1.3) haben eine Unizitätslänge von ∞. Somit ergibt sich auch, dass OTPs nicht gebrochen werden können. Dies kann dadurch begründet werden, dass die Zeichenfolge des OTPs eine unendliche Länge hat. Daraus folgt, dass $H(K)$ unendlich ist, somit ergibt sich auch $n \approx \infty$. [18, S. 8–10]

Kerckhoffs' Prinzip Kerckhoffs' Prinzip verlangt, dass ein Kryptosystem auch dann sicher sein soll, wenn der Angreifer die Funktionsweise des Verfahrens kennt. Die Sicherheit des Verfahrens soll also nur vom gewählten Schlüssel abhängen. Dieses von Auguste Kerckhoffs im Jahre 1883 definierte Prinzip spielt eine wichtige Rolle, wenn es um die Sicherheit eines Kryptosystemes geht. In der Vergangenheit existierten immer wieder Verfahren (Content Scrambling System (CSS) für DVDs, Chaocipher (3.6), GSM), die, nachdem die Funktionsweise öffentlich bekannt wurde, relativ einfach gebrochen werden konnten. Deshalb stellt dieses Prinzip die Bedingung, dass selbst nach dem Bekanntwerden des Verfahrens die Sicherheit unverändert gewährleistet sein soll. [13, S. 11]

2.3. Verwandte Arbeiten

In diesem Kapitel werden Arbeiten vorgestellt, die sich ebenfalls mit den in dieser Arbeit vorgestellten Verfahren beschäftigen.

T9-Code Für das Verfahren des T9-Codes gibt es einige Online-Angebote, die die Codierung und Decodierung implementieren. Jedoch besitzen diese Implementierungen nur selten die Möglichkeit einen kodierten Text zu dekodieren. Ein Angebot, das neben der Dekodierung auch eine Beschreibung des Verfahrens bietet, ist die Internetseite Kryptografie.de von Oliver Kuhlemann. [9]

Straddling-Checkerboard Das Straddling-Checkerboard wird auf der Internetseite von Kuhlemann mit einigen Beispielen erläutert. [10] Zusätzlich bietet die Internetseite eine Anwendung, mit der das Straddling-Checkerboard im Browser durchgeführt werden kann. Dazu muss lediglich der Schlüssel und der zu verarbeitende Text angegeben werden. Außerdem wird dieses Verfahren in David Kahns Buch „The Codebreakers - The Story of Secret Writing“ beschrieben. [7, S. 635–636]

Ché Guevara Das Verfahren von Ché Guevara wird unter anderem durch einen Gastbeitrag von David Kahn in einer Biografie über Guevara beschrieben. Dort beleuchtet Kahn auch die Sicherheit des OTPs. [6] Außerdem wird das Verfahren auf der Internetseite von Kuhlemann beschrieben. Er bietet dort eine Eingabemaske, mit der Texte mit dem Ché-Guevara-Verfahren verschlüsselt werden können. [8]

Bacon-Chiffre Die Bacon-Chiffre ist das wohl bekannteste und einfachste aller, hier vorgestellten, Verfahren. Einige Internetseiten, wie etwa dcode.fr[1] oder cryptii.com [2], bieten die Möglichkeit, das Verfahren zum Ver- und Entschlüsseln von Texten zu verwenden. Das Verfahren wird auf diesen Seiten jedoch meist nur als Closed-Source-Software, also ohne Zugang zum Quellcode der Seiten, angeboten. Die American Cryptogram Association (ACA) hat auf ihrer Website das Verfahren beschrieben und einige Anwendungsmöglichkeiten erläutert. [1] Außerdem beschäftigt sich ein Beitrag im Journal „Genetics“ mit dem Verfahren. [5]

Josse-Cipher Für die Josse-Cipher existierte, neben den Original-Dokumenten, nur ein einziger wissenschaftlicher Artikel, in dem das Verfahren beschrieben wird. [4] Die-

[1] https://www.dcode.fr/bacon-cipher
[2] https://cryptii.com/pipes/bacon-cipher

ses Paper beleuchtet Josses Umfeld, die Funktionsweise des Verfahrens und führt eine einfache kryptografische Analyse des Verfahrens durch. Während dieser Arbeit entwickelte jedoch George Lasry Angriffe mittels Simulated Annealing und Isomorphismen gegen das Josse-Cipher-Verfahren. In seinem Paper beschreibt Lasry neben den Angriffen auch die Anwendung des Verfahrens. [11] Lasry beschreibt das Verfahren so, dass durch die Beschreibung eine Anwendung des Verfahrens möglich ist. Durch die Beschreibung von Rémi Géraud-Stewart und David Naccache in [4] ließ sich das Verfahren nicht reproduzieren.

Chaocipher Moshe Rubin betreibt eine Website[3], auf der er viele Informationen über das Verfahren gesammelt hat und regelmäßig Updates veröffentlicht. Die Updates bestehen meist aus Verweisen auf Arbeiten, in denen das Chaocipher-Verfahren erwähnt wird. Rubin beschrieb das Verfahren im Jahre 2010 kurz nach der Veröffentlichung der Dokumente in einer Arbeit mit dem Titel „CHAOCIPHER REVEALED: THE ALGORITHM“ [15]. Außerdem wurde die Sicherheit des Verfahrens durch Lasry, Rubin, Kopal und Wacker im Jahr 2016 in einem Paper [12] analysiert.

2.4. Schreibweise der Verfahren

Auch wenn Bacon ein Code ist, hat sich die Bezeichnung „Bacon-Chiffre“ etabliert, so dass wir diese beibehalten.

Ebenso schreiben wir Josse-Cipher oder Josses Cipher und Chaocipher, mal getrennt und mal zusammen, weil das die üblichen Schreibweisen sind.

[3] http://www.chaocipher.com/

3. Funktionsweise und Kryptoanalyse der ausgewählten Verfahren

Das nachfolgende Kapitel erklärt die Funktionsweise von sechs verschiedenen Verfahren, ergänzt um Beispiele und deren Kryptoanalyse. Alle Verfahren waren bislang nicht in CT2 implementiert. Josses Cipher wurde bislang noch nicht in einer öffentlichen Implementierung zur Verfügung gestellt. Für die Chaocipher ist dies die bislang erste interaktive Visualisierung des Verfahrens.

3.1. T9-Code

Der T9-Code ist eine Kodierung, die durch das Layout von Tastaturen auf alten Handys inspiriert ist. Tastaturen von Mobiltelefonen enthalten auf den Tasten 2-9 die Buchstaben a-z, wobei die Buchstaben zumeist in Gruppen zu je drei Buchstaben zusammengefasst sind. Eine Ausnahme davon stellen die Tasten 7 und 9 dar, auf ihnen sind statt drei vier Buchstaben gelistet. Der Aufbau einer typischen Tastatur eines Mobiltelefons ist in Abbildung 1 dargestellt.

Um Buchstaben mittels dieser Tastatur eingeben zu können, mussten die Tasten mehrfach gedrückt werden, dadurch wurden die Buchstaben auf der Taste zyklisch im Display angezeigt. Somit musste man für ein a die Taste 2 einmal drücken, für ein b die Taste 2 zweimal und für ein c die Taste 2 dreimal drücken. Da durch diese Vorgehensweise eine Taste häufig mehrfach gedrückt werden musste, war die Geschwindigkeit, mit der man Texte eingeben konnte, recht gering. Dieses Problem wurde durch die Erfindung von T9 gelöst.

Bei der Eingabemethode T9 muss nur einmal diejenige Taste gedrückt werden, die den Buchstaben enthält, den man eingeben möchte. Über ein Wörterbuch, in dem jedes Wort mit der Häufigkeit seiner Verwendung gelistet ist, kann bestimmt werden, welches Wort der Benutzer wahrscheinlich eingeben wollte. Da durch die Kombination der verschiedenen Tasten nicht immer auf ein eindeutiges Wort geschlossen werden kann, wird dem Benutzer bei Unklarheit eine Liste aller Möglichkeiten angezeigt. Somit entfällt das mehrfache Drücken einzelner Tasten und die Eingabegeschwindigkeit kann gesteigert werden. [9]

3.1.1. Funktionsweise

Verschlüsselung Aus dieser Eingabemethode kann ein Code hergeleitet werden, indem die Zahlen der Tasten notiert werden, die notwendig sind, um mit T9 ein Wort einzuge-

2	ABC	6	MNO
3	DEF	7	PQRS
4	GHI	8	TUV
5	JKL	9	WXYZ

Tabelle 1: Gruppierung der Buchstaben auf einer Handy-Tastatur

ben. Nach jedem Wort wird eine Null eingefügt, da auf üblichen Handy-Tastaturen die 0 für das Leerzeichen verwendet wird.

Entschlüsselung Um aus einer Reihe von Zahlen wieder einen Klartext erzeugen zu können, muss im ersten Schritt jede 0 durch ein Leerzeichen ersetzt werden. Somit entstehen Gruppen von Zahlen, die durch Leerzeichen getrennt sind, wobei jede Zahlengruppe für ein Wort steht. Um aus einer solchen Zahlengruppe wieder ein Wort zu erhalten, muss für jede Zahl aus der Gruppe die Menge der möglichen Buchstaben als Gruppe aufgeschrieben werden. Anschließend wird das Kartesische Produkt aus allen so entstandenen Gruppen gebildet. Die so entstandene Menge muss mittels eines Wörterbuchs auf existierende Wörter untersucht werden. Bei dieser Konstruktion kann der Fall auftreten, dass zwei Wörter in der Menge des Kartesischen Produktes im Wörterbuch enthalten sind, somit kann der Code nicht eindeutig in einen Klartext umgewandelt werden. In diesem Fall muss über die Semantik des Textes das richtige Wort ausgewählt werden.

3.1.2. Beispiel

Verschlüsselung Um die Wörter `HALLO WELT` zu kodieren, wird mittels der Tabelle 1 zuerst das `H` durch eine `4` ersetzt. Anschließend wird das `A` durch eine `2` ersetzt, dann zweimal das `L`, durch eine `5` ersetzt. Anschließend wird das `O` durch eine `6` ersetzt. Für das Leerzeichen wird eine `0` ergänzt. Die weiteren Ersetzungen lauten dann W=>9, E=>3, L=>5 und T=>8. Somit erhält man den Geheimtext `4255609358`.

Entschlüsselung Bei der Entschlüsselung von Codes, die mit T9 codiert wurden, können zwei verschiedene Fälle eintreten. Zum einen kann der Fall auftreten, dass bei der Decodierung eine eindeutige Lösung entsteht, und zum anderen können mehrere Lösungen beim Decodieren entstehen. Dies kann an dem kodierten Text `3709270532537` nachvollzogen werden. Teilt man diesen Text zunächst in seine einzelnen Wörter erhält man die Wörter `37`, `927` und `532537`. Das Wort `37` kann sowohl durch die Substitution $3 \rightarrow E$ und $7 \rightarrow S$ zu `ES`, als auch durch die Substitution $3 \rightarrow E$ und $7 \rightarrow R$ zu `ER` werden. Für die nächste Gruppe kann aus `927` sowohl `WAR`, als auch `WAS` ersetzt werden. Dazu können die Substitutionen $9 \rightarrow W$, $2 \rightarrow A$ und $7 \rightarrow S/R$ verwendet werden. Das letzte Wort in diesem Satz kann eindeutig decodiert werden. `532537` kann nur durch

die Substitutionen $5 \rightarrow L$, $3 \rightarrow E$, $2 \rightarrow C$, $5 \rightarrow K$, $3 \rightarrow E$ und $7 \rightarrow R$ zu `LECKER` decodiert werden. Somit ergibt sich der Decodierte Satz: `[ER|ES] [WAR|WAS] LECKER`. Aus der Semantik des Satzes kann geschlossen werden, dass es sich um den Klartext `ES WAR LECKER` handeln muss.

3.1.3. Kryptoanalyse

Für das Verfahren des T9-Codes kann keine Kryptoanalyse durchgeführt werden, da es sich bei diesem Verfahren, wie oben bereits erwähnt, um eine Codierung handelt. Codierungen werden nicht verwendet, um Informationen vor den Augen Dritter zu schützen, sondern um Informationen in eine andere Form zu überführen.

3.2. Straddling-Checkerboard

Die Bezeichnung Straddling-Checkerboard setzt sich aus den englischen Wörtern für „spreizen“ und „Schachbrett“ zusammen. Spreizen bedeutet dabei, dass ein einzelnes Zeichen auf ein oder mehr Zeichen abgebildet wird. Das Verfahren basiert auf einer Tabelle, die, ähnlich wie ein Schachbrett, am Rand um eindeutige Bezeichnungen für die einzelnen Spalten bzw. Zeilen ergänzt wurde. Die Beschreibung in Kapitel 3.2.1 orientiert sich an Kuhlemann [10]:

3.2.1. Funktionsweise

Die Tabelle hat üblicherweise 3 Zeilen und 10 Spalten. Die Größe kann jedoch variieren. Die Variation hängt vom Anwender ab, dieser kann mit der Wahl des Schlüsselwortes und des Alphabetes die Größe der Tabelle bestimmen. In der Kopfzeile sowie in der Vorspalte der Tabelle sind die Bezeichner in Form von einstelligen positiven Zahlen gelistet (in Abbildung 3 gelb bzw. grün markiert). Diese Zahlen können in einer beliebigen Reihenfolge aufgelistet werden.

Im Inneren der Tabelle werden Zeichen des Klartextalphabets (hier Buchstaben) notiert. In der ersten Zeile wird ein Schlüsselwort notiert, das keinen Buchstaben doppelt enthalten darf. In der ersten Zeile werden die Felder freigelassen, deren Spaltenbezeichnung bereits für die Bezeichnung einer Zeile verwendet werden. Ab der zweiten Zeile werden die Buchstaben in alphabetischer Reihenfolge aufgeschrieben, wobei die Buchstaben ausgelassen werden, die in der ersten Zeile im Schlüsselwort verwendet wurden. Es können, müssen aber nicht, weitere Felder freigelassen werden. Somit ergibt sich eine Tabelle, wie sie in Abbildung 3 zu sehen ist. Das Schlüsselwort in diesem Beispiel lautet `MEIN`

HAUS (in Abbildung 3 hellblau markiert). Die restlichen Buchstaben in alphabetischer Reihenfolge sind in der Abbildung dunkelblau hinterlegt.

Verschlüsselung Zum Verschlüsseln wird jeder Buchstabe im Klartext mittels des Tabelleninhaltes auf die Zahlen am Rand abgebildet. Die Zahlen am Rand nehmen dabei in etwa die Funktion von Koordinaten an. So wird zum Beispiel der Buchstabe M auf Zeile nichts und Spalte 0, also 0, abgebildet und der Buchstabe G wird auf Zeile 4 und Spalte 4, also 44 abgebildet.

Entschlüsselung Zum Entschlüsseln eines Geheimtextes müssen die Zahlen mittels des Staddling Checkerboards aus Abbildung 3 wieder zurück in den Klartext übersetzt werden. Dazu wird der Klartextbuchstabe über die Bezeichnungen am Rand des Checkerboards basierend auf dem Ciphertext bestimmt. Beispielsweise wird aus 56474718 mit 5 ein H und mit 6 ein A. Dann folgt der erste Buchstabe, der gespreizt wurde, also auf zwei Zeichen abgebildet wurde, denn das L wurde auf die 47 abgebildet und das O wurde auf die 48 abgebildet. Somit ergibt sich der Klartext HALLO.

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

Abbildung 3: Straddling-Checkerboard mit einfachem Alphabet

3.2.2. Beispiel

Verschlüsselung Um den Text CRYPTOOL zu verschlüsseln werden zuerst alle Kleinbuchstaben durch Großbuchstaben ersetzt. Daraus entsteht der Text CRYPTOOL. Als Schlüssel wird das Wort MEINHAUS verwendet und die Spalten werden in aufsteigender Reihenfolge nummeriert. Für die Beschriftung der Zeilen wird 49 verwendet. Aus diesen Angaben ergibt sich das in Abbildung 3 dargestellte Straddling-Checkerboard. Zuerst wird das C durch 41 ersetzt. Anschließend werden folgende Ersetzungen angewendet: $R \rightarrow 91$, $Y \rightarrow 96$, $P \rightarrow 49$, $T \rightarrow 92$, $O \rightarrow 48$, $O \rightarrow 48$, $L \rightarrow 47$. Daraus ergibt sich der Geheimtext 4191964992484847.

Entschlüsselung Für die Entschlüsselung wird dasselbe Setup wie in Abbildung 3 verwendet. Es soll der Geheimtext `46471242` entschlüsselt werden. Da die erste Zahl eine 4 ist und 4 ein Teil der Beschriftung der Zeilen ist, muss der nächste Buchstabe auch noch betrachtet werden, also wird `46` mittels des Straddling-Checkerboards zu einem `K` entschlüsselt. Der restliche Schlüsseltext lautet jetzt noch `471242`. Das erste Zeichen ist jetzt eine `4` und 4 ist wieder Teil der Beschriftung der Zeilen, somit wird die `4` und die nächste Zahl, also die `7` zusammen entschlüsselt, daraus ergibt sich ein `L`. Der restliche Schlüsseltext lautet jetzt noch `1242`. Das erste Zeichen ist jetzt eine `1` und 1 ist nicht Teil der Beschriftung der Zeilen, somit wird die `1` entschlüsselt, daraus ergibt sich ein `E`. Die nächste Zahl ist eine `2`, die ist wieder nicht Teil der Beschriftungen der Zeilen, also kann direkt aus der ersten Zeile entschlüsselt werden, somit ergibt sich ein `I`. Der restliche Schlüsseltext lautet jetzt noch `42`. Da die erste Zahl eine `4` ist und die `4` Teil der Beschriftung der Zeilen ist, muss die nächste Zahl betrachtet werden. Die nächste Zahl ist eine `2`, also muss die `42` entschlüsselt werden, daraus ergibt sich der Buchstabe `D`. Damit ist der gesamte Schlüsseltext entschlüsselt und der Klartext lautet `KLEID`.

3.2.3. Kryptoanalyse

Da Straddling-Checkerboards im Kern eine Ersetzung, also eine Substitution, verwenden, ist es möglich, dass das Verfahren mittels statistischer Analyse gebrochen werden kann. Durch die Spreizung wird die Substitution zwar verschleiert, jedoch lässt sich die Spreizung ebenfalls durch statistische Analyse brechen. Für die Analyse eines Geheimtextes, der mittels eines Straddling-Checkerboards verschlüsselt wurde, bietet sich zunächst eine Häufigkeitsanalyse (Kapitel 2.2.3) an. Mittels der Häufigkeitsanalyse können die Zahlen herausgefunden werden, die für die Beschriftung der Zeilen und damit für die Spreizung verwendet wurden. Die Häufigkeitsanalyse der Zahlen aus dem Geheimtext in Abbildung 4 kann der Tabelle 5 entnommen werden. Daraus folgt, dass die Zahlen 7 und 8 die mit Abstand am häufigsten vorkommenden Zahlen im Geheimtext 4 sind.

```
29867 88166 58784 47373 80814 73776 78477 80682 92478
96729 17676 71824 98473 77706 84807 67381 47377 49737
77768 31817 38347 70477 82818 74960 49715 14234 73777
04980 73476 73813 73571 48367 78373 80824 55497 89672
91767 67349 82777 20457 34073 72410 51482 71483 67775
98678 81667 84918 17367 74771 77737 67349 47782 77701
82807 89607 34947 77564 77774 77
```

Abbildung 4: Geheimtext (verschlüsselt durch ein Straddling-Checkerboard)

Aus dem häufigen Auftreten der Zahlen 7 und 8 kann geschlossen werden, dass die Zeilen im Straddling-Checkerboard wahrscheinlich mit 7 und 8 beschriftet wurden. Daraus

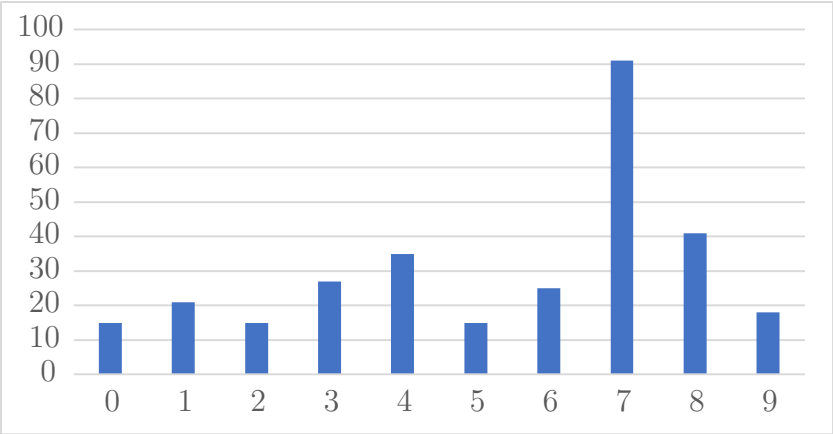

Abbildung 5: Häufigkeit der einzelnen Zahlen in Abbildung 4

2 9 86 78 81 6 6 5 87 84 4 73 73 80 81 4 73 77 6 78 4 77 80 6
82 9 2 4 78 9 6 72 9 1 76 76 71 82 4 9 84 73 77 70 6 84 80 76
73 81 4 73 77 4 9 73 77 77 6 83 1 81 73 83 4 77 0 4 77 82 81
87 4 9 6 0 4 9 71 5 1 4 2 3 4 73 77 70 4 9 80 73 4 76 73 81 3
73 5 71 4 83 6 77 83 73 80 82 4 5 5 4 9 78 9 6 72 9 1 76 76
73 4 9 82 77 72 0 4 5 73 4 0 73 72 4 1 0 5 1 4 82 71 4 83 6
77 75 9 86 78 81 6 6 78 4 9 1 81 73 6 77 4 77 1 77 73 76 73
4 9 4 77 82 77 70 1 82 80 78 9 6 0 73 4 9 4 77 75 6 4 77 77
4 77

Abbildung 6: Geheimtext aus Abbildung 4 (unterteilt in einzelne Buchstaben)

folgt, dass die freien Felder in der ersten Zeile in der 7. beziehungsweise 8. Spalte liegen müssen. Mit diesem Wissen kann der Geheimtext in einzelne Buchstaben unterteilt werden, wie in Abbildung 6 zu sehen ist. Dabei spielt es keine Rolle, in welcher Reihenfolge die 7 und 8 für die Beschriftung der Zeilen verwendet wurden. Das Aufsplitten des Geheimtextes, um die Spreizung zu umgehen ist auch ohne die Kenntnis der Reihenfolge möglich, wie die Abbildung 6 verdeutlicht.

In der neuen Form (Abbildung 6) entspricht jede Zahl, die durch ein Leerzeichen von einer anderen Zahl getrennt ist, einem Buchstaben. Diese Form kann nun mittels Standardverfahren zum Brechen von monoalphabetischer Substitution gebrochen werden. Dazu kann erneut eine Häufigkeitsanalyse durchgeführt werden, jedoch diesmal nicht auf einzelne Zeichen, sondern auf die Zahlengruppen, die durch Leerzeichen getrennt einzelnen Buchstaben entsprechen. Die Häufigkeitsverteilung der einzelnen Zahlengruppen kann der Tabelle 5 entnommen werden. Durch das Aufsplitten des Geheimtextes in einzelne Buchstaben reduziert sich das Problem, den Geheimtext zu entschlüsseln, auf

2 r 86 78 81 6 6 5 87 84 e i i 80 81 e i n 6 78 e n 80 6 82 r 2
e 78 r 6 72 r 1 76 76 71 82 e r 84 i n 70 6 84 80 76 i 81 e i n
e r i n n 6 83 1 81 i 83 e n 0 e n 82 81 87 e r 6 0 e r 71 5 1
e 2 3 e i n 70 e r 80 i e 76 i 81 3 i 5 71 e 83 6 n 83 i 80 82
e 5 5 e r 78 r 6 72 r 1 76 76 i e r 82 n 72 0 e 5 i e 0 i 72 e
1 0 5 1 e 82 71 e 83 6 n 75 r 86 78 81 6 6 78 e r 1 81 i 6 n e
n 1 n i 76 i e r e n 82 n 70 1 82 80 78 r 6 0 i e r e n 75 6 e
n n e n

Abbildung 7: Teilweise entschlüsselter Geheimtext aus Abbildung 4.

das Problem eine monoalphabetische Substitution zu brechen. Würde man versuchen alle Möglichkeiten der Substitution durchprobieren, so gibt es bei der Verwendung des deutschen Alphabetes mit 26 Buchstaben $26! \approx 4 \cdot 10^{26} \approx 2^{88}$ Möglichkeiten für die Zuordnung der Buchstaben. Ein Straddling-Checkerboard mit drei Zeilen bietet Platz für $(3{\cdot}10)-2 = 28$ Buchstaben und hat damit einen Schlüsselraum von $28! \approx 3,05{\cdot}10^{29} \approx 2^{98}$ möglichen Schlüsseln. Dieser Schlüsselraum ist zu groß, um ihn in annehmbarer Zeit mit modernen Computern zu durchsuchen.

Der Schlüsselraum kann jedoch mittels Statistik reduziert werden. Wenn zum Beispiel bekannt ist, dass der der Klartext in deutscher Sprache verfasst wurde (Ciphertext-only-Angriff 2.2.1), dann kann eine entsprechende Häufigkeitstabelle Verwendet werden, um die Chiffre zu brechen. Dieser Angriff entspricht einem Ciphertext-only-Angriff. Im Anhang des Buches Kryptografie von Dietmar Wätjen [18] ist eine solche Häufigkeitstabelle für die deutsche Sprache zu finden. Aus dieser Tabelle geht hervor, dass der häufigste Buchstabe das `e` mit 17,74% ist, danach folgen `n` mit 10,01%, `i` mit 7,60% und `r` mit 6,98%. Die Häufigkeit der Buchstaben aus Abbildung 6 ist in der Tabelle 8 zu sehen. Wenn die häufigsten Buchstaben des Geheimtextes durch die häufigsten Buchstaben der deutschen Sprache ersetzt werden, erhält man somit den Text in Abbildung 7. Zu beachten ist jedoch, dass das Gelingen dieses Angriffes von der Art und Länge des Textes abhängt. Ist der Geheimtext zu kurz, können die statistischen Werte des Textes zu weit von den Vergleichswerten abweichen und somit eine Zuordnung erschweren. Ebenso können Texte mit sehr vielen Sonderzeichen und/oder Zahlen das Ergebnis verfälschen. Hierzu bietet sich die Analyse von Bi- bzw. Trigrammen im Geheimtext an.

Durch ein wenig überlegen und ausprobieren kann somit aus dem Text in Abbildung 4 der Klartext rekonstruiert werden. Dieser lautet: `CRYPTOOL ZWEI IST EIN OPENSOURCEPR-OGRAMM FUER WINDOWS MIT EINER INNOVATIVEN BENUTZEROBERFLAECHE IN DER SIE-MITHILFE VON VISUELLER PROGRAMMIERUNG BELIEBIGE ABLAEUFE VON KRYPTOOPERA-`

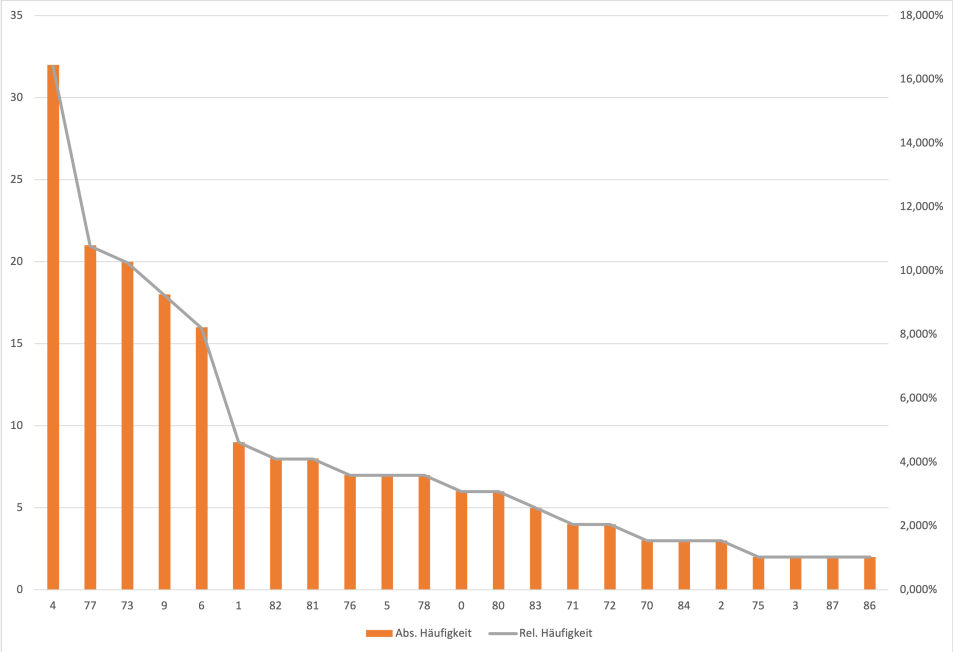

Abbildung 8: Häufigkeit der Zahlen in Abbildung 6

TIONEN ANIMIEREN UND AUSPROBIEREN KOENNEN. Das zum Verschlüsseln verwendete Straddling-Checkerboard ist in Abbildung 9 dargestellt.

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

Abbildung 9: Straddling-Checkerboard (verwendet zum Verschlüsseln des Textes in Abbildung 4)

Die Unizitätslänge für das Straddling-Checkerboard-Verfahren kann folgendermaßen hergeleitet werden: Wie bereits oben angeben, hat der Schlüsselraum eines Straddling-Checkerboards mit 3 Zeilen eine Größe von 28!. Unter der Annahme, dass alle Schlüssel gleich wahrscheinlich sind, kann $H(k)$ durch $\log_2(28!) = 97.94$ bestimmt werden. Für die Bestimmung der Unizitätslänge wird weiter angenommen, dass es sich um

einen deutschen Text handelt, für einen deutschen Text kann eine Entropie von $D = \log 226 - 1,5 = 4,7 - 1,5 = 3,2$ angenommen werden. Somit ergibt sich eine Unizitätslänge von $H(k)/D = 97.94/3,2 \approx 30,60$.

3.3. Ché Guevara

Ernesto „Che“ Guevara (1953-1959) war ein argentinischer Revolutionär und Arzt. Während der kubanischen Revolution tauschte er Nachrichten mit Fidel Castro in verschlüsselter Form aus. Die Nachrichten wurden durch ein Straddling-Checkerboard (3.2) zusammen mit einem OTP vor den Augen Dritter geschützt. [6]

3.3.1. Funktionsweise

	8	2	0	6	4	9	1	3	7	5
	e	s	t	a	d	o	y			
3	b	c	f	g	h	i	j	.	;	,
7	k	l	m	n	ñ	p	q	/	/	
5	r	u	v	w	x	z				

Abbildung 10: Staddling Checkerboard nach dem Vorbild von Ernesto Ché Guevara und Fidel Castro.

Das Straddling-Checkerboard, das Ché Guevara verwendete ist in Abbildung 10 zu sehen. Das verwendete Schlüsselwort lautet ëstadoy". Die Spalten, ebenso wie die Zeilen, sind, im Gegensatz zu den in Kapitel 3.2 beschrieben Straddling-Checkerboards, zufällig nummeriert. Um mehr Zeichen verschlüsseln zu können, wurde eine zusätzliche Zeile ergänzt. Besonders hervorzuheben sind die Zeichen an den Positionen 73 und 77. Das Zeichen an der Position 73 steht dafür, dass alle nachfolgenden Zeichen im Klartext als Zahlen zu interpretieren sind, entsprechend ist das Zeichen an Position 77 die Umkehrung und steht dafür, dass alle nachfolgenden Zeichen wieder als Buchstaben interpretiert werden müssen.

Nachdem der Text durch das Straddling-Checkerboard in eine Zahlenreihe umgewandelt wurde, wird im nächsten Schritt die Zahlenreihe mit einem -Pad um eine zusätzliche

Sicherheitsstufe ergänzt. Bei einem OTP handelt es sich um eine Reihe von zufällig erzeugten Zahlen. Wie der Name schon andeutet, darf jede Zahlenfolge, des OTPs nur einmal verwendet werden, ansonsten kann die Sicherheit des Verfahrens nicht mehr sichergestellt werden.

Verschlüsselung Die Verschlüsselung bei diesem Verfahren läuft äquivalent zum Vorgehen aus Kapitel 3.2 ab. Jedoch ist nach der Verschlüsselung durch das Straddling-Checkerboard die Verschlüsselung noch nicht abgeschlossen. Es wird anschließend noch ein OTP, also eine zufällige Zahlenfolge, die nur einmal verwendet werden darf, benötigt. Das OTP muss dazu mindestens die Länge des Geheimtextes haben, der durch das Straddling-Checkerboard erstellt wurde. Um die Verschlüsselung nach Ché-Guevaras-Verfahren abzuschließen, wird das OTP und die Zahlenreihe, die als Ergebnis der Verschlüsselung durch das Straddling-Checkerboard enstanden ist, miteinander addiert. Dazu werden die einzelnen Zahlen der Zahlenreihe und die einzelnen Zahlen des OTPs mit Modulo 10 zusammen addiert.

Entschlüsselung Die Entschlüsselung läuft ähnlich, wie bei dem Verfahren aus Kapitel 3.2 ab. Allerdings muss, bevor mit dem Entschlüsseln begonnen werden kann, die Anwendung des OTPs umgekehrt werden. Um diese Umkehrung zu realisieren, muss das OTP verfügbar sein. Die Zahlenfolge wird dann vom Chiffrat abgezogen. Wenn das Ergebnis dieser Subtraktion in einer negativen Zahl resultiert, muss angenommen werden, dass im Chiffrat die Zahl um 10 größer ist. Nur so kann ein korrektes Ergebnis erzielt werden. Nachdem die Anwendung des OTPs erfolgreich invertiert wurde, kann mittels des Straddling-Checkerboard-Verfahrens aus Kapitel 3.2 die Zahlenfolge zurück in den Nachrichtentext übersetzt werden.

3.3.2. Beispiel

Wie bereits zu Beginn dieses Kapitels erwähnt wurde, basiert das Verfahren von Ché Guevara auf dem Straddling-Checkerboard-Verfahren. Somit ist die Anwendung identisch zu dem in Kapitel 3.2.2 beschrieben Vorgehen. Daher wird hier nur die Anwendung des OTPs verdeutlicht.

Verschlüsselung In Abbildung 11 wurde das Wort `CRYPTOOL` bereits mit dem Straddling-Checkerboard von Ché Guevara verschlüsselt. Dazu wurde das gleiche Schlüsselwort wie in Abbildung 10 verwendet. Daraus ergibt sich der Geheimtext `325817909972`, dieser wird nun mit dem OTP `713318326459` addiert. Die Addition der einzelnen Zahlen modulo 10 ergeben die Rechenschritte, die in Abbildung 11 dargestellt sind.

```
Klartext:       C  R  Y P  T O O L
Geheimtext:     32 58 1 79 0 9 9 72
One-Time-Pad:   71 33 1 83 2 6 4 59
                ------------------
Ergebnis:       03 81 2 52 2 5 3 21
```

Abbildung 11: Verschlüsselung mit Chè Guevara (Anwendung des One-Time-Pads)

Entschlüsselung Bei der Entschlüsselung wird das OTP vom Geheimtext abgezogen, um das Ergebnis anschließend mittels des Straddling-Checkerboards entschlüsseln zu können. Für das Beispiel in Abbildung 12 wird ein anderer Geheimtext und ein anderes OTP als in Abbildung 11 verwendet: Geheimtext `279430712243` und OTP `9546238133-71`. Bei der Subtraktion des OTPs von dem gegebenen Geheimtext, muss geprüft werden, ob die Geheimtextzahl kleiner ist, als die Zahl des OTPs. Wenn dies der Fall ist, muss die Geheimtextzahl um 10 erhöht werden (Dieser Fall ist mit einem `x` in der Abbildung gekennzeichnet). Die Abbildung 12 zeigt die durchzuführenden Rechenschritte.

```
Geheimtext:     27 94 3 07 1 2 2 43
One-Time-Pad:   95 46 2 38 1 3 3 71
                ------------------
Übertrag:       x   x   xx   x x x
Ergebnis:       32 58 1 79 0 9 9 72
Klartext:       C  R  Y P  T O O L
```

Abbildung 12: Entschlüsselung mit Chè Guevara (Anwendung des One-Time-Pads)

3.3.3. Kryptoanalyse

Da Ché Guevaras Verfahren, wie bereits erläutert, auf dem Verfahren des Straddling-Checkerboards aufbaut, kann dieses Verfahren mit dem gleichen methodischen Vorgehen gebrochen werden. Jedoch muss bei Ché Guevaras Verfahren das OTP in den Fokus gesetzt werden. Durch die Anwendung des OTPs wird das Verfahren für einen Angreifer zu einem unknackbaren Geheimtext, wie David Kahn in seinem Buch „The Codebreakers - The Story of Secret Writing“ [7, pp. 398-400] schreibt. Ché Guevara hat mit seinem Verfahren also eine Chiffre konstruiert, die, wenn sie richtig angewendet wird, nicht gebrochen werden kann.

Würde man versuchen alle möglichen Belegungen des Straddling-Checkerboards von Ché Guevara durchprobieren, so gibt es $37! \approx 1,38 \cdot 10^{43} \approx 2^{144}$ Möglichkeiten für die Zuordnung der Zeichen. Da dieser Schlüsselraum um ein vielfaches größer ist, als

der des in Kapitel 3.2 beschriebenen Straddling-Checkerboards, kann das Straddling-Checkerboard von Ché Guevara ebenfalls nicht in annehmbarer Zeit durch Brute-Force gebrochen werden.

Die Unizitätslänge des Verfahrens ist mit unendlich zu beziffern, da eine unendliche zufällige Zeichenfolge keine Wiederholungen besitzen kann, ist es nicht möglich, dass für ein OTP eine Unizitätslänge bestimmt werden kann.

3.4. Bacon-Chiffre

Bei der Bacon-Chiffre handelt es sich um ein steganographisches Verfahren von Francis Bacon (1561-1626). Allen Buchstaben des Alphabets werden fünfstellige Kombinationen aus a's und b's zugeordnet, um nur zwei Unterscheidungsmerkmale zu haben, die unterschieden werden müssen.

3.4.1. Funktionsweise

Diese Zuordnung ist in Tabelle 2 dargestellt. Zu beachten ist hierbei, dass in der Originalfassung von Francis Bacon die Buchstaben I und J, sowie U und V zusammen gefasst wurden. Daraus ergibt sich ein Alphabet mit 24 Zeichen. [5] [1]

Es werden nicht die einzelnen Codes für die Buchstaben direkt an den Empfänger übermittelt. Stattdessen werden diese Codes genutzt, um sie in anderen Texten zu verstecken, wie bei steganografischen Verfahren üblich. Dazu gibt es verschiedene Vorgehensweisen. Alle Verfahren basieren darauf, dass jeweils dem Buchstaben a und b eine unterschiedliche Bedeutung zugewiesen wird. Dies kann beispielsweise durch zwei unterschiedliche Schriftarten realisiert werden, indem eine der beiden Schriftarten das a repräsentiert und die andere Schriftart das b. Alternativ können die Buchstaben des Textes, indem die Nachricht verborgen werden soll, durch große und kleine Buchstaben ersetzt werden. Außerdem kann das Alphabet in der Hälfte geteilt werden, dass die Buchstaben von A-M werden als a interpretiert und die Buchstaben N-Z repräsentieren ein b, somit wäre das Resultat eine zufällig wirkende Zeichenfolge.

Verschlüsselung Um einen Text mit diesem Verfahren verstecken zu können, muss zunächst jeder Buchstabe des Klartextes mittels der Tabelle 2 in die 5-stellige Darstellung übertragen werden. Somit wird aus dem Text `HALLO` der Ausdruck `aabbb aaaaa ababa ababa abbab`. Diese Zeichenkette kann dann mittels einer oben vorgestellten Technik in einen Text eingebettet werden.

Zeichen	Code	Zeichen	Code	Zeichen	Code
A	aaaaa	I, J	abaaa	R	baaaa
B	aaaab	K	abaab	S	baaab
C	aaaba	L	ababa	T	baaba
D	aaabb	M	ababb	U, V	baabb
E	aabaa	N	abbaa	W	babaa
F	aabab	O	abbab	X	babab
G	aabba	P	abbba	Y	babba
H	aabbb	Q	abbbb	Z	babbb

Tabelle 2: Alphabet der Bacon-Chiffre

Entschlüsselung Um den Klartext aus einem gegebenen Geheimtext zu extrahieren, muss man wissen, worauf man bei dem vorliegenden Text achten muss. Ohne diese Information wird es schwierig, das Schema zu finden, um die Zeichenfolge aus a's und b's rekonstruieren zu können. Hat man Kenntnis darüber, wie die Informationen versteckt wurden, kann man die einzelnen a's und b's extrahieren. Danach muss die Zeichenkette in Gruppen zu je 5 Buchstaben unterteilt werden. Anschließend kann jede Gruppe mittels der Tabelle 2 wieder zurück in einen Klartextbuchstaben übersetzt werden.

3.4.2. Beispiel

Die Funktionsweise der Bacon-Chiffre wird nachfolgend anhand zweier Beispiele verdeutlicht. Um die Verschlüsselung eines Textes zu verdeutlichen, wird hier der Klartext `GEHEIM` verwendet. Für die Entschlüsselung wird der Geheimtext `EJDOKSICMKWJPTJMWOSMRBEWKHTVMSKNUITHQEWE` verwendet.

Verschlüsselung Um die Verschlüsselung mittels der Bacon-Chiffre zu verdeutlichen, wird der Klartext `GEHEIM` verschlüsselt. Dazu werden mittels der Tabelle 2 alle Buchstaben in eine Zwischenform übertragen. Diese Zwischenform substituiert jedes Zeichen mit dem fünfstelligen Code aus der Tabelle, somit lautet die Zwischenform `aabba aabaa aabbb aabaa abaaa ababb`. Anschließend werden die a-Zeichen durch ein zufälliges Zeichen aus der Gruppe der Zeichen von A bis M ersetzt und die b-Zeichen durch ein zufälliges Zeichen aus der Gruppe der Zeichen von N-Z. Somit ergibt sich zum Beispiel folgender Geheimtext `BEZQB IKOFM CAPYS HFOFI DXKGI IOEQX`, aber auch der Geheimtext `EMVUF IBTIL BDOVR EMRDB KXIMH ETFUS` wäre gültig. Abschließend werden die Leerzeichen aus dem Geheimtext entfernt, daraus ergeben sich folgende Geheimtexte: `BEZQBIKOFMCAPYSHFOFIDXKGIIOEQX` bzw. `EMVUFIBTILBDOVREMRDBKXIMHETFUS`.

Entschlüsselung In diesem Beispiel soll der Geheimtext `EJDOKSICMKWJPTJMWOSMRBEWKHTVMSKNUITHQEWE` entschlüsselt werden. Dazu wird zunächst der gesamte Geheimtext in Gruppen zu je 5 Buchstaben zerteilt. Daraus ergibt sich dieser Text: `EJDOK SICMK WJPTJ MWOSM RBEWK HTVMS KNUIT HQEWE`. Da bekannt ist, dass die Zeichen A-M ein a repräsentieren und die Zeichen N-Z ein b repräsentieren, kann durch das Ersetzen der einzelnen Zeichen folgende Zwischenform erstellt werden: `aaaba baaaa babba abbba baaba abbab abbab ababa`. Durch die Ersetzungen aus der Tabelle 2 können die einzelnen Zeichengruppen zu den Buchstaben `C R Y P T O O L` ersetzt werden. Somit ergibt sich der aus dem Geheimtext `EJDOKSICMKWJPTJMWOSMRBEWKHTVMSKNUITHQEWE` der Klartext `CRYPTOOL`.

3.4.3. Kryptoanalyse

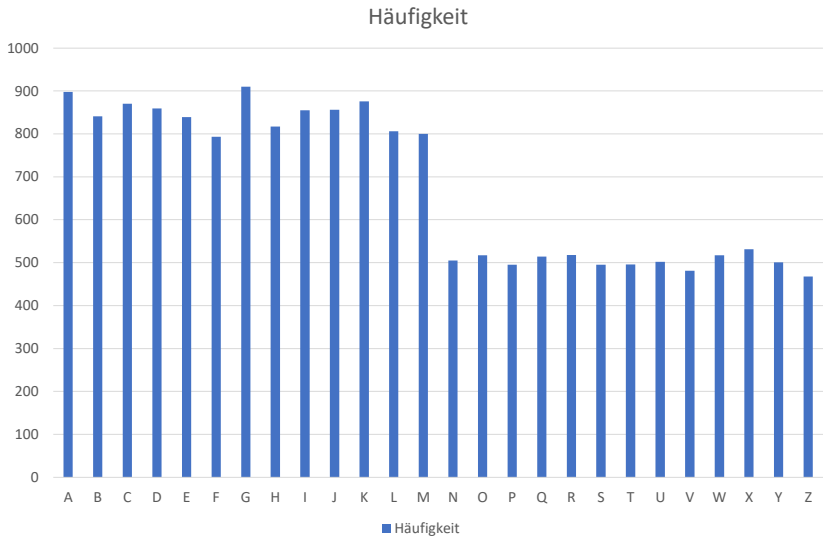

Abbildung 13: Häufigkeit der Zeichen in durch die Bacon-Chiffre kodiertem Text

Francis Bacons Verfahren stellt eine einfache, wenn auch unauffällige Methode zur Steganographie dar, da jede Gruppe von Dingen, die sich in zwei Untergruppen teilen lässt, benutzt werden kann, um darin Informationen zu verstecken. Die hier gezeigte Variante zielt darauf ab, dass Informationen in einer scheinbar zufälligen Zeichenkette versteckt werden. Eine Häufigkeitsanalyse (Kapitel 2.2.3) mit dieser Zeichenkette zeigt jedoch, dass es sich offensichtlich um einen biliteralen Code handelt. Eine beispielhafte Verteilung der Zeichen ist in Abbildung 13 dargestellt. Für die Abbildung wurden die ersten 4 Kapitel des Bellum Gallicum von Cäsar mit dem oben beschrieben Vorgehen codiert und anschließend die Häufigkeit der Zeichen gezählt.

In der von Franzis Bacon erdachten Version ist keine Veränderung der Substitution durch ein Schlüsselwort oder Ähnliches vorgesehen. Somit gibt es für das erdachte Verfahren

nur einen einzigen Schlüssel und zwar die bereits in Tabelle 2 dargestellte Ersetzung. Da das Verfahren mit einem fünfstelligen Code arbeitet, ist es theoretisch möglich, dass bis zu $2^5 = 32$ Zeichen mit diesem Verfahren codiert werden. Weitere Zeichen können hinzugefügt werden, indem der Code statt von 5 Zeichen auf mehr Zeichen erweitert wird. Eine korrekte Decodierung ist dann nur möglich, wenn die Codelänge, die bei der Encodierung verwendet wurde, bekannt ist.

Für die von Francis Bacon entwickelte Version des Verfahrens hat einen Schlüsselraum der Größe 1, da es nur eine fest definierte Substitution gibt. Die Unizitätslänge kann für dieses Verfahren nicht sinnvoll bestimmt werden.

3.5. Josses Cipher

Josses Cipher ist eine Verschlüsselung, die von Major H. D. Josse (1852-1929) erfunden wurde. Das Verfahren wurde in Frankreich für den Deutsch-Französischen Krieg (1870-1871) entwickelt und sollte auf den Schlachtfeldern eingesetzt werden, um Nachrichten an die Befehlshaber gesichert zu übermitteln. Das Verfahren wurde von Rémi Géraud-Stewart und David Naccache im Jahr 2020 untersucht und beschrieben. [4]

3.5.1. Funktionsweise

Ein Schlüsselwort wird bei Josses Cipher verwendet, um eine Ersetzungstabelle zu erzeugen. In der Beschreibung von Josse sind für die Ersetzungstabelle nur kleine lateinische Buchstaben ohne Interpunktion und Sonderzeichen vorgesehen.

Um die Ersetzungstabelle zu erstellen, werden zunächst die Buchstaben des Schlüsselwortes in eine Zeile geschrieben, aber doppelt vorkommende Zeichen nur bei ihrem ersten Vorkommen vermerkt. So wird zum Beispiel aus „ETAT MAJOR GENERAL“ das Schlüsselwort: „ETAMJORGNL“. Anschließend werden die Buchstaben des Alphabetes, die nicht im Schlüsselwort enthalten sind, in alphabetischer Reihenfolge unter die Buchstaben aus der ersten Zeile geschrieben. Somit entsteht die Abbildung 14. Sie ist eine Kombination aus den ersten beiden Tabellen, die Josse verwendete, um sein Verfahren zu verdeutlichen.

Das Alphabet, das Josse für dieses Verfahren verwendete, enthält nicht den Buchstaben „w“, vermutlich, da dieses Zeichen in der französischen Sprache selten vorkommt. [4] Durch das Entfernen des Buchstabens reduziert sich die Anzahl der Buchstaben im Alphabet von 26 auf 25, was die späteren Rechnungen für das Ver- bzw. Entschlüsseln ohne Hilfsmittel erleichtert.

E = 1	T = 4	A = 7	M = 10	J = 13	O = 16	R = 18	G = 20	N = 22	L = 24
B = 2	C = 5	D = 8	F = 11	H = 14	I = 17	K = 19	P = 21	Q = 23	S = 25
U = 3	V = 6	X = 9	Y = 12	Z = 15					

Abbildung 14: Ersetzungstabelle mit dem Schlüsselwort `ETAMJORGNL` [4]

Die so entstandene Tabelle in Abbildung 14 wird spaltenweise gelesen und den Buchstaben entsprechend eine Ziffer beginnend bei 1 zugeordnet.

Verschlüsselung Zunächst wird der zu verschlüsselnde Text in einzelnen Buchstaben getrennt. Die Buchstaben werden mit Abstand hintereinander in einer Zeile aufgeschrieben. Der entsprechende Ausschnitt aus den originalen Dokumenten ist in Abbildung 15 dargestellt. Anschließend wird unter jeden Buchstaben in einer neuen Zeile die Zahl aus dem Schlüsselalphabet (Abbildung 14) geschrieben. In die dritte Zeile werden die Zahlen geschrieben, die man folgendermaßen erhält:

- Der erste Buchstabe wird auch im Geheimtext durch die Zahl repräsentiert, die er durch das Schlüsselalphabet erhalten hat.
- Für alle weiteren Buchstaben wird die Summe über alle vorherigen Buchstaben berechnet und das Ergebnis mit modulo 25 berechnet.

Diese Rechnung wird für alle Zahlen aus der zweiten Zeile durchgeführt. In der vierten Ziele werden die Zahlen mittels des Schlüsselalphabets wieder zurück in Buchstaben gewandelt. Damit der erste Buchstabe im Geheimtext nicht derselbe Buchstabe wie im Klartext ist, zieht man den Wert des Buchstabens von 25 ab. Wenn der erste Buchstabe zum Beispiel U = 3 ist, dann wird die 3 durch $25 - 3 = 22$ ersetzt und entsprechend der 22. Buchstabe im Schlüsselalphabet, also hier das N an erster Stelle verwendet.

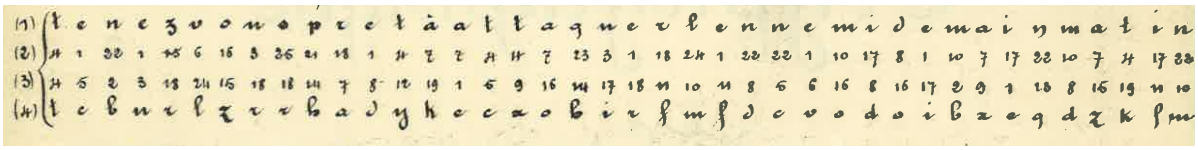

Abbildung 15: Verschlüsselung nach Josses Vorbild (Abbildung nach [4])

Abschließend werden die Buchstaben des erzeugten Schlüsseltextes in Gruppen zu je 5 Buchstaben zusammengefasst und durch ein Trennzeichen verbunden. Wenn die Anzahl der Buchstaben im Schlüssel nicht durch fünf teilbar ist, muss der Text durch zufällig gewählte Buchstaben auf ein vielfaches von 5 aufgefüllt werden. [4]

Entschlüsselung Zunächst wird der verschlüsselte Text in einzelne Buchstaben getrennt. Die Buchstaben werden mit Abstand zueinander aufgeschrieben. Ein Ausschnitt, der den Aufbau der Tabelle für das weiteren Vorgehen zeigt, ist in Abbildung 16 dargestellt. Unter jeden Buchstaben wird der Wert des Buchstabens geschrieben. Unter jeden dieser Werte wird dann eine Zahl geschrieben, die sich wie folgt berechnet:

- Für den ersten Buchstaben wird der Zahlenwert des Buchstabens von der Gesamtzahl der Buchstaben in der Ersetzungstabelle abgezogen.
- Für den zweiten Buchstaben wird die erste Zahl mit der zweiten Zahl addiert. Die Summe wird modulo der Anzahl der Buchstaben in der Ersetzungstabelle gerechnet. In der originalen Beschreibung von Josse wird das mehrfache Subtrahieren von 25 beschrieben. Dies kommt dem Anwenden der Modulo-Operation gleich, wie sie auch im Paper von Géraud-Stewart und Naccache [4] beschrieben wird.
- Für alle weiteren Buchstaben wird der Wert des aktuellen Buchstabens an der nächsten Stelle plus die Anzahl der Buchstaben in der Ersetzungstabelle minus den Zahlenwert des aktuellen Buchstabens gerechnet.

Das Ergebnis dieser Berechnungen steht nun in der dritten Zeile. Abschließend können die Zahlen mittels des Schlüsselalphabets zurück in Buchstaben übersetzt werden. Die so erhaltenen Buchstaben bilden wieder den Klartext. Am Ende des erhaltenen Klartextes können bis zu 4 zufällige Buchstaben stehen, die zum Auffüllen verwendet wurden und keine Bedeutung haben.

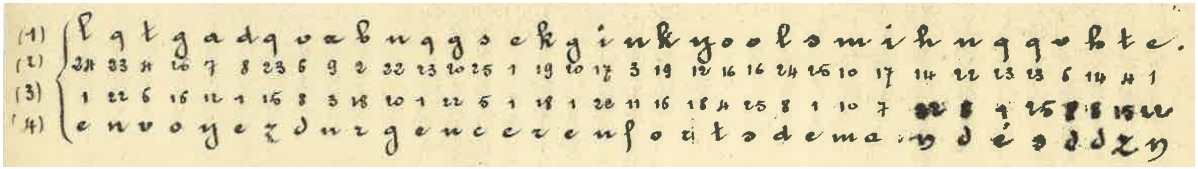

Abbildung 16: Entschlüsselung nach Josses Vorbild (Abbildung nach [4])

3.5.2. Beispiel

Die Funktionsweise von Josses Cipher wird hier durch ein Beispiel für die Ver- bzw. Entschlüsselung verdeutlicht. Dazu wird zunächst der Klartext `FRANKREICH` mit dem Schlüsselwort `GEHEIM` verschlüsselt. Beim Entschlüssel wird ebenfalls das Schlüsselwort `GEHEIM` verwendet, um den Geheimtext `JHLERJQZCO` zu entschlüsseln.

Verschlüsselung Um das Wort `FRANKREICH` mit Josses Cipher zu verschlüsseln, muss zunächst die Ersetzungstabelle gebildet werden. Diese Ersetzungstabelle ist in Tabelle 3

dargestellt. Für die Erstellung dieser Tabelle wurde das Schlüsselwort `GEHEIM` gewählt, da das Schlüsselwort zweimal den Buchstaben `E` enthält, wird das zweite E entfernt. Somit ergibt sich der Schlüssel `GEHIM`.

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

Tabelle 3: Ersetzungstabelle mit dem Schlüsselwort `GEHEIM`

Anschließend werden die Buchstaben der Tabelle spaltenweise durchnummeriert. Daraus ergibt sich die Tabelle 4.

G	A	J	P	U	E	B	K	Q	V	H	C	L
1	2	3	4	5	6	7	8	9	10	11	12	13
R	X	I	D	N	S	Y	M	F	O	T	Z	
14	15	16	17	18	19	20	21	22	23	24	25	

Tabelle 4: Zuordnung der Buchstaben aus der Ersetzungstabelle zu den entsprechenden Zahlen.

Mittels dieser Tabelle kann nun das Wort `FRANKREICH` verschlüsselt werden. Die einzelnen Operationen, die dazu notwendig sind, können der Tabelle 5 entnommen werden. Der Schlüsseltext für den Klartext `FRANKREICH` lautet also `JHLERJQZCO`.

Klartext	Numerische Darstellung	Berechnung	Ergebnis	Geheimtext
F	22	25-22	3	J
R	14	$22 + 14 \mod 25$	11	H
A	2	$11 + 2 \mod 25$	13	L
N	18	$13 + 18 \mod 25$	6	E
K	8	$6 + 8 \mod 25$	14	R
R	14	$14 + 14 \mod 25$	3	J
E	6	$3 + 6 \mod 25$	9	Q
I	16	$9 + 16 \mod 25$	25	Z
C	12	$25 + 12 \mod 25$	12	C
H	11	$12 + 11 \mod 25$	23	O

Tabelle 5: Rechenschritte zum Verschlüsseln des Wortes `FRANKREICH` mittels Josses Cipher

Entschlüsselung Für die Entschlüsselung wird im Folgenden die Ersetzungstabelle aus dem vorherigen Beispiel zur Verschlüsselung mit dem Schlüsselwort `GEHIM` verwendet.

Es soll folgender Schlüsseltext entschlüsselt werden: `JHLERJQZCO`. Die entsprechenden Rechenschritte zur Entschlüsselung sind in Tabelle 6 dargestellt.

Geheimtext	Numerische Darstellung	Berechnung	Ergebnis	Klartext
J	3	$25 - 3$	22	F
H	11	$11 + 3 \mod 25$	14	R
L	13	$13 - 11 \mod 25$	2	A
E	6	$6 - 13 \mod 25$	18	N
R	14	$14 - 6 \mod 25$	8	K
J	3	$3 - 14 \mod 25$	14	R
Q	9	$9 - 3 \mod 25$	6	E
Z	25	$25 - 9 \mod 25$	16	I
C	12	$12 - 25 \mod 25$	12	C
O	23	$2 - 9 \mod 25$	18	H

Tabelle 6: Rechenschritte zum Entschlüsseln des Textes `JHLERJQZCO` mittels Josses Cipher

3.5.3. Kryptoanalyse

Géraud-Stewart und Naccache beschreiben in [4] das Verfahren als einfache Substitutionschiffre, die um zusätzliche Sicherheitsschichten ergänzt wurde. Zu diesen Schichten zählen sie zum einen, dass der erste Buchstabe anders verschlüsselt wird als die restlichen Buchstaben. Zum anderen zählen sie dazu, dass es eine Art von Fehlerfortpflanzung gibt, ein fehlerhafter Buchstabe an einer beliebigen Stelle macht also den gesamten nachfolgenden Text unbrauchbar. Dies wird erreicht, indem über die gesamte Laufzeit der Verbzw. Entschlüsselung eine Summe um den Wert des aktuell verarbeiteten Zeichens inkrementiert wird. Somit ist sichergestellt, dass der Text, sollte er an einer Stelle manipuliert werden, ab dieser Position nicht mehr korrekt entschlüsselt werden kann. Außerdem zählen sie die initiale Permutation des Alphabets in Abhängigkeit des Schlüssels zu einer weiteren Sicherheitsschicht.

Bei einem Chosen-Plaintext-Angriff auf dieses Verfahren (Kapitel 2.2.1) kann der Schlüssel berechnet werden, indem die einzelnen Schritte zur Verschlüsselung bzw. Entschlüsselung zurückgerechnet werden. Da alle Zeichen abhängig von ihrem vorherigen Zeichen sind, kann somit mit wenigen Versuchen ein Teil des Schlüssels rekonstruiert werden. [4]

Der Schlüsselraum für das Verfahren hat eine maximale Größe von $25! \approx 4 \cdot 10^{26} \approx 2^{83{,}68}$ Schlüsseln. Diese Angabe ist jedoch nur ein theoretisches Maximum, da in der Praxis kurze Schlüssel mit 4 bis 8 Zeichen realistischer sind/waren. Durch diese kurzen Schlüssel reduziert sich der mögliche Schlüsselraum massiv. So umfasst der Schlüsselraum für einen

vierstelligen Schlüssel nur $25^4 = 390.625 \approx 2^{19}$ Schlüssel und für einen achtstelligen Schlüssel $25^8 = 152.587.890.625 \approx 2^{38}$ mögliche Schlüssel.

Die Länge des Schlüssels hat auch Auswirkungen auf die Größe der Unizitätslänge. Für die maximale Größe des Schlüsselraumes, also $25! \approx 2^{83,68}$ ergibt sich eine Unizitätslänge von $U = \frac{H(K)}{D} = \frac{83,68}{3,2} = 26,15$. Die weiteren Unizitätslängen für die entsprechenden Schlüssellängen können dem Graphen 17 entnommen werden.

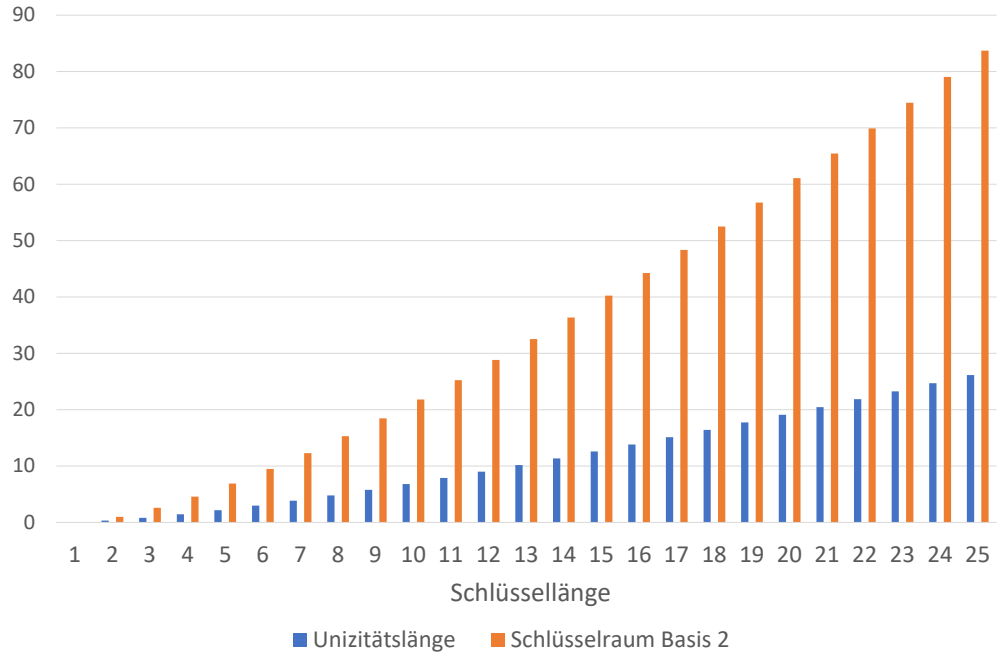

Abbildung 17: Josses Cipher: Unizitätslänge und Schlüsselraum für verschiedene Schlüssellängen

Außerdem heben Géraud-Stewart und Naccache [4] hervor, dass die Länge des Schlüssels und der Schlüssel selbst hergeleitet werden kann. Wenn der Schlüssel zu kurz ist, dann befinden sich die meisten Buchstaben in der Ersetzungstabelle noch an der ursprünglichen Position, somit kann der Schlüssel rekonstruiert werden. Wenn angenommen wird, dass die Transposition durch ein Gitter der Größe N, mit $1 \leq N \leq 25$ gegeben ist, dann kann ein Zeichen nur um den Offset N von seiner richtigen Position entfernt sein.

Ein Ansatz, um dieses Verfahren zu brechen, wird (während diese Arbeit entsteht) von dem israelischen Kryptologen George Lasry entwickelt. Sein Angriff nutzt Simulated Annealing, das wiederum auf dem Hillclimbing-Verfahren aufbaut. Beim Hillclimbing-Verfahren wird ein zufälliger Schlüssel gewählt und mit diesem zufälligen Schlüssel ein

gegebener Geheimtext entschlüsselt und anschließend das Ergebnis bewertet (s. Kapitel 2.2.1 - Ciphertext-Only-Angriff). Im nächsten Schritt wird versucht, den Schlüssel durch das Ersetzen von Zeichen oder Vertauschen von Zeichen so zu verändern, dass mit dem entschlüsselten Geheimtext ein besserer Wert erreicht wird.

Der Unterschied zwischen Hillcliming und Simulated Annealing liegt darin, dass beim Simulated Annealing auch Ergebnisse akzeptiert werden, bei denen sich die Bewertung verschlechtert. Die akzeptierte Abweichung ist dabei zu Beginn sehr groß und wird mit jedem Schritt verringert. Dadurch kann eine bessere Annäherung an das Ergebnis erzielt werden. Lasry verwendet bei seinem Angriff Tetragramme, da er mit Trigrammen bzw. Pentagrammen wesentlich schlechtere Ergebnisse erzielte. [11]

Im Rahmen dieser Arbeit wurde eine Analysekomponente für die Josse-Cipher entwickelt (Kapitel 4.6). Diese Komponente bietet die Möglichkeit, Geheimtexte mit dem Simulated Annealing-Verfahren zu brechen. Für die Bewertung des entschlüsselten Klartextes wurden Tetragramme verwendet, da diese laut Lasry [11] die beste Erfolgschance boten.

Die Erfolgsrate des Angriffs in Relation zur Länge des Schlüssels kann der Abbildung 18 entnommen werden. Für die Bewertung des Angriffs wurde jeweils 1000 mal ein Geheimtext entschlüsselt, der mit der entsprechenden Schlüssellänge verschlüsselt wurde. Für die Schlüssel wurden englische Begriffe verwendet, die keine doppelten Zeichen enthalten. Eine Ausnahme bildet die Schlüssellänge 16, da es in der englischen Sprache kein Wort mit 16 Zeichen gibt, in dem kein Buchstabe doppelt vorkommt: Daher wurde hier eine zufällige Zeichenkette gewählt. Die Schlüsselwörter, die für die Verschlüsselung verwendet wurden, sind in Tabelle 7 aufgelistet.

Es wurde für jede Schlüssellänge ein Klartext in deutscher Sprache mit 242 Zeichen zunächst mit dem entsprechenden Schlüssel verschlüsselt und dann durch den Simulated-Annealing-Angriff entschlüsselt. Den entsprechenden Klartext und die ersten fünf Geheimtexte finden Sie im Anhang (Anhang A). Dazu wurde ein CPX-51 vServer mit 16 CPU-Kernen des Anbieters Hetzner[4] verwendet. Dieser benötigte für die Berechnung aller 16.000 Simulated-Annealing-Angriffe etwa 16,5 Stunden.

3.6. Chaocipher

Chaocipher ist eine Verschlüsselung, die von John F. Byrne (1880-1960) im Jahre 1918 entworfen wurde. Sein ganzes Leben lang versuchte Byrne seine Verschlüsselung an die Regierung der USA zu verkaufen, da er aber die Funktionsweise geheim hielt, blieb dies ohne Erfolg. Seine Familie spendete 2010 seine Unterlagen an das National Cryptologic

[4] https://www.hetzner.com/cloud

Schlüssellänge	Schlüssel	Schlüssellänge	Schlüssel
1	d	9	jumboizes
2	an	10	washingtoz
3	one	11	atmospheric
4	chat	12	ambidextrous
5	secrt	13	documentarily
6	friday	14	dermatoglyphic
7	another	15	dermatoglyphics
8	absolute	16	ypbkjzkxjgiowyvb

Tabelle 7: Simulated Annealing: Verwendete Schlüssel verschiedener Länge

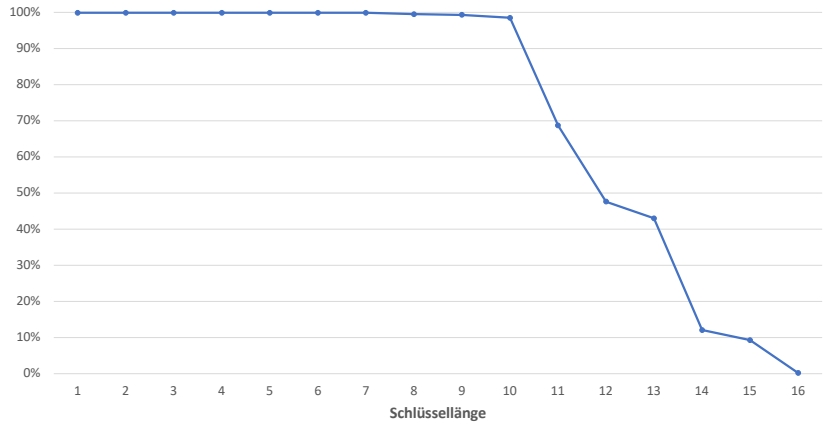

Abbildung 18: Erfolgsrate des implementierten Simulated-Annealing-Angriffs

Museum in den USA, wodurch die Funktionsweise des Verfahrens erstmals öffentlich verfügbar wurde. [15] Byrne entwarf das Verfahren in Form einer Maschine, jedoch wurde sein Entwurf nie realisiert.

3.6.1. Funktionsweise

Diese Maschine sollte aus zwei Scheiben bestehen, auf den Rändern der Scheiben sollten jeweils die Buchstaben des Alphabets in einer zufälligen Reihenfolge angeordnet sein. Die Buchstaben sollten nicht fest montiert sein, sondern frei bewegbar sein, sodass ihre Position vertauscht werden kann. Am äußeren Rand der Scheiben befinden sich Zähne wie bei einem Zahnrad. Die Scheiben sind so zueinander angeordnet, dass durch die Drehung einer Scheibe sich die andere Scheibe in die entgegengesetzte Richtung dreht. Die linke Scheibe repräsentiert das Geheimalphabet und die rechte Scheibe ist für das Klartextalphabet zuständig. Da diese Maschine niemals gebaut wurde, hat ein Erbe von Byrne die Maschine aus Pappe nachgebaut. Dieser Nachbau ist in Abbildung 19 dargestellt.

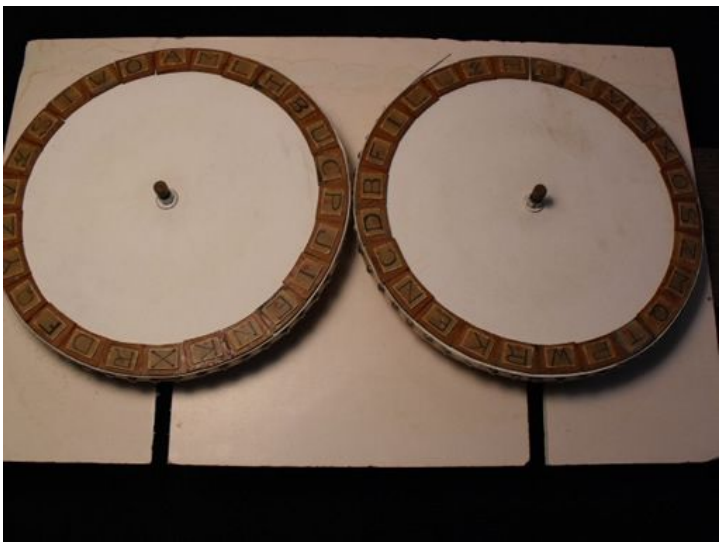

Abbildung 19: Nachbau der Chaocipher-Maschine (Bild mit freundlicher Genehmigung der National Cryptologic Foundation, 2010)

Weiterhin definiert Byrne zwei Positionen auf den Scheiben. Zum einen den Zenit und zum anderen den Nadir. Der Zenit befindet sich an der 1. Position der Scheibe und der Nadir befindet sich auf der gegenüberliegenden Seite, also an Position 14. Diese Positionen spielen bei der Permutation der Zeichen auf der Scheibe eine wichtige Rolle.

Die Grundidee des Verfahrens ist, dass nach jedem ver- bzw. entschlüsselten Zeichen die Position der Zeichen auf den Scheiben permutiert wird. Die Permutation hängt dabei von den bereits ver- bzw. entschlüsselten Buchstaben ab. Für die Permutation des Verfahrens sind nicht unbedingt die von John F. Byrne beschriebenen Scheiben notwendig. Abbildung 20 zeigt eine vereinfachte Schreibweise. Diese Schreibweise ist intuitiver und lässt sich kompakter darstellen. Dabei steht das Symbol `Z` für den Zenit und das Symbol `N` für den Nadir.

```
                 Z           N
Linke Scheibe : TUVWXYZABCDEFGHIJKLMNOPQRS
Rechte Scheibe: ABCDEFGHIJKLMNOPQRSTUVWXYZ
                --------------------------
Position:       12345678911111111112222222
                          01234567890123456
```

Abbildung 20: Vereinfachte Darstellung der Chaocipher

Die erste Zeile in Abbildung 20 stellt die linke Scheibe, also das Geheimalphabet dar. In der zweiten Zeile werden die Buchstaben des Klartextalphabets aufgelistet. Die Positionen unter der Trennlinie werden von oben nach unten gelesen werden und stellen

die Positionen auf der Scheibe, beginnend bei eins, dar. Die besonderen Positionen Zenit und Nadir sind hier durch die Zeichen „Z“ und „N“ markiert.

Verschlüsselung Ein Buchstabe wird verschlüsselt, indem der Buchstabe, der verschlüsselt werden soll, auf der rechten Scheibe an die Position des Zenits gedreht wird. Durch die Drehung der rechten Scheibe dreht sich die linke Seite entgegengesetzt. Der Buchstabe, der am Ende auf der linken Scheibe an der Position des Zenits ist, stellt den verschlüsselten Buchstaben da.

Nach jedem Buchstaben, der verschlüsselt wurde, müssen die Scheiben permutiert werden. Dazu wird folgendermaßen vorgegangen:

Permutation der linken Scheibe Die Permutation der linken Scheibe wird in vier Schritten durchgeführt. Zuerst wird das Zeichen, das gerade verschlüsselt wurde, auf die Position des Zenits gedreht (Position 1). Anschließend wird das Zeichen, das sich rechts des Zenits befindet (Zenit + 1) entfernt. Danach werden alle Zeichen rechts des Zenits (Zenit + 2) bis einschließlich der Nadir (Zenit + 13) Position im Uhrzeigersinn um eine Position verschoben. Dadurch schließt sich die Lücke rechts des Zenits (Zenit + 1) und eine neue Lücke an der Position des Nadirs (Zenit + 13) entsteht. Zum Schluss wird die Lücke an der Stelle des Nadirs (Zenit + 13) gefüllt, indem das Zeichen, das im ersten Schritt entfernt wurde, an der Stelle des Nadirs (Zenit + 13) wieder eingefügt wird. Die Permutation der linken Scheibe ist damit abgeschlossen.

Permutation der rechten Scheibe Die Permutation der rechten Scheibe wird in fünf Schritten durchgeführt. Im ersten Schritt wird die rechte Scheibe so gedreht, dass sich das Klartextzeichen, das gerade verschlüsselt wurde, an der Position des Zenits befindet (Position 1). Anschließend wird die Scheibe um die Position eines einzelnen Zeichens weiter nach links gedreht. Im dritten Schritt wird der zweite Buchstabe rechts des Zenits entfernt (Zenit + 2), so das an dieser Position eine Lücke entsteht. Im vierten Schritt werden alle Buchstaben nach der neu entstandenen Lücke (Zenit + 3) einschließlich des Nadirs (Zenit + 13) um eine Position nach links verschoben. Nach diesem Schritt entsteht eine Lücke an der Stelle des Nadirs. Im fünften Schritt wird in die Lücke an der Stelle des Nadirs (Zenit + 13) der Buchstabe eingefügt, der im ersten Schritt entfernt wurde. Die Permutation der rechten Scheibe ist damit abgeschlossen.

Entschlüsselung Bei der Entschlüsselung eines verschlüsselten Textes werden die gleichen Permutationen für die linke bzw. rechte Scheibe verwendet wie bei der Verschlüsselung. Der wichtige Unterschied ist jedoch, dass beim Entschlüsseln das Zeichen aus dem

Geheimtext auf der linken Scheibe gesucht und entsprechend zum Zenit gedreht wird. Anschließend wird dieses Zeichen durch den Buchstaben auf der rechten Seite ersetzt.

3.6.2. Beispiel

Das Verfahren Chaocipher wird hier durch ein Beispiel für die Ver- bzw. Entschlüsselung verdeutlicht. Dazu wird zunächst der Klartext `KRYPTOGRAFIE` mit dem Schlüssel verschlüsselt. Dazu wird die Konfiguration der Scheiben aus Abbildung 20 verwendet. Beim Entschlüsseln wird ebenfalls diese Konfiguration verwendet, um den Geheimtext `DJPFIBYLWWDPL` zu entschlüsseln.

Verschlüsselung Die Verschlüsselung startet mit der in Abbildung 20 dargestellten Konfiguration. Um nun den Klartext `KRYPTOGRAFIE` zu verschlüsseln, muss zunächst der erste Buchstabe, also das `K`, auf die Position des Zenit auf der rechten Scheibe gedreht werden. Dieser Zustand ist in Abbildung 21 dargestellt. Das entsprechende Geheimzeichen kann auf der linken Scheibe abgelesen werden. In diesem Fall handelt es sich um das Zeichen `D`. Nachdem das erste Zeichen verschlüsselt wurde, müssen die Zeichen auf den Scheiben neu angeordnet werden.

```
                  Z            N
Linke Scheibe : DEFGHIJKLMNOPQRSTUVWXYZABC
Rechte Scheibe: KLMNOPQRSTUVWXYZABCDEFGHIJ
                --------------------------
Position:       12345678911111111112222222
                          01234567890123456
```

Abbildung 21: Buchstabe K auf der Position des Zenits

Zunächst wird die linke Scheibe permutiert. Die einzelnen Schritte dazu wurden bereits in Kapitel 3.6.1 erklärt, deshalb wird hier auf eine genaue Erläuterung verzichtet. Die praktische Durchführung der einzelnen Schritte ist in Abbildung 22 dargestellt. In Schritt 1 wird das Zeichen `K`, das zuletzt verschlüsselt wurde, auf die Position des Zenit verschoben. In Schritt 2 wurde das Zeichen `L` rechts des Zenits entfernt. Anschließend wurden in Schritt 3 alle Zeichen von Position 3 bis einschließlich der Position 14 um eine Position nach links verschoben. In Schritt 4 wurde die Lücke, die so an der Position des Nadirs entstanden ist, wieder mit dem Buchstaben `L` gefüllt.

Für die Permutation der rechten Scheibe müssen die Schritte durchgeführt werden, die in Abbildung 23 dargestellt sind. In Schritt 1 wird zunächst der Klartextbuchstabe, der gerade verschlüsselt wurde, an die Position des Zenits gedreht. In Schritt 2 wird die Scheibe um eine weitere Position nach links gedreht. Anschließend wird in Schritt 3 das

```
Position:       12345678911111111112222222
                        01234567890123456
                Z            N
Linke Scheibe : DEFGHIJKLMNOPQRSTUVWXYZABC
                -------------------------
Schritt 1:      KLMNOPQRSTUVWXYZABCDEFGHIJ
Schritt 2:      K.MNOPQRSTUVWXYZABCDEFGHIJ
Schritt 3:      KMNOPQRSTUVWX.YZABCDEFGHIJ
Schritt 4:      KMNOPQRSTUVWXLYZABCDEFGHIJ
```

Abbildung 22: Chaocipher (Permutation der linken Scheibe)

Zeichen `N` an Position 3 entfernt. Die Zeichengruppe `OPQRSTUVWXY` wird in Schritt 4 nach links verschoben, um die Lücke an der dritten Position zu schließen. Im letzten Schritt wird das Zeichen `N` in die verbleibende freie Stelle an der Position des Nadirs eingefügt (Zenit + 13).

```
Position:       12345678911111111112222222
                        01234567890123456
                Z            N
Rechte Scheibe: KLMNOPQRSTUVWXYZABCDEFGHIJ
                -------------------------
Schritt 1:      KLMNOPQRSTUVWXYZABCDEFGHIJ
Schritt 2:      LMNOPQRSTUVWXYZABCDEFGHIJK
Schritt 3:      LM.OPQRSTUVWXYZABCDEFGHIJK
Schritt 4:      LMOPQRSTUVWXY.ZABCDEFGHIJK
Schritt 5:      LMOPQRSTUVWXYNZABCDEFGHIJK
```

Abbildung 23: Chaocipher (Permutation der rechten Scheibe)

Die weiteren Schritte laufen analog zu dem zuvor beschriebenen Vorgehen ab, deshalb sind in der Abbildung 8 nur die Zustände der Scheiben aufgelistet. Die Zustände beschreiben jeweils die Scheiben, nachdem ein Klartextzeichen verschlüsselt wurde und die Permutierung der Scheiben abgeschlossen wurden. Nachdem das Wort `KRYPTOGRAFIE` verschlüsselt wurde, ergibt sich der Geheimtext `DJPFIBUSWSDM`.

Entschlüsselung Für die Entschlüsselung soll die in Abbildung 20 dargestellten Konfiguration verwendet werden. Um den Geheimtext `DJPFIBYLWWDPL` zu entschlüsseln, muss zunächst das erste Zeichen, also das *D*, auf die Position des Zenits auf der linken Scheibe gedreht werden. Dieser Zustand ist in der Abbildung 21 dargestellt. Das entsprechende Klartextzeichen kann auf der rechten Scheibe abgelesen werden. In diesem Fall handelt es sich um das Zeichen `K`. Nachdem das erste Zeichen entschlüsselt wurde, müssen

Linke Scheibe	Rechte Scheibe
DFGHIJKLMNOPQERSTUVWXYZABC	LMOPQRSTUVWXYNZABCDEFGHIJK
JLMNOPQERSTUVKWXYZABCDFGHI	STVWXYNZABCDEUFGHIJKLMOPQR
PERSTUVKWXYZAQBCDFGHIJLMNO	NZBCDEUFGHIJKALMOPQRSTVWXY
FHIJLMNOPERSTGUVKWXYZAQBCD	QRTVWXYNZBCDESUFGHIJKALMOP
ILMNOPERSTGUVJKWXYZAQBCDFH	VWYNZBCDESUFGXHIJKALMOPQRT
BDFHILMNOPERSCTGUVJKWXYZAQ	PQTVWYNZBCDESRUFGXHIJKALMO
UJKWXYZAQBDFHVILMNOPERSCTG	XHJKALMOPQTVWIYNZBCDESRUFG
STGUJKWXYZAQBCDFHVILMNOPER	UFXHJKALMOPQTGVWIYNZBCDESR
WYZAQBCDFHVILXMNOPERSTGUJK	LMPQTGVWIYNZBOCDESRUFXHJKA
SGUJKWYZAQBCDTFHVILXMNOPER	XHKALMPQTGVWIJYNZBOCDESRUF
DFHVILXMNOPERTSGUJKWYZAQBC	JYZBOCDESRUFXNHKALMPQTGVWI

Tabelle 8: Chaocipher (Zwischenzustände beim Verschlüsseln von `KRYPTOGRAFIE`)

die Zeichen auf den Scheiben neu angeordnet werden. Dazu kann dem Vorgehen aus dem Beispiel für die Verschlüsselung gefolgt werden. Die weiteren Zwischenzustände der Scheiben nach der Permutation der Zeichen, die beim Entschlüsseln entstehen, können der Tabelle 9 entnommen werden. Nachdem alle Zeichen entschlüsselt wurden, erhält man den Klartext `KRYPTOANALYSE`.

Linke Scheibe	Rechte Scheibe
DFGHIJKLMNOPQERSTUVWXYZABC	LMOPQRSTUVWXYNZABCDEFGHIJK
JLMNOPQERSTUVKWXYZABCDFGHI	STVWXYNZABCDEUFGHIJKLMOPQR
PERSTUVKWXYZAQBCDFGHIJLMNO	NZBCDEUFGHIJKALMOPQRSTVWXY
FHIJLMNOPERSTGUVKWXYZAQBCD	QRTVWXYNZBCDESUFGHIJKALMOP
ILMNOPERSTGUVJKWXYZAQBCDFH	VWYNZBCDESUFGXHIJKALMOPQRT
BDFHILMNOPERSCTGUVJKWXYZAQ	PQTVWYNZBCDESRUFGXHIJKALMO
YAQBDFHILMNOPZERSCTGUVJKWX	LMPQTVWYNZBCDOESRUFGXHIJKA
LNOPZERSCTGUVMJKWXYAQBDFHI	ZBDOESRUFGXHICJKALMPQTVWYN
WYAQBDFHILNOPXZERSCTGUVMJK	LMQTVWYNZBDOEPSRUFGXHICJKA
WAQBDFHILNOPXYZERSCTGUVMJK	MQVWYNZBDOEPSTRUFGXHICJKAL
DHILNOPXYZERSFCTGUVMJKWAQB	NZDOEPSTRUFGXBHICJKALMQVWY
PYZERSFCTGUVMXJKWAQBDHILNO	TRFGXBHICJKALUMQVWYNZDOEPS

Tabelle 9: Chaocipher (Zwischenzustände beim Entschlüsseln von `KRYPTOANALYSE`)

3.6.3. Kryptoanalyse

Der Schlüssel für die Chaocipher wird durch die initiale Position der Zeichen auf den Scheiben bestimmt. Pro Scheibe stehen 26 Positionen für die Zeichen zur Verfügung. Somit ergeben sich pro Scheibe 26! Möglichkeiten, um die Zeichen anzuordnen. Da bei

der Chaocipher jedoch zwei Scheiben zum Einsatz kommen, gibt es $(26!)^2$ Möglichkeiten, um die Zeichen auf beiden Scheiben anzuordnen. Somit würde sich ein theoretischer Schlüsselraum von etwa 2^{177} möglichen Schlüsseln ergeben. Jedoch sind ein Teil dieser möglichen Schlüssel äquivalent zueinander, da sie durch eine Drehung der beiden Scheiben erzeugt werden können. Deshalb muss die Anzahl der Möglichkeiten noch durch die Anzahl der Positionen geteilt werden. Somit ergibt sich ein Schlüsselraum von $\frac{26!^2}{26} \approx 2^{173}$ Möglichkeiten.

Die Unizitätslänge für das Chaocipher-Verfahren berechnet sich wie folgt: Jedes der 26 Zeichen des Alphabets kann $\log 226 \approx 4,7$ Bits an Informationen darstellen, somit ist $D = 4,7$. $H(k)$ berechnet sich für die Chaocipher durch $H(k) = \log_2 \frac{26!^2}{26} \approx 172.063$. Somit berechnet sich die Unizitätslänge durch $U = \frac{H(k)}{D} = \frac{\log_2 \frac{26!^2}{26}}{\log_2 26} \approx 36.605$.

Bis 2010 konzentrierte sich die Kryptoanalyse des Chaocipher-Verfahrens lediglich auf die Funktionsweise des Algorithmus. Da diese Funktionsweise jedoch durch die Erben des Erfinders im Jahre 2010 für die Öffentlichkeit zugänglich gemacht wurden, veränderte sich der Fokus der Forschung dahin, das Verfahren zu brechen, wie in einem im Jahre 2016 im Journal Cryptologia durch Lasry, Rubin, Kopal und Wacker veröffentlichten Artikel[12] beschrieben wird. In dieser Veröffentlichung werden außerdem mehrere Angriffe gegen das Chaocipher-Verfahren erläutert. Diese werden in dieser Arbeit nur grob beschrieben.

Known-Plaintext-Angriff Schon früh in der Erforschung der Chaocipher wurde ein Known-Plaintext-Angriff gegen das Chaocipher-Verfahren entwickelt. Dabei wird versucht, durch Backtracking die initiale Position der Zeichen auf den Scheiben zu erschließen. Für den Angriff werden die beiden Scheiben zu Beginn leer gelassen. Es wird zu Beginn versucht, durch Zufälliges einfügen und Anschließendes ausschließen aller Möglichkeiten, die zu einem Konflikt führen, die initiale Zeichenfolge auf den Scheiben wiederherzustellen. Dabei ist es wichtig, dass bei einem Konflikt zwischen den wiederhergestellten Scheiben und dem Schlüssel- bzw. Klartext zurück zu der letzten gültigen Konfiguration gesprungen wird und von dort aus die nächsten Möglichkeiten ausprobiert werden.

Ein guter Einstiegspunkt für dieses Verfahren sind mehrere Vorkommen des gleichen Zeichens im Schlüssel- bzw. Klartext, somit können bereits zu Beginn möglichst viele Position bestimmt werden. Nach jedem bestimmten Zeichen müssen die Permutationen (Kapitel 3.6.1) der Scheiben durchgeführt werden. Somit können, bei ausreichender Länge des Schlüssel- bzw. Klartextes, die initialen Positionen der Zeichen auf den Scheiben rekonstruiert werden. Eine genauere Beschreibung des Verfahrens wird in dem bereits erwähnten Paper [12] beschrieben.

Ein neuerer Ansatz, um Geheimtexte, die mit dem Chaocipher-Verfahren verschlüsselt wurden, zu brechen, ist der Einsatz von Hillclimbing-Algorithmen, wie sie in [12] beschrieben werden. Mittels Hillclimbing sind sowohl Ciphertext-Only-Angriffe als auch Known-Plaintext-Angriffe möglich. Die dazu verwendeten Hillclimbing-Algorithmen unterscheiden sich in der Art, wie der Score berechnet wird und ob Transformationen nur für eine Scheibe oder beide Scheiben durchgeführt werden. Es gibt drei zulässige Transformationen, die angewendet werden können. Zum einen können zwei Zeichen im linken oder im rechten Alphabet vertauscht werden oder es können zwei Zeichen gleichermaßen im linken und im rechten Alphabet vertauscht werden.

Für die Berechnung des Scores wurden zwei verschiedene Methoden angewandt. Die erste Methode, die für den Ciphertext-Only-Angriff verwendet wurde, ist der Koinzidenzindex. Durch ihn konnte ein Divide-and-Conquer-Angriff ermöglicht werden, da ein Klartext, der durch einen mutmaßlichen Schlüssel entschlüsselt wurde, nur von der initialen Belegung der linken Scheibe abhängt. Die zweite Methode, die für den Known-Plaintext-Angriff verwendet wurde, ist die sogenannte „Plaintext-Ciphertext Weighted Match“-Methode. Diese Methode basiert darauf, dass eine Summe über alle Scores der einzelnen Zeichen berechnet wird. Somit errechnet sich der gesamte Score für den entschlüsselten Klartext, der für den Hillclimbing-Algorithmus verwendet werden kann. Die einzelnen Zeichen erhalten ihren Score durch die Formel $\frac{1000}{N}$, wobei N die Position des Zeichens im Text bezeichnet. Wenn ein Klartextzeichen mit dem bekannten Klartext übereinstimmt, dann bekommt das Zeichen an dieser Stelle den entsprechenden Wert, ansonsten bekommt das Zeichen den Wert 0. Durch diese Formel bekommen Lösungen, die am Anfang des Klartextes bereits von der geforderten Lösung abweichen, eine sehr viel niedrigere Gewichtung als solche, die am Ende erst von der geforderten Lösung abweichen. Somit kann der Hillclimbing-Algorithmus mit wenigen Klartextbeispielen eine Lösung errechnen.

3.7. Vergleich der Verfahren

Alle hier erläuterten Verfahren sind händisch durchführbar, trotzdem unterscheidet sich das Sicherheitsniveau, das diese Verfahren bieten, zum Teil sehr stark. Betrachtet man beispielsweise, wie resistent die vorgestellten Verfahren hinsichtlich eines Brute-Force-Angriffes sind, so stellt man schnell fest, dass das sicherste Verfahren gegen diesen Angriff das Ché-Guevara-Verfahren ist, da hier der Schlüsselraum eine unendliche Größe hat. Er kann deshalb niemals vollständig durchsucht werden. Diese Eigenschaft ist dem OTP zu verdanken. Da Ché Guevara sein OTP jedoch mehrmals verwendete [6], konnten seine Nachrichten entschlüsselt werden. Lässt man jedoch das OTP außer Acht und bewertet nur das Sicherheitsniveau des Straddling-Checkerboards, das dem Verfahren von Ché Guevara zugrunde liegt, stellt man fest, dass hier der Schlüsselraum $28! \approx 2^{98}$ mögliche Schlüssel umfasst. Dieser Schlüsselraum genügt bereits, um gegen einen Brute-

Force-Angriff sicher zu sein. Jedoch kann das Verfahren durch einen Häufigkeitsangriff gebrochen werden, wie in Kapitel 3.2.3 gezeigt wurde.

Das Chaocipher-Verfahren ist das jüngste der hier vorstellten Verfahren. Dieses Verfahren besitzt den zweitgrößten Schlüsselraum und damit auch die zweitgrößte Unizitätslänge aller hier vorgestellten Verfahren. Es ist durchaus denkbar, dass in den nächsten Jahren weitere Angriffe gegen dieses Verfahren entdeckt werden, da es erst im Jahre 2010 öffentlich wurde.

Josses Cipher wurde zwar um 1889 von Major H. D. Josse entwickelt, jedoch wurde dieses Verfahren erst im Jahre 2020 in einem wissenschaftlichen Paper öffentlich beschrieben. Es bietet maximal einen Schlüsselraum mit 2^{38} möglichen Schlüsseln und besitzt eine maximale Unizitätslänge von $26,15$. Durch das interne Aufaddieren der bereits verschlüsselten Zeichen bietet das Verfahren einen guten Schutz gegen bekannte statistische Angriffe.

Das Bacon-Verfahren wurde als steganografisches Verfahren entwickelt. Mit ihm sollten Texte versteckt werden. Jedoch kann das Verfahren einfach entdeckt werden, indem die Häufigkeit der Zeichen analysiert wird. Die ACA beschreibt dieses Verfahren auf ihrer Internetseite [1] als einfache Substitution, bei der das Ergebnis in einen Text eingebunden wird. Dabei ist die Substitution feststehend. Somit gibt es für dieses Verfahren keinen Schlüssel.

Eine besondere Rolle bei den hier vorgestellten Verfahren spielt der T9-Code. Anders als bei den anderen Verfahren geht es hierbei nicht darum, einen Text zu verschlüsseln, sondern lediglich darum, die Interaktion zwischen Mensch und Mobiltelefon zu verbessern. Daher besitzt dieses Verfahren keinen Schlüsselraum und keine Unizitätslänge. Es wurde jedoch in diese Arbeit aufgenommen, da der T9-Code häufig verwendet wird, um einen Klartext vor Dritten zu verstecken und weil es einen interessanten Ansatz zur Dekodierung erfordert.

	Zeitraum	**Schlüsselraum**	**Unizitätslänge**
T9-Code (Mobile phone code)	2000er	-	-
Straddling-Checkerboard	$1555 - 1980$	$28! \approx 2^{98}$	$30,60$
Ché Guevara	1960er	∞	∞
Bacon	1561-1626	-	-
Josses Cipher	≈ 1889	$25^8 \approx 2^{38}$	$26,15$
Chaocipher	1918	$\frac{26!^2}{26} \approx 2^{173}$	$36,605$

Tabelle 10: Vergleich der Metriken der vorgestellten Verfahren

4. Design und Implementierung der Verfahren als CT2-Komponenten

Dieses Kapitel beschäftigt sich mit dem Design, der Implementierung und Visualisierung der sechs Verfahren, die in Kapitel 3 vorgestellt wurden. Da das Verfahren des Straddling-Checkerboards (Kapitel 3.2) und das Verfahren von Ché Guevara (Kapitel 3.3) beide auf die gleiche Ersetzungstabelle zurückgeführt werden können, sind diese beiden Verfahren hier in einem Kapitel (Kapitel 4.3) zusammengefasst worden.

4.1. Entwicklung von Plugins für CrypTool2

Die Funktionsweise von CT2-Plugins wurde bereits in anderen Arbeiten ausführlich beleuchtet, daher wird an dieser Stelle auf eine ausführliche Erläuterung verzichtet. [17, S. 28–32] [19, S. 11–17] Es werden nur die wichtigsten Elemente der Plugin-Implementierung beleuchtet, um die Entscheidungen bei der Implementierung nachvollziehbar zu machen. Detaillierte Informationen über die Implementierung von Plugins werden auf der Internetseite von CT2 zur Verfügung gestellt.[5]

Aufbau eines Plugins Die Software CT2 wird in der Sprache C# entwickelt und kann durch Plugins um zusätzliche Funktionalität erweitert werden. Um ein solches Plugin zu erstellen, gibt es eine Vorlage für die Software Visual Studio 2019. Diese Vorlage beinhaltet zwei Klassen, die notwendig für ein Plugin in CT2 sind. Eine Klasse implementiert das Interface *ICrypComponent* und die andere Klasse implementiert das Interface *ISettings*. Die Klasse, die das Interface *ICrypComponent* implementiert, wird per Konvention nach dem Namen des Plugins benannt und beinhaltet die Logik, die für die Funktion der Komponente notwendig ist. Die Klasse, die das Interface *ISettings* implementiert, wird per Konvention nach dem Schema *<PluginName>Settings* benannt und wird verwendet, um die Einstellungen der Chiffre zu definieren. Der Code für das Plugin, sowie alle weiteren Dateien werden beim Compilieren in ein Assembly kopiert. Dieses Assembly kann von CT2 geladen werden. Beim Laden wird das Assembly mittels Reflection nach dem bereits erwähnten *ICrypComponent*-Interface durchsucht und entsprechend als Komponente geladen.

ICrypComponent Das Interface *ICrypComponent* definiert Methoden, die für den Lebenszyklus einer Komponente notwendig sind. Für diese Arbeit ist hauptsächlich die *Execute*-Methode von Bedeutung. Jedoch wurde für alle Plugins die *Execute*- und die

[5] https://www.cryptool.org/en/ct2/resources

Stop-Methode implementiert, um ein unterbrechen der Ausführung zu realisieren. Dazu wird nach jedem Schritt in der *Execute*-Methode eine Boolean-Variable überprüft. Wenn die Variable den Wert *false* hat, wird die Ausführung unterbrochen. Die Variable wird mit dem Wert *true* initialisiert und durch die *Stop*-Methode auf *false* gesetzt.

Die *Execute*-Methode wird immer dann aufgerufen, wenn die Ausführung des Plugins erforderlich ist. Dies ist insbesondere dann der Fall, wenn neue Werte für die Komponente vorliegen. Der Eingang für neue Werte wird auch in der Klasse definiert, die das Interface *ICrypComponent* implementiert, ebenso, wie die Ausgänge einer Komponente. Außerdem definiert das Interface eine Eigenschaft, die die Visualisierung der Komponente bereitstellt.

ISettings Das Interface *ISettings* erbt das Interface *INotifyPropertyChanged*, welches ein Event definiert. Dieses Event muss aufgerufen werden, sobald eine Einstellung der Komponente verändert wird.

Visualisierung Eine Visualisierung kann von einem Plugin optional bereitgestellt werden. Sie wird in CT2 dazu verwendet, das Verfahren für den Nutzer innerhalb der Komponente grafisch aufzubereiten. Die Visualisierung basiert auf einer Windows Presentation Foundation (WPF) Komponente. Das WPF bietet die Möglichkeit mittels Extensible Application Markup Language (XAML), vektorbasierte Benutzeroberflächen erstellen zu können.

Internationalisierung CT2 bietet die Möglichkeit Komponenten in verschiedene Sprachen zu übersetzen. Dazu müssen innerhalb des Plugins sogenannte Resource-Dateien angelegt werden. Diese Dateien enthalten für ein definiertes Schlüsselwort die entsprechende Übersetzung. Das Schlüsselwort wird während der Entwicklung im Code der Komponente anstelle eines Textes verwendet. Zur Laufzeit kann CT2 die Resource-Datei laden, die für die Sprache des Benutzers hinterlegt wurde, und das Schlüsselwort durch den Text aus der Resource-Datei ersetzen. Alle Komponenten, die im Rahmen dieser Arbeit entwickelt wurden, sind in deutscher und englischer Sprache Verfügbar.

Templates Templates werden in CT2 auf dem Startbildschirm angezeigt und sollen dem Benutzer einen einfacheren Einstig in CT2 bieten. Sie enthalten zum Beispiel einen Ver- und Entschlüsselungsprozess mittels einer Chiffre. Für alle Verfahren, die im Rahmen dieser Arbeit umgesetzt wurden, wurden Templates erstellt, die jeweils den Ver- und Entschlüsselungsprozess mit diesem Verfahren zeigen. Bei der Erstellung der Templates wurde außerdem auf die Internationalisierung geachtet, dadurch sind alle Templates in deutscher, englischer, chinesicher und russischer Sprache verfügbar.

4.2. T9-Code

Im nachfolgenden Abschnitt werden die Anforderungen an die T9-Code-Komponente, sowie der Aufbau und der Entwicklungsprozess der Komponente beschrieben. Für die Erläuterung der Funktionsweise der Komponente wurde außerdem ein Template erstellt, das in Abbildung 24 dargestellt ist.

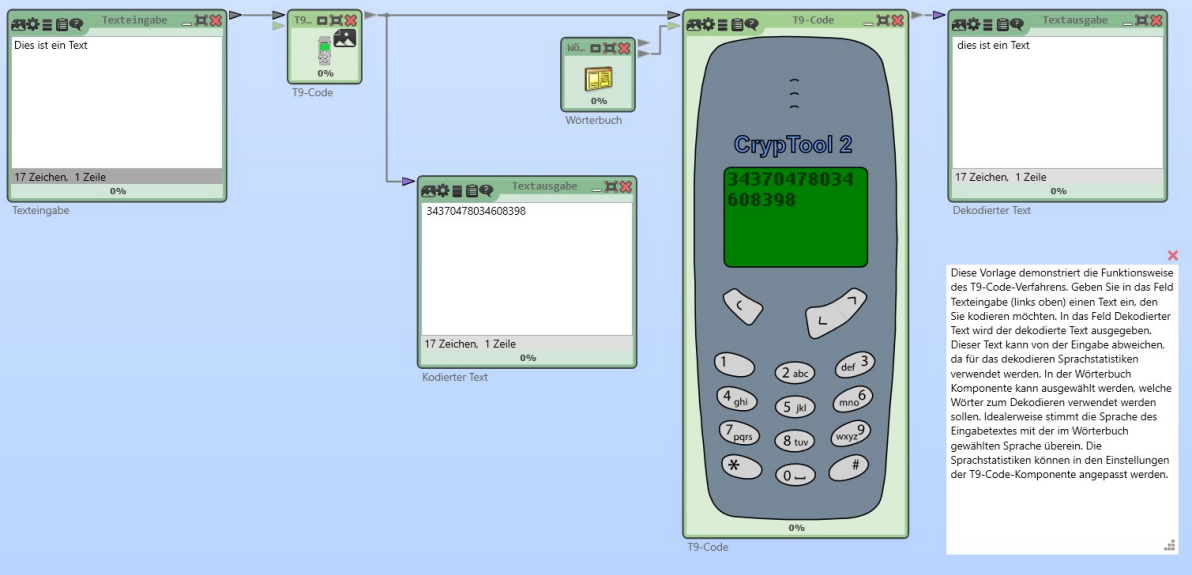

Abbildung 24: Template der T9-Code-Komponente

4.2.1. Anforderungen

Die T9-Code-Komponente soll eine Visualisierung besitzen. Die Visualisierung soll aus einem Mobiltelefon bestehen, das interaktiv bedient werden kann. Das Mobiltelefon soll dazu über eine 12-Tasten-Tastatur verfügen. Außerdem soll die Eingabe des Benutzers auf dem Display des Mobiltelefons erscheinen.

Die Komponente soll über einen Eingang ein Wörterbuch zur Verfügung gestellt bekommen. Dazu soll die Wörterbuch-Komponente, die bereits in CT2 implementiert ist, verwendet werden. Durch das Wörterbuch und die Verwendung von Sprachstatistiken soll somit eine dynamische Version des T9-Verfahrens ermöglicht werden. Dynamisch bedeutet in diesem Zusammenhang, dass die Komponente während der Ausführung in der Lage sein soll, das Wörterbuch auszutauschen und somit den Text in einer anderen Sprache dekodieren, kann. Da es beim T9-Code, wie in Kapitel 3.1.1 beschrieben, dazu kommen kann, dass es mehrere mögliche Wörter gibt, soll mittels N-Grammen das Wort ausgesucht werden, dass am wahrscheinlichsten für die ausgewählte Sprache ist.

4.2.2. Aufbau der Komponente

Der Aufbau der T9-Code-Komponente wird im Folgenden beschrieben. Die Einstellungen sowie die Ein- und Ausgänge der T9-Code-Komponente sind in Abbildung 25 dargestellt.

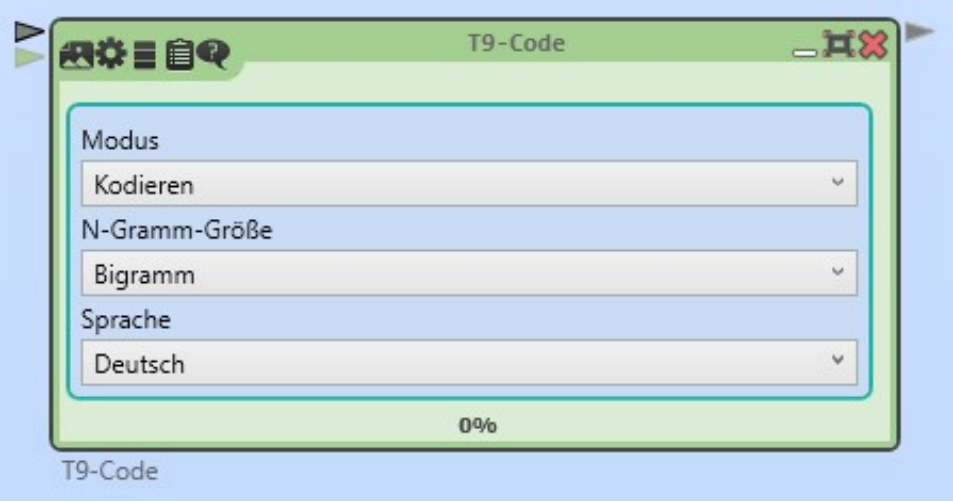

Abbildung 25: Einstellungen der T9-Code-Komponente

Eingänge Neben dem Eingang für den zu verarbeitenden Text besitzt die T9-Code-Komponente einen Eingang für ein Wörterbuch. Dieser Eingang kann mit der Wörterbuch-Komponente von CT2 verbunden werden. Über ihn erhält die Komponente das Wörterbuch, das verwendet wird, um die Wörter zu dekodieren. Daher kann die Komponente ohne diesen Eingang nicht ausgeführt werden, wenn sie sich im Dekodieren-Modus befindet.

Ausgänge Die Komponente besitzt nur einen Ausgang. Dieser Ausgang stellt das Ergebnis für andere Komponenten zur Verfügung.

Einstellungen Die Einstellungen der T9-Code-Komponente sind in Abbildung 25 veranschaulicht. Über die Einstellung *Modus* kann ausgewählt werden, ob die Komponente zum Dekodieren oder zum Kodieren verwendet wird. Die Einstellung *N-Gramm-Größe* gibt an, welcher N-Gramm-Typ verwendet werden soll, um eine Mehrdeutigkeit beim Dekodieren aufzulösen. Die Einstellung *Sprache* gibt an, für welche Sprache die N-Gramm-Statistik geladen werden soll. Die Sprache sollte gleich der des Wörterbuches sein, um das bestmögliche Ergebnis zu erzielen.

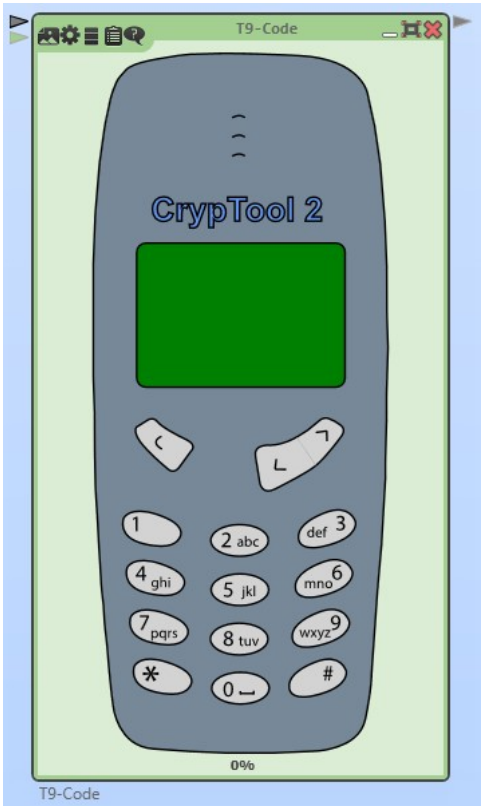

Abbildung 26: Visualisierung der T9-Code-Komponente

Visualisierung Die Komponente besitzt eine Visualisierung in Form eines Mobiltelefons, welche in Abbildung 26 dargestellt ist. Durch die Zahlen 2-9 kann die entsprechende Zahl im Display ergänzt werden. Außerdem kann die Taste über der 1 zum Löschen eines einzelnen Zeichens verwendet werden. Alle weiteren Tasten dienen nur der Visualisierung und haben keine Funktion. Die Visualisierung wurde so entwickelt, dass sie sich dynamisch an die Größe der Komponente anpasst.

4.2.3. Implementierung der Komponente

Zunächst wurde die Logik des T9-Codes in einer C#-Konsolenanwendung implementiert, um eine schnellere Entwicklung zu ermöglichen. Die Entwicklung dieser Anwendung wurde durch Test-Driven-Design (TDD) unterstützt, indem zunächst Tests erstellt wurden, die die Kodierung mittels T9 testen. Anschließend wurde die Kodierung implementiert, bis die Tests erfolgreich ausgeführt werden konnten. Die Dekodierung wurde zunächst so implementiert, dass alle Zeichenketten erzeugt wurden, die durch die gegebenen Zahlen möglich sind. Auch die Dekodierung wurde mit Hilfe des TDD-Konzeptes entwickelt.

Anschließend wurde der Code der Konsolenanwendung in das CT2-Plugin-Template übertragen und die notwendigen Eingänge, Ausgänge und Einstellungen ergänzt. Für die Visualisierung wurde zunächst ein Mobiltelefon als SVG-Grafik erstellt. Anschließend wurde die SVG-Grafik mittels des Plugins „Ai->XAML Export Plug-In Version 0.3“[6] zu XAML konvertiert. Der XAML-Code wurde anschließend etwas vereinfacht und in die WPF-Komponente des Plugins eingefügt. Damit die Grafik interaktiv bedient werden kann, haben die entsprechenden Tasten einen Namen zugewiesen bekommen. Beim Laden der Komponente wird im Konstruktor für jeden klickbaren Pfad in der Visualisierung ein Event-Listener hinzugefügt. Da die Beschriftung der Tastatur über den Pfaden der Tasten liegt, konnte die darunterliegende Taste nicht geklickt werden, wenn sich der Cursor über einer Beschriftung befindet. Um die Beschriftung der Tasten beim Klicken zu ignorieren, wurde jedem Pfad die Eigenschaft *IsHitTestVisible="False"* hinzugefügt. Anschließend wurden alle Flächen mit einer Füllung versehen, woraus dann die Visualisierung in Abbildung 26 entstand. Um die eingegebenen Zahlen darstellen zu können, wurde über das Display des Mobiltelefons eine Textbox gelegt, die einen transparenten Hintergrund und keinen Rahmen besitzt. Somit ist sie für den Betrachter unsichtbar. Um zu verhindern, dass ein Nutzer mit der Textbox interagieren kann, wurde das Property *IsEnabled* auf *False* gesetzt. Somit kann der Nutzer die Zeichen im Textfeld nicht mehr verändern.

Wird die T9-Code-Komponente ausgeführt, wird zunächst in der *Execute*-Methode geprüft, ob die Komponente im Modus zur Kodierung oder Dekodierung gestartet wurde. Wenn die Komponente im Modus zum Kodieren gestartet wurde, wird die Methode *Encode* der Klasse *CryptoService* aufgerufen. Diese Methode ersetzt jedes Zeichen des gegebenen Klartextes durch die entsprechende Zahl. Dazu wird ein Dictionary verwendet, das in der Klasse *CharMapping* fest definiert ist.

Wird die Komponente jedoch im Modus zur Dekodierung gestartet, wird zunächst das N-Gramm für die entsprechende Sprache geladen, das über die Einstellungen der Komponente definiert wurde. Im nächsten Schritt wird der Wörterbuch-Eingang überprüft. Wenn dort keine Daten anliegen, wird die Ausführung abgebrochen, da die Dekodierung ohne Wörterbuch nicht möglich ist. Anschließend wird überprüft, ob die interne Repräsentation des Wörterbuches neu erstellt werden muss. Die interne Repräsentation wird später genauer erläutert. Nachdem das Wörterbuch eingelesen wurde, wird das Display des Mobiltelefons in der Visualisierung aktualisiert, indem die Zahlen, die dekodiert werden sollen, angezeigt werden. Im letzten Schritt wird die Methode *Decode* der Klasse *CryptoService* aufgerufen. Diese unterteilt den zu dekodierenden Text zunächst in die einzelnen Wörter, indem die Eingabe überall dort geteilt wird, wo eine „0“ verwendet wird. Anschließend wird jede so erhaltene Zeichenkette mittels des Wörterbuchs dekodiert. Wenn ein Wort nicht dekodiert werden kann, wird ein entsprechender Hinweis dazu in das Ergebnis hinzugefügt. Sollten für eine Zeichenkette mehrere Wörter möglich

[6] By Mike Swanson (http://blog.mikeswanson.com/)

sein, werden die Wörter mittels N-Grammen und einer Bewertungsfunktion bewertet und das Wort mit dem höchsten Wert wird als Ergebnis betrachtet.

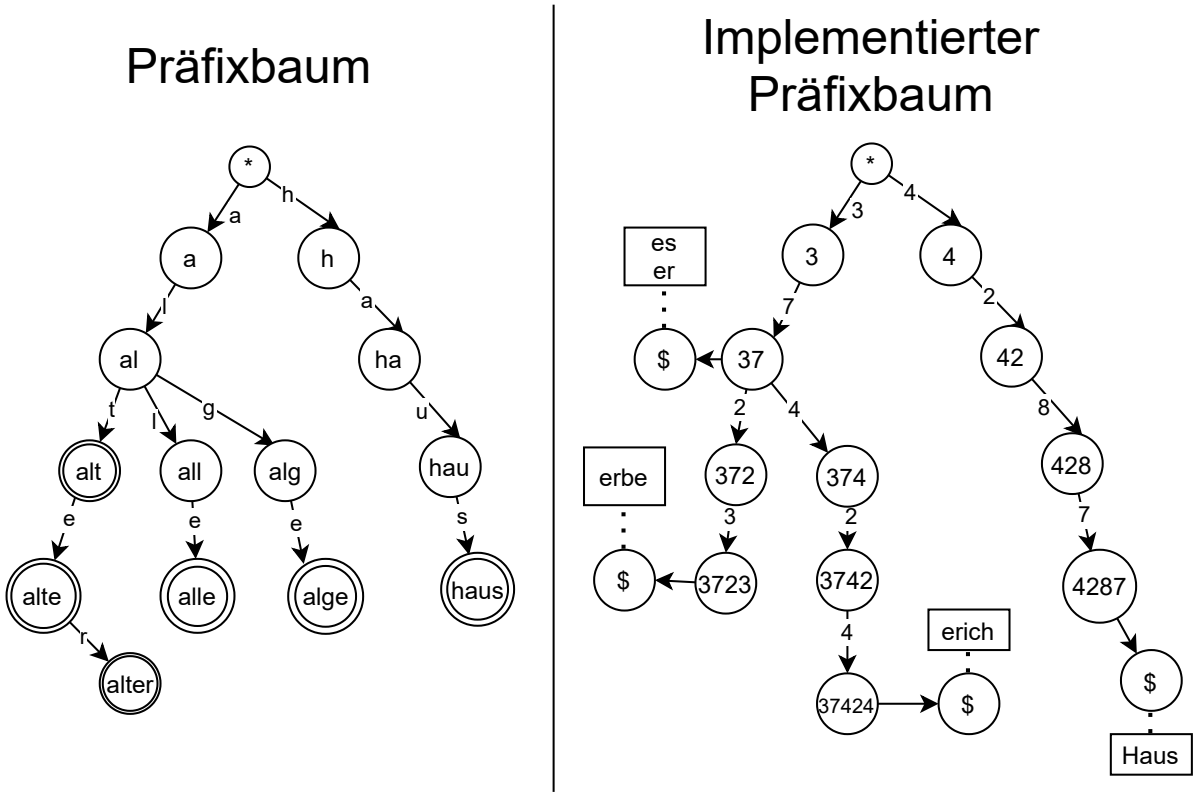

Abbildung 27: Herkömmlicher Präfixbaum und implementierter Präfixbaum

Interne Speicherung des Wörterbuchs CT2 bietet durch die Wörterbuch-Komponente für eine Vielzahl von Sprachen Wörterbücher, die von anderen Komponenten genutzt werden können. Die T9-Code-Komponente verwendet die Wörterbuch-Komponente, um die Dekodierung durchführen zu können. Aufgrund der Größe der Wörterbücher in CT2 kann jedoch nicht einfach das gesamte Wörterbuch durchsucht werden. Daher wird das Wörterbuch nur einmal zu Beginn der Ausführung eingelesen und dabei in einem Präfixbaum[3] gespeichert. Bei einem Präfixbaum können von einem Knoten mehrere Kanten ausgehen. Jede Kante stellt dabei ein Zeichen dar. Die Blattknoten repräsentieren ganze Wörter, wobei die inneren Knoten das Präfix für alle nachfolgenden Wörter bilden, die sich tiefer im Baum befinden.

Beim Laden des Wörterbuches wird jedes Wort zunächst kodiert. Dadurch erhält man die T9-Repräsentation des Wortes, welche nur aus Zahlen besteht. Deshalb wurde der Präfixbaum so angepasst, dass die Kanten nur Zahlen von zwei bis neun sowie ein Terminalsymbol enthalten können. Das Terminalsymbol wird verwendet, um einen Knoten zu markieren, in dem alle Wörter aus dem Wörterbuch gespeichert sind, die durch das

Präfix möglich sind. Durch diese Speicherung des Wörterbuches kann das Dekodieren wesentlich beschleunigt werden, da pro Zahlenkette nur noch so viele Schritte ausgeführt werden müssen, wie die Zahlenkette lang ist.

In Abbildung 27 ist auf der linken Seite der Aufbau eines herkömmlichen Präfixbaums veranschaulicht. Bei der implementierten Version auf der rechten Seite besitzen Knoten, die ein gültiges Wort bilden (zum Beispiel „3723“ bzw. „erbe“), einen zusätzlichen Kinderknoten, der ein Terminalsymbol enthält. Der Knoten mit dem Terminalsymbol enthält eine Liste mit allen Wörtern, die durch diese Zahlenkette möglich sind. In den meisten Fällen enthält diese Liste nur ein Wort. Sollte jedoch der Fall auftreten, dass mehrere Wörter durch die gleiche Zahlenkette gebildet werden können, wird mittels N-Grammen entschieden, welches dieser Wörter ausgewählt werden soll. Da die Bewertung mittels N-Grammen recht aufwendig ist und es nur selten vorkommt, dass Wörter durch die gleiche Zeichenkette gebildet werden können, wird die Bewertung nicht bereits beim Einfügen vorgenommen. Dadurch konnte die initiale Startzeit der Komponente verringert werden.

4.3. Straddling-Checkerboard und Ché Guevara

Dieser Abschnitt erläutert die Anforderungen, die an die *Straddling-Checkerboard*-Komponente zur Design-Zeit gestellt wurden. Daraus resultierend wird der Aufbau der Komponente beschrieben. Im letzten Teil dieses Abschnittes wird die Implementierung anhand des Ablaufs beim Ausführen der Komponente erläutert. Das Template, das zur Erläuterung der Komponente erstellt wurde, ist in Abbildung 28 dargestellt.

4.3.1. Anforderungen

Aus der Beschreibung der Funktionsweise in Kapitel 3.2.1 geht hervor, dass das Verfahren sowohl den zu verarbeitenden Text als auch die Zeilen und Spalten und den Inhalt des Straddling-Checkerboards benötigt. Somit ergibt sich, dass für jede dieser Eingaben ein entsprechender Eingang für die Komponente notwendig ist. Um die Anzahl der Eingänge möglichst gering zu halten, soll die Definition für die Zeilen und Spalten in einem Eingang zusammengefasst werden.

Wie bereits erklärt, basiert das Verfahren von Ché Guevara auf einem Straddling-Checkerboard. Deshalb sollte es möglich sein, mit der Straddling-Checkerboard-Komponente auch das Verfahren von Ché Guevara zu unterstützen, indem die Implementierung des Verfahrens so dynamisch gestaltet wird, dass die Zeilen und Spalten die Größe des Straddling-Checkerboards bestimmen.

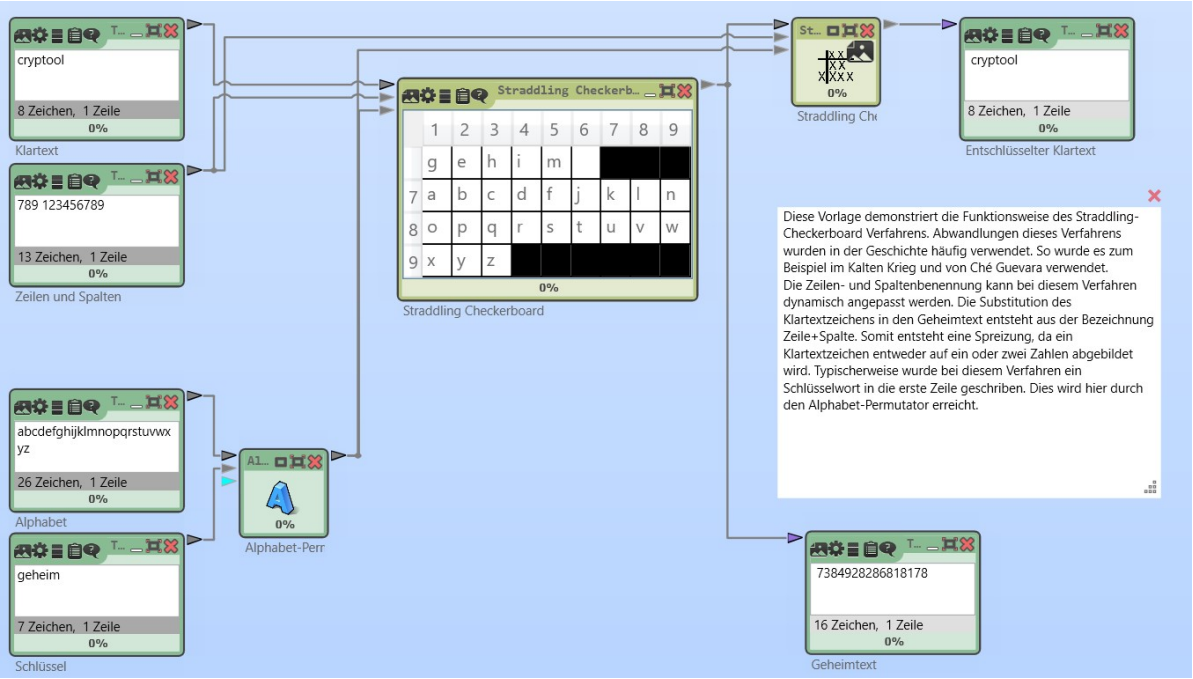

Abbildung 28: Template der Straddling-Checkerboard-Komponente

Die Komponente sollte eine Visualisierung besitzen, in der der Aufbau des Straddling-Checkerboards dargestellt wird. Die Visualisierung sollte sich aktualisieren, sobald neue Daten an den Eingängen der Komponente zur Verfügung stehen. Durch dieses dynamische Verhalten soll der Benutzer dazu angeregt werden das Verfahren zu ergründen. Die Felder, die keine Zeichen enthalten, sollen als solche erkennbar sein.

4.3.2. Aufbau der Komponente

Im folgenden wird der Aufbau der Straddling-Checkerboard-Komponente beschrieben. Dazu werden die Ein- und Ausgänge sowie die Einstellungen der Komponente beschrieben. Die Einstellungen der Komponente sind in Abbildung 29 dargestellt.

Eingänge Die Komponente besitzt drei Eingänge. Über den ersten Eingang kann der zu verarbeitende Text übergeben werden. Dieser Eingang ist erforderlich, damit die Komponente durch CT2 ausgeführt wird, ansonsten stoppt die Ausführung. Der zweite Eingang stellt der Komponente die Zeilen bzw. Spalten zur Verfügung. Diese beiden Zahlengruppen müssen durch ein Trennzeichen voneinander abgegrenzt werden. Die Definition der Zeilen und Spalten ist jedoch auch über die Einstellungen möglich, daher ist dieser Eingang nicht für die Ausführung der Komponente erforderlich. Der dritte Eingang bestimmt den Inhalt des Straddling-Checkerboards.

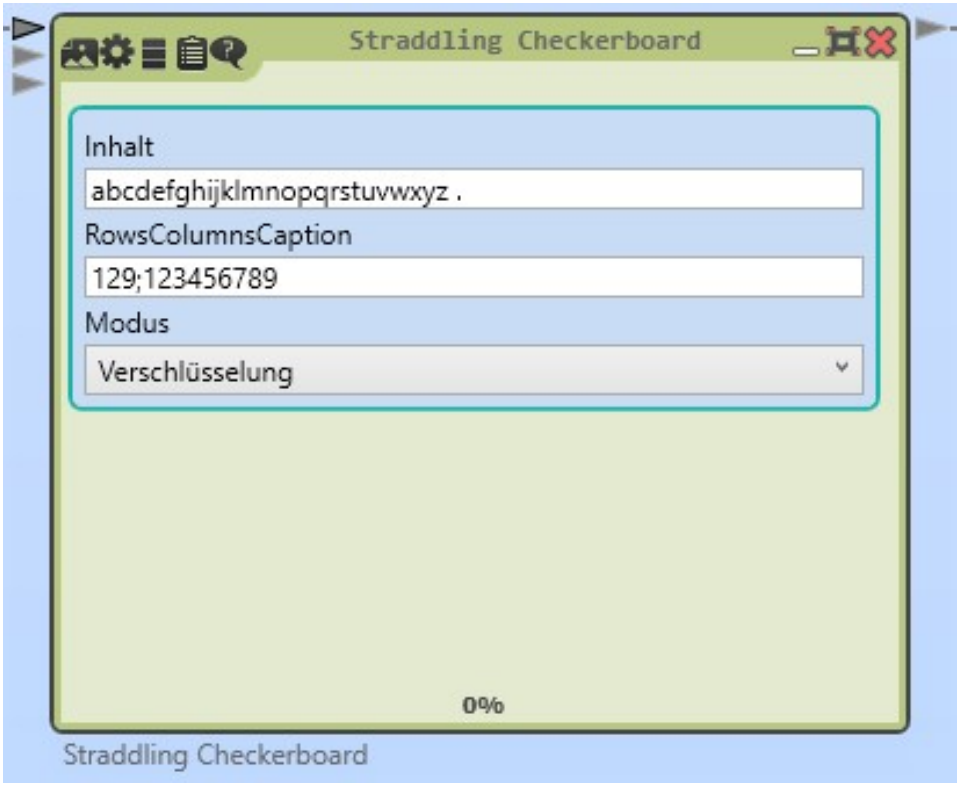

Abbildung 29: Einstellungen der Straddling-Checkerboard-Komponente

Ausgänge Die Komponente besitzt nur einen Ausgang. An diesem Ausgang wird der verarbeitete Text ausgegeben.

Einstellungen Entsprechend der zwei optionalen Eingänge gibt es auch zwei Einstellungen, die beide Eingänge repräsentieren. Sie werden verwendet, wenn keine Komponente an den Eingängen verbunden wurde. Außerdem gibt es die Einstellung „Modus“. Sie legt fest, ob die Komponente zum Ver- oder Entschlüsseln verwendet wird. Die Einstellungen der Straddling-Checkerboard-Komponente sind in Abbildung 29 abgebildet.

Visualisierung Die Visualisierung der Komponente stellt das Straddling-Checkerboard dar, das durch die gegebenen Werte erstellt wurde. Die Visualisierung passt sich während der Ausführung dynamisch an die neuen Werte an, wenn die Daten an den Eingängen geändert werden. Die Felder, die kein Zeichen enthalten, werden schwarz dargestellt. Somit ist erkennbar, welches Feld ein nicht druckbares Zeichen, wie etwa ein Leerzeichen, enthält.

4.3.3. Implementierung der Komponente

Zunächst wurde die Logik des Verfahrens in einer C#-Konsolenanwendung implementiert, um beim Starten der Anwendung bzw. beim Debugging den zusätzlichen Overhead durch das Laden von CT2 zu umgehen. Diese Konsolenanwendung wurde durch TDD entwickelt. Dies bedeutet, dass zunächst Testfälle erstellt wurden, die die Implementierung erfüllen muss. Als Grundlage dazu wurden bereits bekannte Klar- bzw. Geheimtexte sowie die dazugehörige Konfiguration des Straddling-Checkerboards verwendet. [10] [6] Die Logik für die Ver- und Entschlüsselung wurde dann so angepasst, dass alle Tests erfolgreich verliefen. Da einer der Testfälle die Verschlüsselung mit Ché Guevaras Verfahren beinhaltete, wurde die Kompatibilität der Komponente mit diesem Verfahren bereits zur Entwicklungszeit sichergestellt.

Nachdem das Verfahren alle Testfälle erfüllte, konnte die Logik zum Ver- und Entschlüsseln in ein CT2-Plugin kopiert werden. Zusätzlich wurden die Einstellungen, die Ein- und Ausgänge sowie die Logik zum Aufrufen der kopierten Methoden ergänzt. Anschließend wurde die Visualisierung ergänzt. Durch die zusätzliche Logik wurde die Komponentenklasse unübersichtlich, deshalb wurde die Erstellung der internen Repräsentation in eine extra Klasse mit dem Namen *Checkerboard* ausgelagert. Die Klasse *Checkerboard* bietet jene Funktionen, die während der Anwendung des Verfahrens direkt mit der Tabelle, also dem Checkerboard, zusammenhängen.

Das Straddling-Checkerboard verwendet nur die *Execute*-Methode des *ICrypComponent*-Interfaces. Im ersten Schritt wird in der Execute-Methode die aktuelle Konfiguration des Straddling-Checkerboards validiert. Dabei wird zum Beispiel geprüft, dass Zeilen und Spalten keine doppelten Einträge enthalten und ausreichend Platz für den übergebenen Inhalt zur Verfügung steht. Wird eine ungültige Konfiguration erkannt, wird die Ausführung abgebrochen und eine Fehlermeldung ausgegeben.

Sollte die Validierung keine Fehler finden, wird im nächsten Schritt die interne Repräsentation des Straddling-Checkerboards aufgebaut. Dazu werden zwei Dictionaries mit den Substitutionen des Verfahrens erzeugt. Die Logik zum Erstellen und Pflegen dieser zwei Dictionaries wird durch die Klasse *Checkerboard* realisiert. Parallel zu diesem Prozess werden die Daten für die Visualisierung aufbereitet. Sobald dieser Prozess abgeschlossen ist, wird die Visualisierung durch diese Daten aktualisiert.

Im nächsten Schritt wird entweder die Ver- oder Entschlüsselungsmethode aufgerufen. Welche Methode aufgerufen werden muss, wird durch die Einstellung *Modus* der Komponente bestimmt. Beide Methoden wenden analog zueinander die jeweilige Substitution an, wie sie in Kapitel 3.2.1 beschrieben wurde. Unbekannte Zeichen werden beim Verschlüsseln unverändert an den Ausgang weitergeben, während beim Entschlüsseln unbekannte Zeichen ignoriert werden. In beiden Fällen wird eine entsprechende Warnung dazu ausgegeben.

Im letzten Schritt werden die durch die Ver- bzw. Entschlüsselung erhaltenen Daten an den Ausgang weitergegeben.

4.4. Bacon-Chiffre

Im nachfolgenden Abschnitt werden die Anforderungen an die Bacon-Komponente aufgelistet, der Aufbau der Komponente beschrieben und der Prozess der Implementierung beschrieben. Das Template, das die Funktionsweise der Komponente erläutert ist in Abbildgun 30 abgebildet.

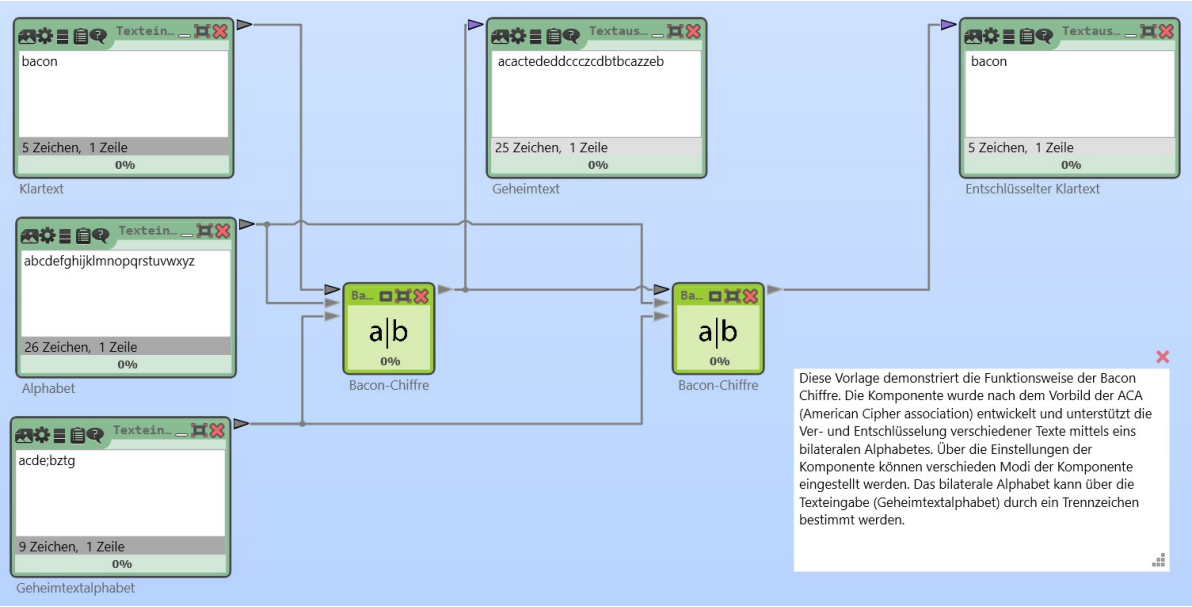

Abbildung 30: Template der Bacon-Komponente

4.4.1. Anforderungen

Das Bacon-Verfahren kann in verschiedenen Arten eingesetzt werden, wie sie in Kapitel 3.4.1 beschrieben wurden. Die Komponente in CT2 soll die Funktionsweise umsetzen, wie sie durch die ACA beschrieben wird. [1] Konkret sollen 3 Modi unterstützt werden:

1. Verwendung von zufälligen Zeichen, die durch einen externen Eingang zur Verfügung gestellt werden.

2. Binär, also a's werden durch eine 0 und b's werden durch eine 1 ersetzt.

3. Zufällige Zeichen aus einem Alphabet, wobei A-M einem a entspricht und N-Z einem b.

Außerdem soll die Komponente die Möglichkeit bieten, die Codelänge anzupassen. Bei der Bacon-Chiffre ist die Codelänge eigentlich auf fünf Zeichen limitiert, wodurch maximal $2^5 = 32$ Zeichen im Klartextalphabet sein können.

4.4.2. Aufbau der Komponente

Dieses Unterkapitel beschreibt den Aufbau der Bacon-Komponente. Dabei werden alle Eingänge, Ausgänge und Einstellungen beschrieben. Die genaue Funktionsweise wird im Kapitel 4.4.3 erläutert.

Eingänge Die Komponente besitzt drei Eingänge. Der erste Eingang ist für die Ausführung der Komponente erforderlich und ist für die Übergabe des zu verarbeitenden Textes notwendig. Über den zweiten Eingang hat der Benutzer die Möglichkeit, das Klartextalphabet bereitzustellen. Dieser Eingang ist äquivalent zu der Einstellung innerhalb der Komponente für das Klartextalphabet. Der dritte Eingang kann dazu verwendet werden, um die Zeichen für die Repräsentation der a's und b's festzulegen.

Ausgänge Die Komponente besitzt nur einen Ausgang. An diesem wird der verarbeitete Text ausgegeben.

Einstellungen Die Einstellungen der Komponente sind in Abbildung 31 dargestellt. Die erste dieser Einstellungen der Bacon-Komponente kann dazu verwendet werden, das Klartextalphabet festzulegen. Sie wird automatisch auf den Wert des entsprechenden externen Eingangs gesetzt, wenn dort Daten zur Verfügung stehen. Über die Einstellung der Komponente kann zudem der Modus (Ver- und Entschlüsselung), in dem die Komponente betrieben werden soll, festgelegt werden. Die dritte Einstellung ist mit dem Namen *Ausgabemodus* benannt und kann dazu verwendet werden, das Verhalten beim Ver- und Entschlüsseln zu beeinflussen. Die drei Modi spiegeln dabei die Funktionen wider, die in den Anforderungen der Komponente definiert wurden.

Außerdem bieten die Einstellungen eine Gruppe mit dem Namen *Länge*. Über die Einstellungen in dieser Gruppe kann die Länge der Codes festgelegt werden. Dies kann entweder statisch über die Zahleneingabe *Codewortlänge* oder über die Option *Dynamische Codewordlänge* erfolgen. Wenn die Option *Dynamische Codewordlänge* aktiviert ist, berechnet die Komponente automatisch die Codelänge, die notwendig ist, um das

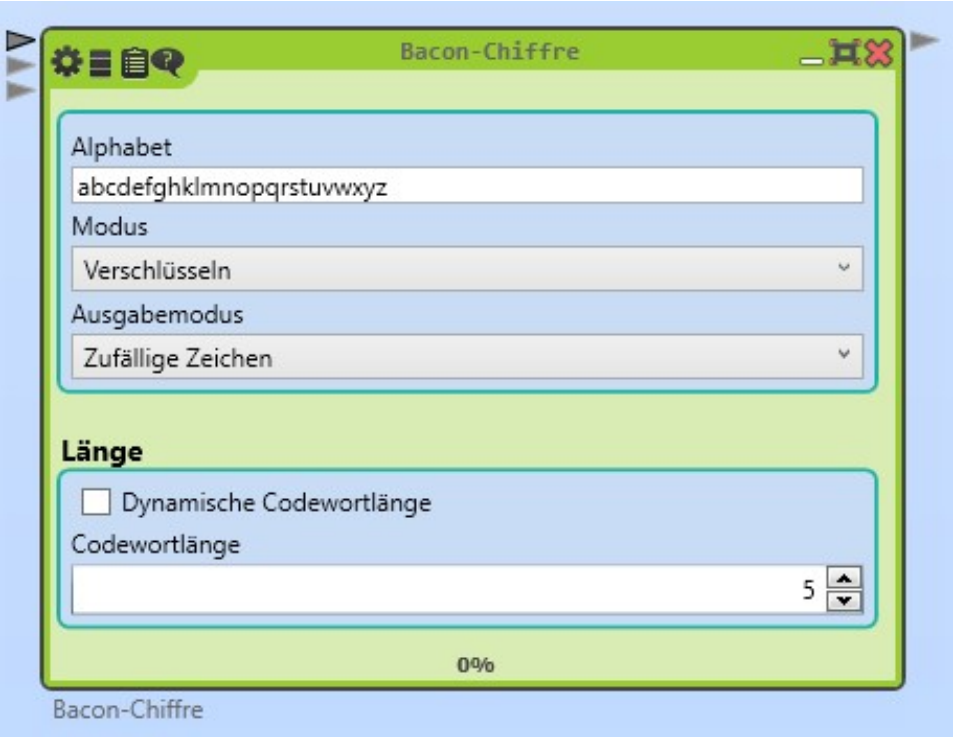

Abbildung 31: Einstellungen der Bacon-Komponente

gesamte Klartextalphabet darstellen zu können. Ist diese Option nicht aktiviert, muss die Codelänge über die Einstellung *Codewortlänge* manuell festgelegt werden, als Standardwert ist 5 festgelegt.

Visualisierung Für die Bacon-Komponente kann keine sinnvolle Visualisierung erstellt werden, daher wurde bei der Implementierung darauf verzichtet.

4.4.3. Implementierung der Komponente

Die *Execute*-Methode prüft im ersten Schritt, welcher Ausgabemodus in den Einstellungen der Komponente ausgewählt wurde. Wenn der Modus *Externes Geheimtextalphabet* ausgewählt wurde, wird versucht, die Daten am Eingang *Geheimtextalphabet* in zwei Gruppen zu unterteilen. Dazu wird zuerst versucht, anhand eines Trennzeichens, die Zeichenkette in zwei Teile zu unterteilen. Gelingt dies nicht, wird die Anzahl der Zeichen durch zwei geteilt und dieser Wert aufgerundet. Alle Zeichen, die vor diesem Wert liegen, gehören dann der ersten Gruppe an, alle Zeichen, nach diesem Wert, gehören der zweiten Gruppe an.

Im nächsten Schritt wird überprüft, ob die in den Einstellungen gewählte Codewortlänge ausreicht, um das bereitgestellte Alphabet damit codieren zu können. Wenn dies nicht der Fall ist, wird eine Fehlermeldung ausgegeben und die Ausführung wird unterbrochen.

Anschließend kann begonnen werden das Alphabet zu kodieren. Dazu wird jedes Zeichen des bereitgestellten Alphabets binär durchnummeriert. Die Zuordnung des Zeichens zu der Kodierung wird in einem Dictionary gespeichert, ebenso wie die Umkehrung, also die Kodierung zu dem entsprechenden Zeichen. Außerdem können Zeichen durch Klammern gruppiert werden. Wenn also als Alphabet „a(bc)d“ bereitgestellt wird, dann erhalten die Zeichen „b“ und „c“ die gleiche Kodierung.

Im letzten Schritt wird überprüft, ob die Ver- oder die Entschlüsselung aufgerufen werden muss, dazu wird der Wert der Einstellung *Modus* überprüft. Beide Methoden durchlaufen jedes einzelne Zeichen, das durch den Eingang *Klartext* bereitgestellt wird. Beim Verschlüsseln wird zuerst das Zeichen mittels des Wörterbucheintrags kodiert. Anschließend wird die Kodierung in das Format konvertiert, das durch den Ausgabemodus festgelegt wurde. Beim Entschlüsseln werden zunächst solange Zeichen hintereinander gehangen, bis die in den Einstellungen festgelegte Länge der Codewörter erreicht wurde. Anschließend wird diese Zeichengruppe entsprechend der Einstellung *Ausgabemodus* zurück in die binäre Darstellung überführt. Im letzten Schritt wird die binäre Darstellung zurück in das Klartextzeichen überführt. Wenn ein Zeichen nicht entschlüsselt werden kann, wird der Vorgang an dieser Stelle unterbrochen und nur der bis zu diesem Zeitpunkt erfolgreich entschlüsselte Text zurückgegeben.

4.5. Josse-Cipher

In diesem Abschnitt werden die Anforderungen an die Komponente zusammengefasst und die Schritte bei der Implementierung der Komponente für das Josse-Cipher-Verfahren beschrieben. Das Template, das zur Erläuterung der Funktionsweise der Komponente erstellt wurde, ist in Abbildung 32 dargestellt.

4.5.1. Anforderungen

Wie in Kapitel 3.5.1 beschrieben, benötigt Josse für das Verfahren ein Schlüsselwort und ein Alphabet, das für den Klar- und Geheimtext genutzt wird. Diese Daten müssen neben dem zu verschlüsselnden Text für die Komponente zur Verfügung stehen, damit das Verfahren erfolgreich ausgeführt werden kann.

Josse reduziert das verwendete Alphabet auf 25 Zeichen, um dadurch im weiteren Verlauf der Ver- bzw. Entschlüsselung die Berechnung zu erleichtern. Da dies nur beim händi-

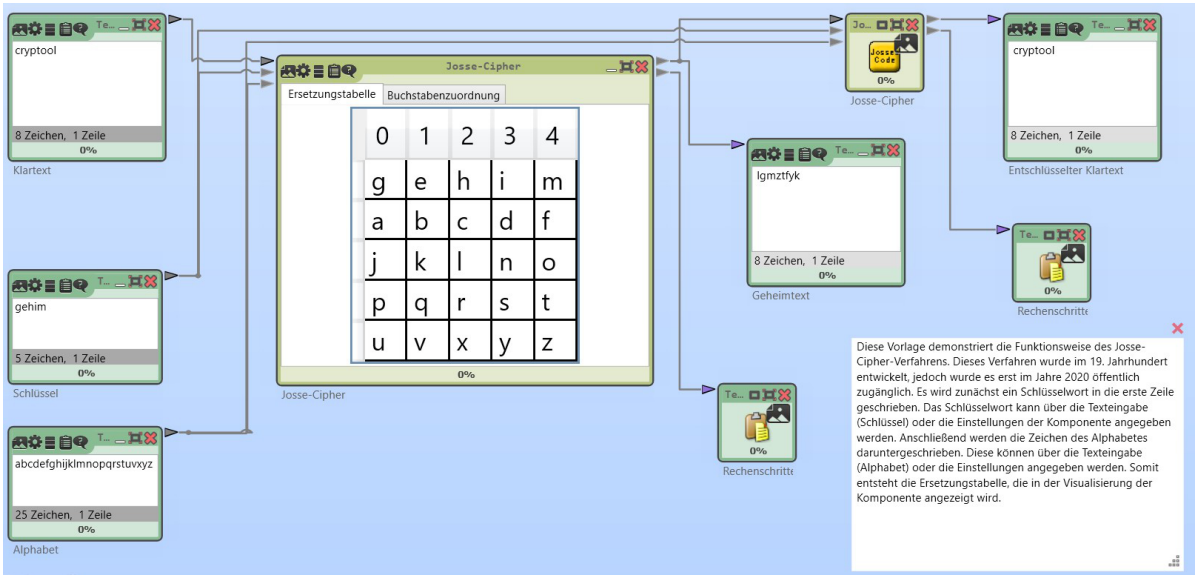

Abbildung 32: Template der Josse-Cipher-Komponente

schen Rechnen einen Vorteil bietet, soll die Komponente so entwickelt werden, dass eine beliebige Länge das Alphabets verwendet werden kann. Es soll also möglich sein, dass ein Benutzer jedes beliebige Alphabet (inklusive des originalen Alphabets von Josse) verwenden kann.

Um die Vorgehensweise bei diesem Verfahren einfacher verdeutlichen zu können, sollen die intern verwendete Ersetzungstabelle sowie die Zuordnung der Zeichen zu ihrer numerischen Repräsentation in einer Visualisierung dargestellt werden. Dabei soll diese Visualisierung so dynamisch sein, dass sie auf Änderungen des Schlüsselwortes oder des Klartextalphabets während der Ausführung der Komponente reagiert.

Für die Ver- bzw. Entschlüsselung verwendet Josse einige Sonderbehandlungen. Damit die einzelnen Rechenschritte für den Nutzer der Komponente verständlich werden, sollen die internen Abläufe einfach zugänglich gemacht werden.

4.5.2. Aufbau der Komponente

In diese Kapitel wird der Aufbau der Josse-Cipher-Komponente beschrieben. Dazu werden zunächst die Ein- und Ausgänge und anschließend die Einstellungen der Komponente beschrieben.

Eingänge Die Josse-Cipher-Komponente besitzt drei Eingänge für Daten. Der einzige Eingang, der für die Komponente erforderlich ist, ist die *Texteingabe*. Der Texteingabe-

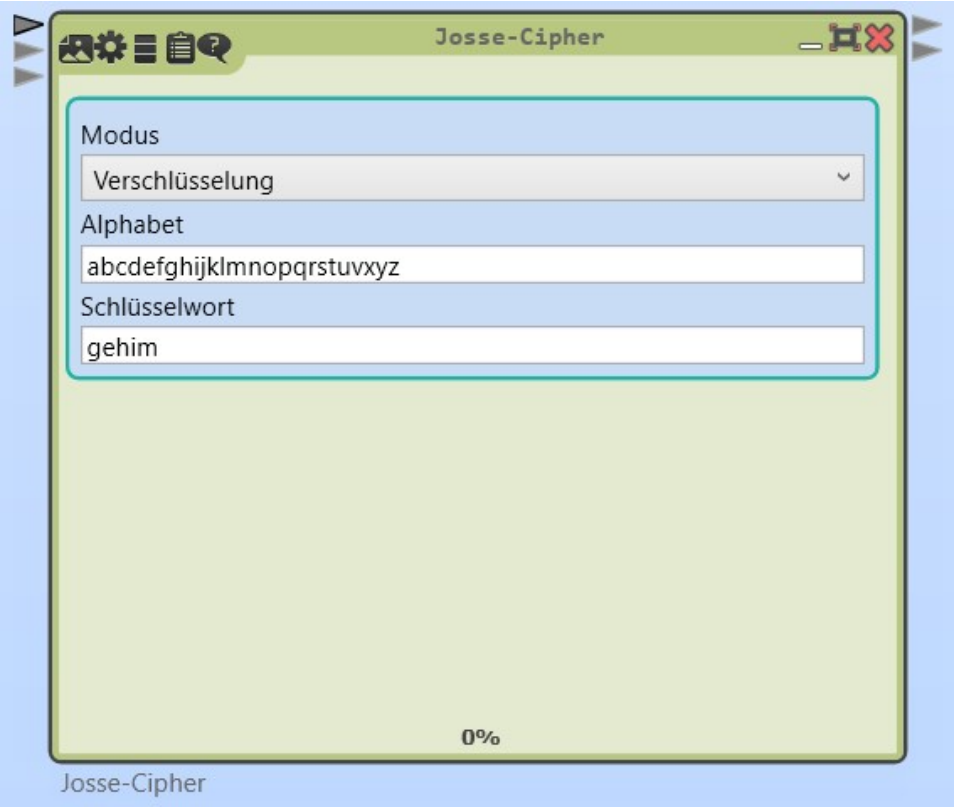

Abbildung 33: Einstellungen der Josse-Cipher-Komponente

Eingang wird dazu verwendet, den zu verschlüsselnden bzw. zu entschlüsselnden Text an die Komponente zu übergeben. Die anderen zwei Eingänge dieser Komponenten sind optional. Der erste optionale Eingang wird mit *Schlüsselwort* bezeichnet und stellt das Schlüsselwort für das Verfahren bereit. Das Schlüsselwort kann außerdem über die Einstellungen der Komponente bereitgestellt werden, daher ist dieser Eingang als optional markiert. Der zweite optinale Eingang ist das *Externe Alphabet*. Dieser Eingang gibt an, welche Zeichen verarbeitet werden können. Alle Zeichen des Schlüsselwortes müssen auch im Alphabet enthalten sein. Die restlichen Zeichen werden verwendet, um die Ersetzungstabelle zu generieren.

Ausgänge Die Komponente besitzt 2 Ausgänge. Der erste Ausgang gibt den verarbeiteten Text des Texteingabe-Eingangs aus. Der zweite Ausgang gibt die verwendeten Rechenschritte aus, um die Rechenschritte, die für das Ergebnis am ersten Ausgang verantwortlich sind, nachvollziehbarer zu machen.

Einstellungen Die Einstellungen der Komponente werden in Abbildung 33 dargestellt. Damit die Komponente sowohl zum Ver- als auch zum Entschlüsseln verwendet werden

kann, kann der Modus, in dem die Komponente betrieben wird, in den Einstellungen geändert werden. Zudem gibt es die Einstellung *Alphabet*, die das Klartext- und Geheimtextalphabet festlegt. Wenn jedoch am Eingang „Externes Alphabet“ Daten zur Verfügung stehen, dann werden die Daten des Eingangs priorisiert behandelt und die Einstellung übernimmt den Wert des Eingangs. Eine weitere Einstellung ist das *Schlüsselwort*. Diese Einstellung kann ebenfalls durch den entsprechenden Eingang überschrieben werden. Standardmäßig ist das Schlüsselwort leer, dann wird die Ersetzungstabelle ohne Schlüsselwort, aber mit 5 Spalten gebildet.

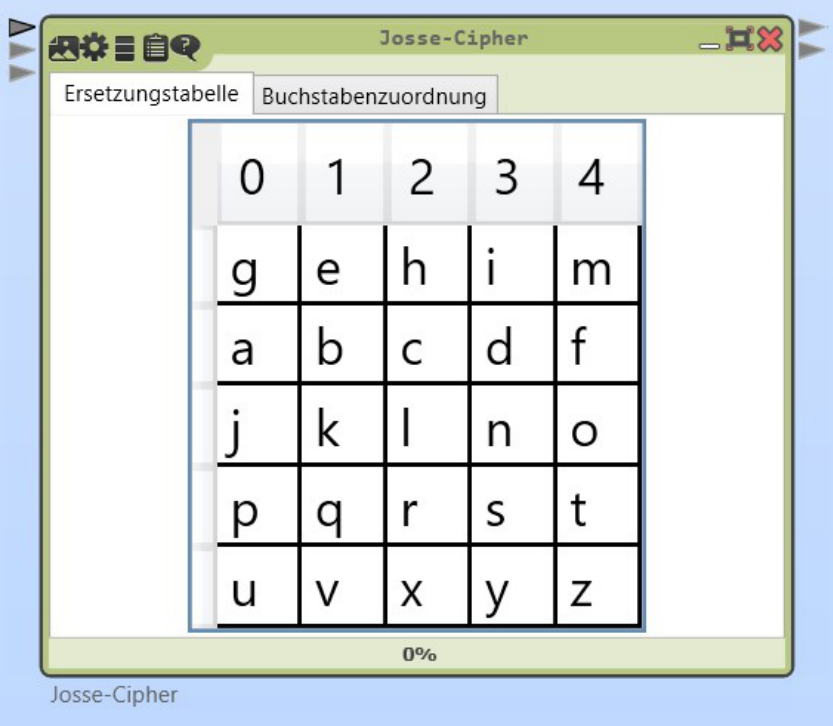

Abbildung 34: Visualisierung der Josse-Cipher-Komponente

Visualisierung In Abbildung 34 ist die Visualisierung der Komponente dargestellt. Die Visualisierung der Komponente ist in zwei Registerkarten unterteilt. In der ersten Registerkarte wird die in Kapitel 3.5.1 beschriebene und in Tabelle 3 dargestellte Ersetzungstabelle angezeigt (Man beachte, dass das Alphabet kein „w“ enthält). Durch diese Darstellung kann dem Benutzer das Resultat seiner Eingabe leichter zugänglich gemacht werden. Die Tabelle enthält in der ersten Zeile das Schlüsselwort, das entweder durch die Einstellung oder den Eingang der Komponente bereitgestellt wird. Anschließend wird die Tabelle mit den Zeichen aus dem Alphabet aufgefüllt, die nicht im Schlüsselwort enthalten sind. Die zweite Registerkarte stellt die Abbildung der Zeichen auf eine Zahl dar. Die Darstellung ist damit äquivalent zu Tabelle 4. Sie hilft dabei dem Nutzer, die generierten Substitutionen einfacher zu verstehen.

4.5.3. Implementierung der Komponente

Wenn die *Execute*-Methode des Josse-Cipher-Plugins aufgerufen wird, erzeugt die Methode *BuildDictionaries* zunächst eine interne Repräsentation für die Substitution. Parallel zu diesem Vorgang werden die Daten für die Visualisierung generiert und anschließend an die WPF-Komponente übergeben. Die Visualisierung besteht aus einer TabControl mit zwei TabItems, die jeweils ein DataGrid enthalten. TabControls bieten die Möglichkeit, verschiedene Registerkarten zu erstellen, und ein Datagrid bietet eine einfache Tabelle. Für die Visualisierung wurden jedoch alle Interaktionsmöglichkeiten mit dem Datagrid deaktiviert, um ein Verändern der Werte oder der Reihenfolge der Darstellung durch den Benutzer zu verhindern. Es dient nur der Anzeige.

Im nächsten Schritt wird die Methode *Encipher* oder *Decipher* aufgerufen. Welche der beiden Methoden aufgerufen wird ist abhängig von der Einstellung *Modus*. Die Methoden implementieren das Vorgehen, das in Kapitel 3.5.1 beschrieben wurde. Parallel zu jedem Ver- bzw. Entschlüsselungsschritt werden die durchgeführten Rechenschritte in einem Array gespeichert. Am Ende des Ver- bzw. Entschlüsselungsprozesses wird das Array mit den einzelnen Rechenschritten durch die Klasse *ArrayPrinter* in eine ASCII-Tabelle überführt. Die Tabelle ist dabei ähnlich zu der Tabelle 5 aus Kapitel 3.5.2. Das Ergebnis der Verschlüsselung sowie die Zeichen der ASCII-Tabelle werden im letzten Schritt an die entsprechenden Ausgänge übergeben.

4.6. Josse-Cipher-Analyse

In diesem Abschnitt wird die Entwicklung der Analyse-Komponente für das Josse-Cipher-Verfahren erläutert. Zunächst werden die Anforderungen definiert. Anschließend wird der Aufbau der Komponente und zum Schluss die Implementierung anhand des Programmablaufes beschrieben. Das Template, das die Funktionsweise der Josse-Cipher-Analyse-Komponente verdeutlicht, ist in Abbildung 35 dargestellt.

4.6.1. Anforderungen

Die Komponente soll so gestaltet werden, dass sie optisch an das Design der anderen Analyse-Komponenten in CT2 anknüpft. Dies bedeutet, dass die Komponente eine Visualisierung benötigt. In der oberen Hälfte soll die Visualisierung einige Statusinformationen bezüglich der Analyse anzeigen. In der unteren Hälfte soll eine Liste mit den gefundenen Schlüsseln angezeigt werden.

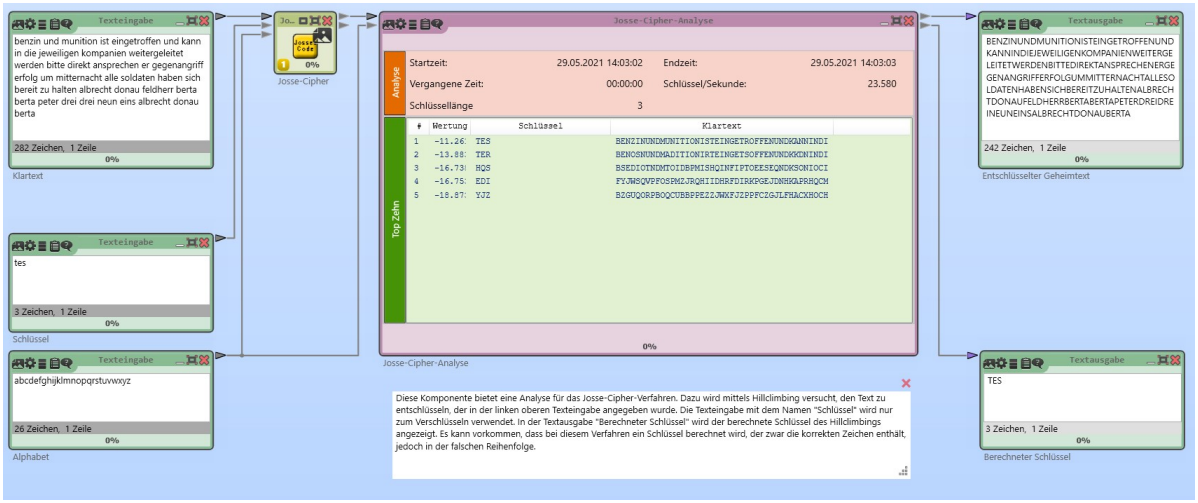

Abbildung 35: Template der Josse-Cipher-Analyse-Komponente

Die Komponente soll so gestaltet sein, dass sie einfach erweitert werden kann. Mögliche Erweiterungen könnten beispielsweise neue Kostenfunktionen oder andere Analyseverfahren sein. Der Quellcode soll entsprechend dynamisch gestaltet sein, damit zukünftige Erweiterungen möglichst nur aus dem Hinzufügen einer weiteren Klasse bestehen.

Zunächst soll nur das Hillclimbing-Verfahren, das in Kapitel 3.5.3 beschrieben wurde mit einigen Kostenfunktionen implementiert werden. Die angewendete Kostenfunktion soll über die Einstellungen der Komponente definiert werden können.

4.6.2. Aufbau der Komponente

Die Ein- und Ausgänge sowie die Einstellungen der Josse-Cipher-Analyse-Komponente werden in diesem Unterkapitel beschrieben. Die Einstellungen der Komponente sind in Abbildung 36 dargestellt.

Eingänge Die Komponente besitzt zwei Eingänge. Über den ersten Eingang erhält die Komponente den Geheimtext, der analysiert werden soll. Wenn dieser Eingang nicht mit einer anderen Komponente verbunden ist, wird die Komponente nicht ausgeführt. Der zweite Eingang ist nicht notwendigerweise für die Ausführung der Komponente erforderlich. Über ihn kann das Alphabet definiert werden, das für die Entschlüsselung verwendet wird, jedoch kann das Alphabet auch über die Einstellungen festgelegt werden.

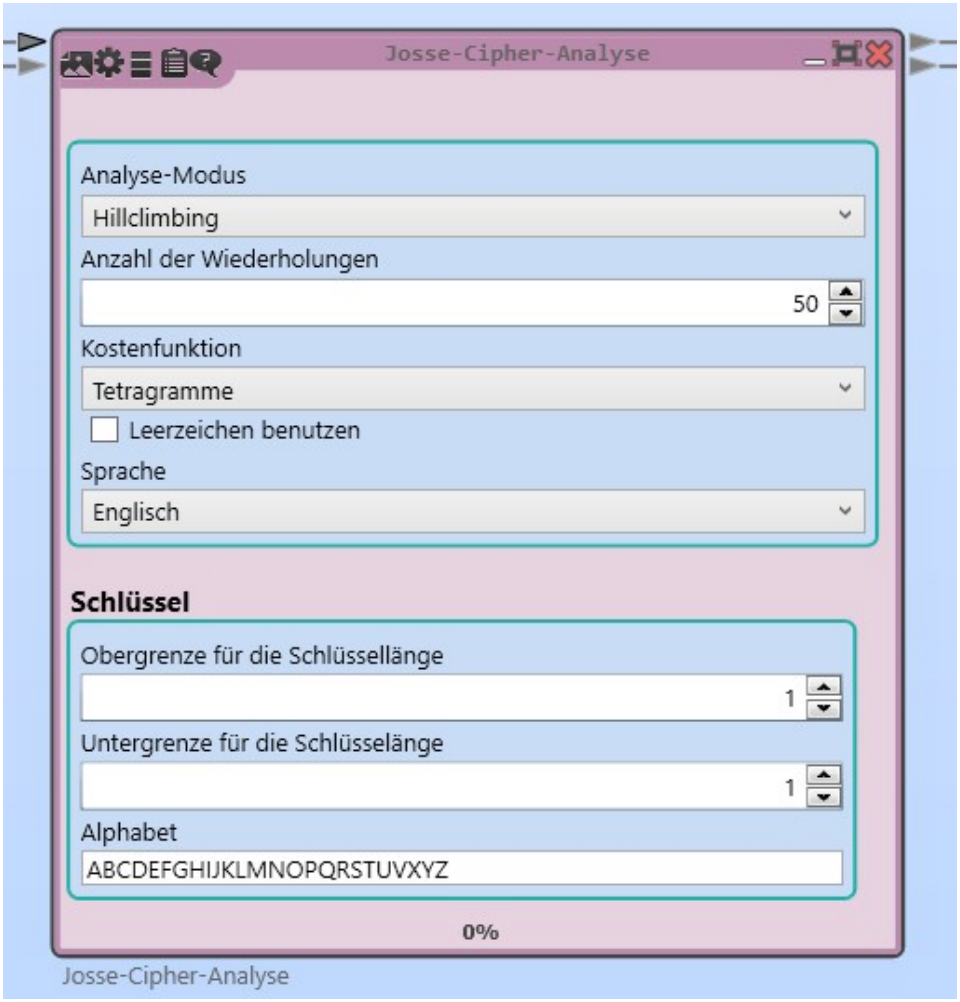

Abbildung 36: Einstellungen der Josse-Cipher-Analyse-Komponente

Ausgänge Die Komponente besitzt zwei Ausgänge. Der obere Ausgang gibt den entschlüsselten Geheimtext aus, während der untere Ausgang den dazugehörigen Schlüssel ausgibt.

Einstellungen Die Einstellungen der Josse-Cipher-Analyse-Komponente sind in Abbildung 36 dargestellt. Über die erste Einstellung kann der Analyse-Modus (hier Hillclimbing) gewählt werden, mit dem der Geheimtext angegriffen werden soll. Das Dropdown bietet die Möglichkeit zwischen einem Hillclimbing-Angriff und einem Angriff auf Basis vom Simulated Annealing zu wechseln. Die Einstellung *Anzahl der Neustarts* bietet die Möglichkeit, die Anzahl der Neustarts, die das Hillclimbing-Verfahren intern macht, festzulegen. Über den Einstellungspunkt *Kostenfunktion* kann bestimmt werden, welches statistische Verfahren angewendet werden soll, um den entschlüsselten Klartext zu bewerten. Die nächsten beiden Einstellungen sind notwendig für die Kostenfunktion.

Zum einen kann eingestellt werden, ob Leerzeichen mit bewertet werden sollen und zum Anderen kann die Sprache bestimmt werden, die für die Bewertung verwendet wird.

Die zweite Gruppe der Einstellungen bezieht sich auf den Schlüssel. Mit den ersten beiden Einstellungen kann die maximale bzw. minimale Länge des Schlüssels festgelegt werden. Die letzte Einstellung in dieser Gruppe bietet die Möglichkeit, das verwendete Alphabet festzulegen. Die Einstellung kann durch den Alphabet-Eingang überschrieben werden.

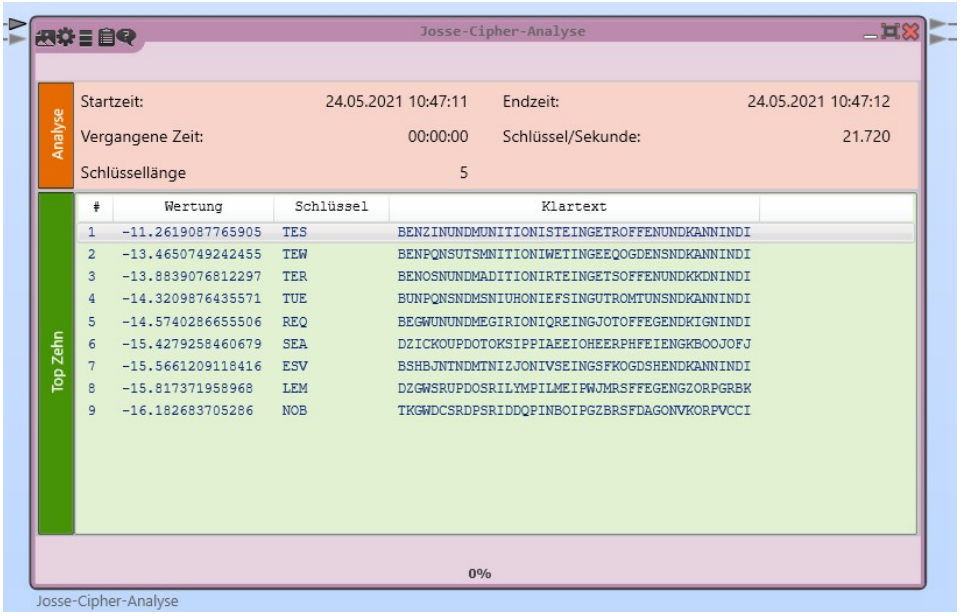

Abbildung 37: Visualisierung der Josse-Cipher-Analyse-Komponente

Visualisierung In CT2 gibt es eine Vorlage, die verwendet wird, um alle Visualisierungen von Analyse-Komponenten möglichst gleich aussehen zu lassen. Dieses WPF-Control heißt *CrypAnalysisViewControl* und bietet das Grundgerüst für die Visualisierung in Abbildung 37. In der Kopfzeile der Komponente werden Metadaten zur Analyse angezeigt. Diese umfassen die Start und Endzeit, die vergangene Zeit während der Ausführung, die Anzahl der getesteten Schlüssel und die Schlüssellänge, die aktuell analysiert wird.

In der Tabelle unter der Kopfzeile werden die Schlüssel angezeigt, bei denen eine Verbesserung der Bewertung stattgefunden hat. Für jeden Schlüssel wird der Wert angezeigt, der sich aus der Kostenfunktion ergeben hat. Außerdem wird der Schlüssel selbst und der daraus resultierende Klartext angezeigt. Die Schlüssel werden in der Tabelle nach der besten Wertung der Kostenfunktion sortiert. Durch einen Doppelklick auf einen Tabellen-Eintrag ändern sich die Werte in den Ausgängen (berechneter Schlüssel und zugehöriger entschlüsselter Geheimtext).

4.6.3. Implementierung der Komponente

Wenn die Komponente aufgeführt wird, wird zunächst ein neues Objekt des verwendeten Analyse-Verfahrens erstellt. Dazu wird die Methode *GetAnalyzer* aufgerufen und überprüft, welcher Analyse-Modus ausgewählt wurde und ein entsprechendes Objekt zurückgegeben. Somit kann durch das Hinzufügen eines weiteren Eintrags einfach ein weiteres Verfahren unterstützt werden. Im nächsten Schritt wird dem Analyse-Objekt eine Kostenfunktion zugewiesen. Diese Kostenfunktion wird, ähnlich wie in der *GetAnalyzer*-Methode, durch die Methode *GetEvaluator* erzeugt und zurückgegeben.

Im nächsten Schritt wird die Visualisierung der Komponente zurückgesetzt, somit wird die Startzeit aktualisiert und Einträge aus vorherigen Ausführungen werden entfernt.

Anschließend beginnt die eigentliche Ausführung des Verfahrens, indem die Methode *Start* der abstrakten Klasse *AttackType* aufgerufen wird. Diese Methode bekommt als Eingabe den Geheimtext und muss von jedem Analyse-Verfahren implementiert werden, das durch die Komponente unterstützt werden soll. Diese Methode liefert als Ergebnis ein Objekt der Klasse *ResultEntry* zurück. Dieses Objekt enthält den Schlüssel mit der höchsten Bewertung und den entsprechenden Klartext. Der Schlüssel und der entsprechende Klartext werden anschließend an die Ausgänge der Komponente übergeben.

Im letzten Schritt wird die Darstellung der Metadaten mit den entsprechenden Werten aktualisiert. Dazu berechnet die Methode *UpdateDisplayEnd* die entsprechenden Werte und übergibt diese an die Visualisierung der Komponente.

Erweiterbarkeit der Komponente Da die Komponente einfach erweiterbar sein soll, werden in der *Execute*-Methode nur Interfaces und abstrakte Klassen verwendet. Somit ist es möglich, dass die Komponente das Verfahren analysieren kann, ohne Kenntnis darüber zu haben, wie die Analyse abläuft. Wie bereits erwähnt, werden Objekte für die Kostenfunktion und das Analyse-Verfahren basierend auf den Einstellungen der Komponente erstellt. Alle Kostenfunktionen implementieren das Interface *IEvaluator*. Dieses Interface definiert eine Methode, die für einen gegebenen String eine Gleitkommazahl zurückgibt. Somit kann die Logik zur Bewertung des Textes von dem Analyse-Verfahren getrennt werden. Es wird dadurch einfacher, neue Kostenfunktionen oder Analyse-Verfahren zu implementieren.

Alle Analyse-Verfahren implementieren die abstrakte Klasse *AttackType*. Diese Klasse definiert neben einigen Methoden, die für die Visualisierung notwendig sind, die Methode *Start*. Die Methode *Start* wird, wie oben beschrieben, in der *Execute*-Methode der Komponente aufgerufen. Sie erzeugt aus einem gegebenen Geheimtext ein *ResultEntry*-Objekt, das den Schlüssel und den Klartext enthält. Außerdem definiert die Klasse *AttackType* eine Eigenschaft vom Typ *IEvaluator*. Diese Eigenschaft enthält das Objekt

für die Kostenfunktion und wird, wie bereits oben beschrieben, in der *Execute*-Methode diesem Analyse-Verfahren zugewiesen.

4.7. Chaocipher

Dieser Abschnitt beschreibt die Anforderungen, die an die Chaocipher-Komponente gestellt wurden. Außerdem werden der Aufbau der Komponente und die Implementierung anhand des internen Ablaufs beim Ausführen der Komponente erläutert. Das Template, das zur Erläuterung der Funktionsweise der Komponente erstellt wurde, ist in Abbildung 38 zu sehen.

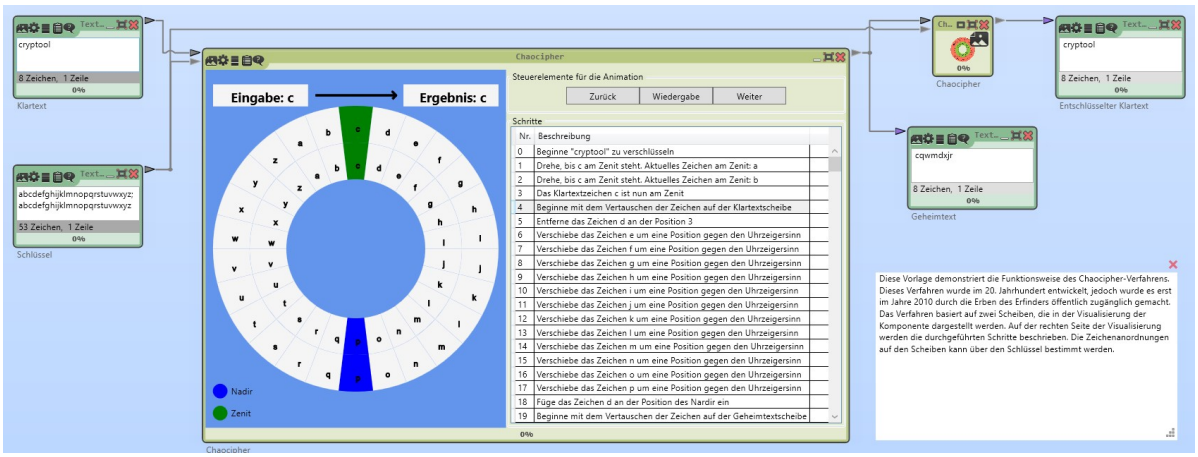

Abbildung 38: Template der Chaocipher-Komponente

4.7.1. Anforderungen

Da die Funktionsweise des Chaocipher-Verfahrens recht komplex ist, empfiehlt es sich, für diese Komponente eine animierte Visualisierung zu erstellen. Dabei sollen die einzelnen Schritte für den Nutzer möglichst einfach und nachvollziehbar präsentiert werden. Um die Visualisierung für den Nutzer attraktiver zu machen, ist eine Animation der beiden Scheiben, sowie das Vertauschen der Positionen wünschenswert.

Die Komponente soll als Eingaben nur den Schlüssel, also die Anordnung der Zeichen, auf den beiden Scheiben, und den zu verarbeitenden Text besitzen. Als Ausgang genügt für die Komponente der verarbeitete Text.

Die Eingabe des Schlüssels soll auf Plausibilität geprüft werden. Damit der Schlüssel plausibel ist, muss er aus einem Teil für die linke Scheibe und einem Teil für die rechte Scheibe bestehen und beide Teile müssen gleich lang sein.

4.7.2. Aufbau der Komponente

Im nachfolgenden Abschnitt werden die Ein- und Ausgänge sowie die Einstellungen der Komponente für das Chaocipher-Verfahren beschrieben. Die Komponente ist in Abbildung 39 abgebildet.

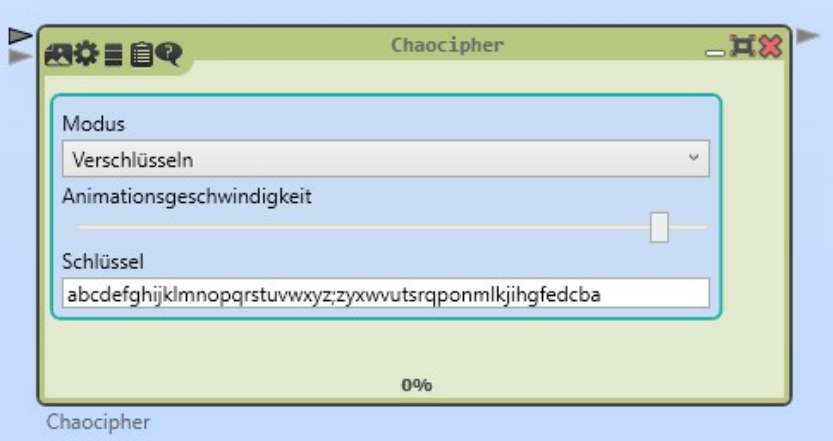

Abbildung 39: Einstellungen der Chaocipher-Komponente

Eingänge Die Komponente besitzt einen Eingang für den zu verarbeitenden Text sowie einen Eingang für den Schlüssel. Da der Schlüssel in zwei Teile geteilt ist, muss der Schlüssel in der Mitte immer ein Trennsymbol enthalten. Dies kann ein Semikolon, ein Zeilenumbruch oder ein Leerzeichen sein. Der Schlüssel kann alternativ auch über die Einstellungen der Komponente bereitgestellt werden.

Ausgänge Über den Ausgang der Komponente wird das Ergebnis der Ver- bzw. Entschlüsselung für andere Komponenten zur Verfügung gestellt.

Einstellungen Die Komponente besitzt drei Einstellungen, die in Abbildung 39 dargestellt sind. Die erste Einstellung bestimmt den Modus der Komponente, also ob die Komponente zum Ver- oder Entschlüsseln verwendet wird. Die zweite Einstellung bestimmt die Geschwindigkeit, mit der die Animation in der Visualisierung abgespielt wird. Diese Einstellung kann mittels eines Schiebereglers verändert werden, wobei die Geschwindigkeit rechts am schnellsten ist. Die letzte Einstellung bietet eine weitere Möglichkeit den Schlüssel festzulegen und wird automatisch mit dem Eingang für den Schlüssel synchronisiert.

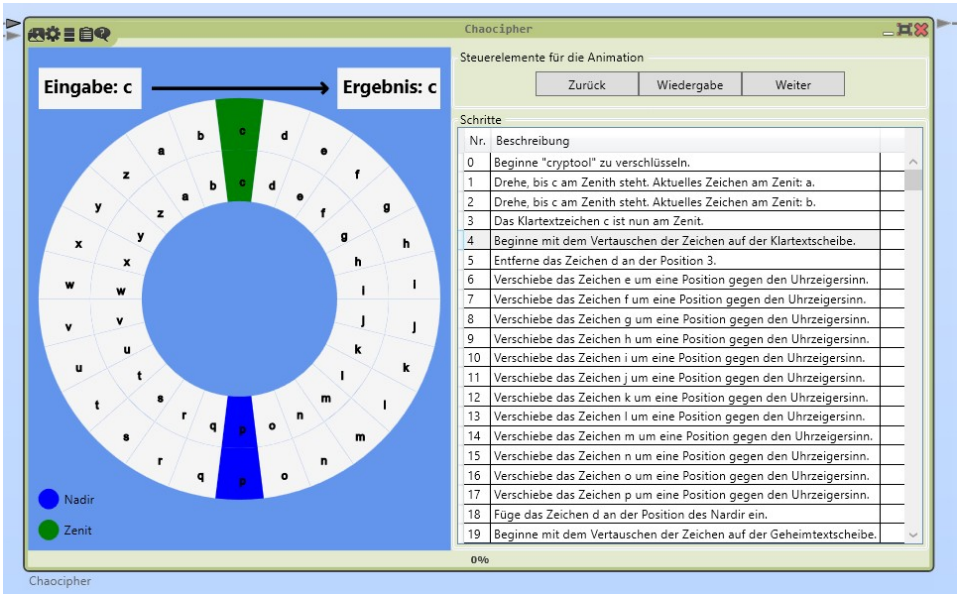

Abbildung 40: Visualisierung der Chaocipher-Komponente

Visualisierung Die Visualisierung der Chaocipher-Komponente wird in Abbildung 40 dargestellt. Auf der linken Seite der Visualisierung befindet sich die Animation der beiden Scheiben, deren Ablauf in Kapitel 3.6.1 beschrieben ist. Links oberhalb der Scheiben wird das aktuelle Zeichen angezeigt. Durch einen Pfeil zu der rechten oberen Box wird dargestellt, auf welches Zeichen das aktuelle Zeichen abgebildet wird. Um die Rotation der Scheiben zu verdeutlichen, werden im Mittelpunkt der Scheiben Kreise eingeblendet.

Auf der rechten Seite der Visualisierung befinden sich im oberen Bereich die Steuerelemente für die Animation. Mit ihnen kann die Animation gestoppt und fortgesetzt werden, außerdem kann ein Schritt zurück oder weiter gesprungen werden. Unterhalb der Steuerelemente befindet sich auf der rechten Seite die Anzeige aller Schritte, die zur Verarbeitung des Textes nötig sind. Der aktuelle Schritt wird dabei immer markiert. Zusätzlich wird jeder Schritt durch einen Text beschrieben. Die Beschreibung kann je nach Spracheinstellung des Benutzers auch in Englisch erfolgen. Durch einen Doppelklick auf einen Schritt in der Liste, springt die Animation zu diesem Schritt. Wenn die Animation durchgelaufen ist oder der Workspace beendet wird, stoppt die Animation.

4.7.3. Implementierung der Komponente

Für die Komponente wurden zuerst einige Tests erstellt, die testen, ob die Implementierung der Komponente beim Aufrufen der Ver- bzw. Entschlüsselungsmethode den erwarteten Geheimtext bzw. Klartext produziert. Die einzelnen Testfälle wurden aus

bekannten Klartext-Geheimtext-Paaren für das Chaocipher-Verfahren erstellt. [15] [12] Dank des TDD war es bereits während der Implementierung des Verfahrens möglich, sicherzustellen, dass die Implementierung korrekt ist. Nachdem alle Testfälle für das Verfahren erfolgreich ausgeführt wurden, konnte die Implementierung als korrekt angesehen werden. Im nächsten Schritt wurde, mittels des CT2-Templates für Plugins, ein neues Plugin erstellt. Das zuvor implementierte Verfahren wurde in die Komponente integriert und die Komponente um die Ein- und Ausgänge sowie die Einstellungen ergänzt.

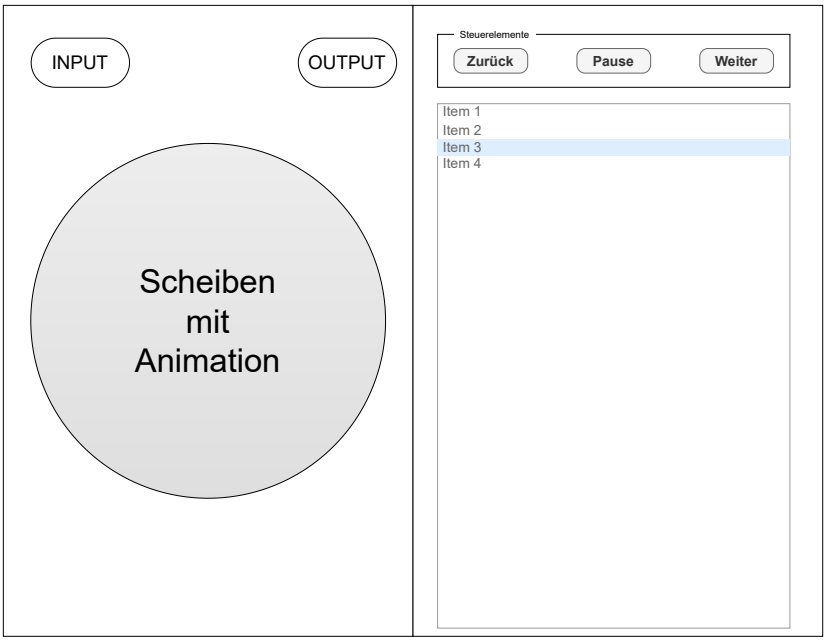

Abbildung 41: Mockup der Chaocipher-Komponente

Nachdem die Komponente erfolgreich zum Ver- und Entschlüsseln in einem CT2-Workspace genutzt werden konnte, wurde die Implementierung der Visualisierung begonnen. Dazu wurde zunächst ein Mockup erstellt, welches in Abbildung 41 zu sehen ist. Das Mockup wurde in WPF übertragen, indem zunächst ein Grid mit zwei Spalten erstellt wurde. Die linke Spalte beinhaltet ein Canvas, auf dem alle Elemente gezeichnet werden. Die rechte Spalte ist erneut mittels eines Grids unterteilt. Das Grid besteht aus zwei Zeilen und einer Spalte. In der oberen Zeile befinden sich die Steuerelemente in der unteren wird die Auflistung der einzelnen Schritte dargestellt.

Um ein Navigieren zwischen den Schritten möglichst effizient zu implementieren, werden während des Ver- bzw. Entschlüsselns die einzelnen Schritte in einem Objekt gespeichert. Für jeden Zustand besitzt dieses Objekt alle Informationen, die für die Darstellung notwendig sind, also den Zustand der Scheibe in diesem Schritt, die Zeichen, die in den Kästen über den Scheiben angezeigt werden sowie die Beschreibung für den einzelnen Schritt, der in der Liste rechts angezeigt wird. Dieses Objekt ist durch die Klasse *PresentationState* definiert. Die Ver- bzw. Entschlüsselung ist in der Klasse *CryptoService*

implementiert. Diese Klasse ruft nach jedem Schritt die Methode *AddPresentationState* auf, wodurch der aktuelle Zustand gespeichert wird. Außerdem wird gespeichert, in welchem Schritt sich der Prozess gerade befindet. Die möglichen Schritte werden im Enum *Step* definiert. Die Information über den aktuellen Schritt ist notwendig, um die passende Beschreibung für diesen Schritt zu generieren.

Nachdem die Ver- oder Entschlüsselung abgeschlossen ist, wird ein Objekt der Klasse *CipherResult* erstellt. Dieses Objekt beinhaltet das Ergebnis des Prozesses sowie die Beschreibung der Schritte. Durch die Methode *GenerateDescription* kann für jeden Schritt innerhalb des *CipherResult*s eine Beschreibung erstellt werden. Das *CipherResult* wird an die Visualisierung übergeben, die diese Daten einliest und visualisiert. Die Visualisierung zeigt zunächst einen Schritt an, wartet dann die in den Einstellungen vorgegebene Zeit, wechselt zum nächsten Schritt und wartet erneut. Dieser Vorgang wird wiederholt, bis alle Schritte dargestellt wurden.

Für die Auswahl des darzustellenden Zustandes ist die Klasse *AnimationPlayerService* zuständig. Sie beinhaltet einen Zähler, der nach jedem Schritt um eins erhöht wird. Zusätzlich enthält sie die Liste aller Zustände. Somit kann jeder Schritt sofort angezeigt werden, wenn der Nutzer zum Beispiel auf *Zurück* oder *Weiter* klickt oder durch einen Doppelklick auf einen Schritt zu diesem Schritt springt. Die Klasse implementiert außerdem das Event *NeedUpdate*. Das Event wird jedes mal ausgelöst, wenn die Visualisierung aktualisiert werden muss. Die Visualisierung abonniert dieses Event und aktualisiert die Anzeige, sobald das Event ausgelöst wird.

5. Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse der Arbeit zusammengefasst. Anschließend werden mögliche Ideen zur Weiterentwicklung der Komponenten erläutert.

5.1. Fazit

In der Einleitung (Kapitel 1) wurde definiert, dass im Rahmen dieser Arbeit sechs kryptografische Handverfahren und Codes behandelt werden. Diese Verfahren wurden zunächst in Kapitel 3 beschrieben, anhand von Beispielen erläutert und anschließend ihre kryptografischen Eigenschaften analysiert. In Kapitel 4 wurden diese Verfahren als Komponenten in CT2 implementiert.

T9-Code Für den T9-Code (Kapitel 3.1) wurde in dieser Arbeit eine Komponente implementiert, die eine Visualisierung in Form eines Mobiltelefons besitzt und interaktiv bedient werden kann. Zudem bietet diese Komponente die Möglichkeit, den T9-Code effizient mit einem beliebigen Wörterbuch auch wieder zu dekodieren (Kapitel 4.2).

Straddling-Checkerboard und Ché Guevara Das Straddling-Checkerboard-Verfahren und das Verfahren von Ché Guevara (Kapitel 3.2) wurden aufgrund ihrer Ähnlichkeit in eine einzelne Komponente zusammengefasst. Diese Komponente stellt, während der Ausführung, das verwendete Straddling-Checkerboard dar. Während der Ausführung passt sich diese Visualisierung dynamisch an die bereitgestellten Daten an. Es wurde gezeigt, dass das Straddling-Checkerboard mittels einer Häufigkeits-Analyse (Kapitel 2.2.3) gebrochen werden kann (siehe Kapitel 3.2.3).

Bacon-Chiffre Die Komponente der Bacon-Chiffre bietet viele Einstellungen, die es ermöglichen, die Funktionsweise des Verfahrens zu beeinflussen (Kapitel 4.4). Somit kann ein Benutzer die Komponente so anpassen, dass er einen Großteil der Anwendungsmöglichkeiten der Bacon-Chiffre mit der Komponente nachbilden kann. Einige denkbare Anwendungsmöglichkeiten des Verfahrens wurden in Kapitel 3.4 beschrieben.

Josse-Cipher Josses Cipher wurde in dieser Arbeit das erste Mal in einer öffentlich zugänglichen Version dokumentiert (Kapitel 3.5) und implementiert (Kapitel 4.5). Dazu mussten teilweise originale Aufzeichnungen aus dem Französischen übersetzt werden, um die Funktionsweise des Verfahrens möglichst original getreu nachbilden zu können.

Die Komponente erzeugt während der Ausführung, neben einer Visualisierung der internen Darstellung, auch ein Protokoll über die durchgeführten Rechenschritte, um die Funktionsweise für den Benutzer besser nachvollziehbar zu machen.

Außerdem wurde im Rahmen dieser Arbeit ein Angriff gegen dieses Verfahren auf der Basis von Lasry [11] implementiert. Die Funktionsweise der Komponente wurde in Kapitel 3.5.3 beschrieben und entsprechend als Komponente in CT2 implementiert (Kapitel 4.6).

Chaocipher Die Chaocipher (Kapitel 3.6) besitzt die aufwendigste Visualisierung aller Verfahren in dieser Arbeit. Sie zeigt interaktiv, wie die Scheiben für die Anwendung des Verfahrens genutzt werden. Parallel zu dieser Visualisierung wird eine Beschreibung des aktuellen Schrittes erstellt, die dem Benutzer erläutert, wieso eine Änderung der Zeichen auf den Scheiben stattgefunden hat (Kapitel 4.7).

Für alle Verfahren wurden in dieser Arbeit zudem Templates erstellt, die einen einfachen Ablauf zur Ver- und Entschlüsselung eines Beispieltextes in CT2 zeigen. Außerdem wurden sämtliche Beschreibungen und Beschriftungen der Eingänge, Ausgänge, Einstellungen und Visualisierungen für alle Komponenten übersetzt. Alle diese Texte stehen mindestens in deutscher und englischer Sprache zur Verfügung. Darüber hinaus sind einige Texte auch schon in chinesischer und russischer Sprache verfügbar, um der internationalen Verwendung von CT2 gerecht zu werden.

5.2. Ausblick

In diesem Kapitel werden mögliche Erweiterungen und Verbesserungen für die Komponenten erläutert.

T9-Code Das Laden des Wörterbuches für die T9-Code-Komponente dauert, je nach Größe des Wörterbuchs, einige Zeit. Dieser Vorgang muss bei jeder Änderung der Daten am Eingang für das Wörterbuch erneut durchgeführt werden. Da durch das Laden Wartezeiten entstehen, sollte diese Zeit durch weitere Optimierungen verkürzt werden.

Eine weiterer Ansatz zu Verbesserung der Nutzererfahrung kann die Visualisierung bieten, indem während der Eingabe über die Tastatur der Visualisierung bereits im Display des Mobiltelefons die möglichen Wörter angezeigt werden.

Bacon-Chiffre Die Bacon-Chiffre-Komponente könnte um einen zusätzlichen Modus ergänzt werden, bei dem die Komponente Informationen in einen Text einbettet. Dazu müsste die Komponente um einen weiteren Eingang ergänzt werden, der den Trägertext für die Komponente bereitstellt. Die Logik der Komponente müsste so angepasst werden, dass der Geheimtext in den Text eingebunden wird, anstatt direkt an den Ausgang ausgegeben zu werden.

Josse-Cipher Die Josse-Cipher-Komponente besitzt aktuell nur eine einfache Visualisierung mittels zweier Registerkarten, in denen sich jeweils ein Grid befindet. Die Visualisierung der Komponente könnte durch die Verwendung eines angepassten Grids verbessert werden. Dabei könnte das Grid so gestaltet werden, dass die erste Zeile des Grids bearbeitet werden kann, wodurch das Schlüsselwort für das Verfahren bestimmt wird.

Chaocipher Die Visualisierung der Chaocipher könnte weiter verbessert werden, indem die Scheiben durch eine flüssigere Rotation dargestellt werden. Außerdem könnten Komponenten für bereits bekannte Angriffe gegen das Verfahren erstellt werden. Mögliche Angriffe gegen das Verfahren wurden von Lasry, Rubin, Kopal und Wacker in [12] veröffentlicht.

A. Anhang: Texte zur Bewertung von Simulated-Annealing

Klartext benzinundmunitionisteingetroffenundkannindiejeweiligenkompanienweitergeleitetwerdenbittedirektansprechenergegenangrifferfolgummitternachtallesoldatenhabensichbereitzuhaltenalbrechtdonaufeldherrbertabertapeterdreidreineuneinsalbrechtdonauberta

Geheimtext verschlüsselt mit Schlüsselwort: „d“ whvvesnabpkyhakznwpjoxlsxrjyekpcymnydoblzdjoycdfodjqvjujwmoblqeagpjognsejsmrlinfglzblfzefoglwqsgzphmqycrwovdhmdbqxpyekphnbovqcqztnskydemgiugletfgibhvcfinauchpsxpucxxsdboinacpskptavwlzawbhtubhzruzrlnqvnhjzeycvwotbcvdjxbxlqzngiuxpuygdaqegaejavx

Geheimtext verschlüsselt mit Schlüsselwort: „an“ vijjvwtunoklxrnrsbwqxghrwqhygpuxsvampqtcdjvnmsrwfudmsvgxkncdpuxuakejbkqejvpuomsjqxyclfagmwovgncdyofmqnfiofpubijlpwozfovmvjyfcqdpichacehslocrwrfuncwbeprvnbwfkvzewbmggblocwbegvzrwajdjnbdniocityqhltkegkqhbdvnuatzrwfldkuxcznfrsmpbfzehslsijlimsjdf

Geheimtext verschlüsselt mit Schlüsselwort: „one“ qhlpedafqfbgvrbfjytpufjwnxrubmtxuyfwedixnixeqvuzkepochynsiqufkqpufagelsglownxwnvbhlufazblouzmiqupfndmochmgtyhmsyelfubmtlxoqnypdtpjqimuemiqdwnwzpybochvnglrkzftoczbrlpkybuqvzcvewnhvroafliuzpyiphaiphcjsxrltjqkrjumtdphmncifjqafegyczbkyuadiqluztpv

Geheimtext verschlüsselt mit Schlüsselwort: „chat“ oniohxuoynoextpgtplsasjwipofnzlfbwgxaysdciwicjmcmasbniypetibqbwzlzdsntmapaixtapoykajcfjyexrducehyqptflwqbtozjyszudbqhlwvdymzwpexteusjuyaisgykfwoyfjysyfozupaeoxedsdlnoutswibjtkeuyairjdolxewgmcwvaporhixrzghordbmlwlvufuozwqbqlgnfojyhemwnispykemx

Geheimtext verschlüsselt mit Schlüsselwort: „secrt“ foqujlhdymikaletzowckaciwchfnaiklbxjwynkhwqzpsrleujvfaxnjbnphnpnrythprlthcbovsbqckhvpwckzozdzamjoiyekthdpvbockhuochcmthpcsqtsuifjnrbfpziueuhnvylvdiwyhudpmujyhvfiwqxsziviuctltmxajrdyoqszluhwkyrbwgjncpsfjwnrdpvlotufiwqiwxucxuzbzjuclsgnqivwlutfp

Literaturverzeichnis

[1] *Baconian*. https://www.cryptogram.org/downloads/aca.info/ciphers/Baconian.pdf. [Online; letzter Zugriff: 20.05.2021].

[2] David Naccache und Rémi Géraud. „Invited Talk: A French Code from the Late 19th Century." In: *Proceedings of the 1st International Conference on Historical Cryptology, HistoCrypt 2018, Uppsala University, English Park Campus, Humanistiska teatern, Uppsala, Sweden, 18-20 June 2018*. Linköping University Electronic Press, 2018.

[3] Edward Fredkin. „Trie memory". In: *Communications of the ACM* 3.9 (Sep. 1960), S. 490–499. URL: https://dl.acm.org/doi/10.1145/367390.367400.

[4] Rémi Géraud-Stewart und David Naccache. „A French cipher from the late 19th century". In: *Cryptologia* 0.0 (2020), S. 1–29. URL: https://doi.org/10.1080/01611194.2020.1753265.

[5] Irwin Goldman. „William Friedman, Geneticist Turned Cryptographer". In: *Genetics* 206 (Mai 2017), S. 1–8. DOI: 10.1534/genetics.117.201624.

[6] Daniel James. *Ché Guevara: a biography*. 1st Cooper Square Press ed. New York: [Lanham, Md.]: Cooper Square Press ; Distributed by National Book Network, 2001. ISBN: 9780815411444.

[7] David Kahn. *The Codebreakers - The Story of Secret Writing*. New York: Macmillan, 1979. ISBN: 9780025604605.

[8] Oliver Kuhlemann. *Ché Guevara Chiffre*. https://kryptografie.de/kryptografie/chiffre/cheguevara.htm. [Online; letzter Zugriff: 16.05.2021].

[9] Oliver Kuhlemann. *Handycode*. https://kryptografie.de/kryptografie/chiffre/handy.htm. [Online; letzter Zugriff: 16.05.2021].

[10] Oliver Kuhlemann. *Spionage Chiffre (Straddling Checkerboard)*. https://kryptografie.de/kryptografie/chiffre/straddling-checkerboard.htm. [Online; letzter Zugriff: 07.12.2020].

[11] George Lasry. „Analysis of a Late 19th Century French Cipher Created by Major Josse". In: *Preview to be submitted* (2021).

[12] George Lasry, Moshe Rubin, Nils Kopal und Arno Wacker. „Cryptanalysis of Chaocipher and solution of Exhibit 6". In: *Cryptologia* 40.6 (2016), S. 487–514. DOI: 10.1080/01611194.2015.1091797.

[13] Christof Paar und Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Heidelberg u.a.: Springer Heidelberg Dordrecht London New York, 2010. ISBN: 978-3642041013.

[14] Rémi Géraud. „Advances in public-key cryptology and computer exploitation. (Avancées en cryptologie à clé publique et exploitation informatique)." Diss. 2017.

[15] Moshe Rubin. *Chaocipher revealed: The algorithm*. http://www.chaocipher.com/ActualChaocipher/Chaocipher-Revealed-Algorithm.pdf. [Online; letzter Zugriff: 20.05.2021]. Juli 2010.

[16] C. E. Shannon. „Communication theory of secrecy systems“. In: *The Bell System Technical Journal* 28.4 (Okt. 1949), S. 656–715. DOI: 10.1002/j.1538-7305.1949.tb00928.x.

[17] Olaf Versteeg. *Visualisierung und Implementierung von Betriebsmodi von Blockchiffren für CrypTool 2*. https://www.cryptool.org/assets/ctp/documents/BA_Versteeg.pdf. [Online; letzter Zugriff: 20.05.2021]. Juli 2018.

[18] Dietmar Wätjen. *Kryptographie Grundlagen, Algorithmen, Protokolle*. 3. Aufl. 2018. Wiesbaden: Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg, 2018. ISBN: 978-3-658-22474-5. DOI: 10.1007/978-3-658-22474-5.

[19] Axel Wehage. *Varianten von blinden Signaturen und ihre Implementierung in CrypTool 2*. https://www.cryptool.org/assets/ctp/documents/MA_Wehage.pdf. [Online; letzter Zugriff: 20.05.2021]. Sep. 2019.

Abkürzungsverzeichnis

ACA	American Cryptogram Association
CT2	CrypTool 2
TDD	Test-Driven-Design
OTP	One-Time-Pad
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, insbesondere keine anderen als die angegebenen Informationen aus dem Internet. Diejenigen Paragraphen der für mich geltenden Prüfungsordnungen, die etwaige Betrugsversuche betreffen, habe ich zur Kenntnis genommen.

Der Speicherung meiner Bachelor- bzw. Masterarbeit zum Zweck der Plagiatsprüfung stimme ich zu. Ich versichere, dass die elektronische Version mit der gedruckten Version inhaltlich übereinstimmt.

_____ _____

(Datum) (Unterschrift)