**UNIVERSITÄT KASSEL**
FACHBEREICH ELEKTROTECHNIK/INFORMATIK
FACHGEBIET ANGEWANDTE INFORMATIONSSICHERHEIT

**Masterarbeit**

# Kryptoanalyse homophoner Chiffren

Nils Rehwald
Matrikelnummer: 31202848

UNIKASSEL
VERSITÄT
Fachgebiet Angewandte Informationssicherheit
Fachbereich Elektrotechnik/Informatik
Universität Kassel

27. März 2017

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

# Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt haben. An erster Stelle geht dieser Dank an Professor Dr. Arno Wacker und M.Sc. Nils Kopal für das interessante Thema und die Möglichkeit, diese Arbeit im Fachgebiet „Angewandte Informationssicherheit“ der Universität Kassel anzufertigen sowie für die umfangreiche Betreuung während der Anfertigung der Arbeit.
Weiterhin möchte ich mich bei meiner Frau Mirjam und meiner Tochter Leia bedanken, die während der Anfertigung der Arbeit immer für mich da waren, mich moralisch unterstützt und immer wieder motiviert und aufgemuntert haben sowie sich um Aufgaben gekümmert haben, zu denen ich neben meiner Arbeit und dem Anfertigen der Masterarbeit keine Zeit gefunden habe.
Vielen Dank auch an Laura Köllstadt, Johanna Kölbel, Josefine Gück, Rebecca Reglin, Elke Reglin und Tim Wächter für das Lesen als fachfremde Personen, wodurch mir aufgezeigt wurde, an welchen Erklärungen besonders im Grundlagenbereich Verbesserungsbedarf bestand. Bei Josefine Gück möchte ich mich weiterhin für ihre Hilfe bei Problemen mit der spanischen Sprache bedanken. Ebenso möchte ich mich bei Konrad Kraft, Bastian Heuser und Dimitrij Borisov bedanken, die mich bei Problemen mit C# und LaTeX immer wieder unterstützt haben.
Nicht zuletzt möchte ich mich bei meiner Familie bedanken, die mir dieses Studium ermöglicht hat und die mich kontinuierlich hinterfragt und unterstützt hat.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einleitung

Diese Arbeit beschäftigt sich mit homophonen Verschlüsselungen. Dies ist eine Art der Verschlüsselung, bei der sich ein Buchstabe im Klartext nicht eindeutig auf einen Geheimtextbuchstaben abbilden lässt. Stattdessen hat der Verschlüsseler eine Auswahl von Geheimtextbuchstaben, von denen er einen auswählen kann, auf den er den Klartextbuchstaben abbildet. Zwei Fälle, in denen homophone Verschlüsselungen verwendet wurden, werden in dieser Arbeit näher betrachtet: Die Nachrichten des Zodiac Killers, eines Serienmörders, der in den 1960er und 1970er Jahren im amerikanischen Bundesstaat Kalifornien aktiv war, und die Spanish Strip Cipher, ein von beiden Seiten genutztes Verfahren im Spanischen Bürgerkrieg in den 1930er Jahren.

## 1.1 Motivation

Klassische Verschlüsselungen, die mit Stift und Papier ausgeführt werden können, werden in der heutigen Zeit nur noch selten genutzt. Mit der zunehmenden Verbreitung des Computers entwickelten sich auch die Verschlüsselungen weiter. Während im zweiten Weltkrieg noch Verschlüsselungsmaschinen wie die Enigma oder die Purple genutzt wurden, so werden heute Verfahren wie RSA oder AES in Computerprogrammen implementiert. Trotzdem gibt es gute Gründe, sich auch heute noch mit klassischer Kryptographie zu beschäftigen. Ein Grund ist der Einsatz klassischer Verschlüsselungen in der Lehre. Klassische Verschlüsselungen sind einfacher darzustellen, da sie in der Regel von Hand durchgeführt wurden. Daher ist es wichtig, sich weiterhin mit ihnen zu beschäftigen, weil dadurch Menschen leichter für Kryptographie begeistert werden können als durch moderne Verschlüsselungen, die von Computern realisiert werden. Klassische Verschlüsselungen kann man auch ohne Computerwissen mit Stift und Papier ausprobieren, wohingegen zum Beispiel zur Implementierung von RSA Kenntnisse in den Bereichen der Programmierung und Mathematik notwendig sind. Ein weiterer Grund für die Beschäftigung mit klassischer Kryptographie ist die Entschlüsselung von bis heute nicht entschlüsselten, historischen Nachrichten. Dies ist besonders für Historiker interessant. So gibt es zum Beispiel bis heute nicht entschlüsselte Enigma-Nachrichten, deren Dechiffrierung Historikern neue Informationen über den zweiten Weltkrieg liefern könnte. [Hoe] Ähnlich verhält es sich mit Nachrichten spanischer Republikaner, die möglicherweise mit der Spanish Strip Cipher verschlüsselt wurden. So gibt es einige verschlüsselte Nachrichten im baskischen Nationalarchiv, die noch nicht komplett

entschlüsselt wurden. [Nat] Nicht nur für Historiker, sondern auch für Kriminalisten interessant, sind die Nachrichten des Zodiac Killers. Da dieser nie gefasst wurde, könnte eine erfolgreiche Entschlüsselung seiner Nachrichten dazu führen, den Zodiac Killer zu überführen und vor Gericht zu bringen. Sollte dies nicht der Fall sein, so könnten seine Nachrichten dennoch interessante Details enthalten, die die damaligen Morde betreffen.

## 1.2 Aufgabenstellung

Die wörtliche Aufgabenstellung lautet wie folgt:
Homophone Chiffren sind monoalphabetische Verschlüsselungen, bei denen einzelne Buchstaben eines Klartextalphabets, z.B. das sechsundzwanzig-buchstabige lateinische Alphabet, auf mehrere Buchstaben eines Geheimtextalphabets abgebildet werden. Das Geheimtextalphabet kann z.B. aus Zahlen zwischen 1 und 100 bestehen, aber auch aus beliebigen anderen Zeichen. Das primäre Ziel von homophonen Verschlüsselungen ist die Verschleierung der, dem Klartext zugrunde liegenden, Textstatistik um so die Kryptoanalyse zu erschweren oder gar unmöglich zu machen. Aus diesem Grund werden bei vielen homophonen Verschlüsselungen besonders häufige Buchstaben, wie z.B. das „E“ oder das „N“, auf mehrere unterschiedliche Geheimtextzeichen abgebildet. Seltenere Buchstaben, wie z.B. das „X“ oder das „Q“, werden wiederrum häufig nur auf ein einziges Geheimtextzeichen abgebildet. Die Abbildungsfunktion von Klartextalphabet auf Geheimtextalphabet und umgekehrt stellt den kryptographischen Schlüssel einer homophonen Chiffre dar, und unterliegt daher der Geheimhaltung. Ziel einer Kryptoanalyse von homophonen Chiffren ist folglich die Bestimmung dieser Abbildung.
Bekannte Beispiele für homophone Chiffren sind unter anderem die Chiffren des sogenannten „Zodiac-Killers“ oder auch die „Spanische Strip-Chiffre“. [Sch12]
Der Zodiac-Killer war ein Serienmörder aus den USA, der in den Jahren 1968 und 1969 insgesamt fünf Menschen ermordete und zwei weitere Menschen schwer verletzte. Die Identität des Mörders konnte bis heute nie ermittelt werden und die Morde gelten als nicht aufgeklärt. Der Mörder versendete mehrere Briefe an verschiedene Lokalzeitungen und Fernsehsender, in denen er Details zu den Morden nannte. Des Weiteren verschickte der Mörder insgesamt vier verschlüsselte Nachrichten, welche die Chiffren „Zodiac-408“ und „Zodiac-340“, enthalten. Die Namen der Chiffren leiten sich von der Textlänge der Chiffren ab. Die 408 Zeichen lange (homophone) Chiffre wurde bereits 1969 eindeutig von Donald and Bettye Harden gelöst. Die drei weiteren Chiffren wurden bis heute nicht gelöst. Die kürzere und 340 Zeichen lange Chiffre ist vermutlich ebenso eine homophone Chiffre, aber aufgrund der deutlichen Kürze erheblich schwerer zu brechen, wenn nicht sogar unbrechbar. Die beiden weiteren Zodiac-Chiffren sind noch erheblich kürzer und sehr wahrscheinlich nicht brechbar. Amrapali et al. haben 2013 Hillclimbing-basierte Verfahren entwickelt, mit denen sie auch die Zodiac Killer Chiffre-340 analysiert haben [DAS13]. Leider konnten sie auch mit ihren Verfahren die Chiffre nicht brechen.

Die „spanische Strip-Chiffre“ ist eine homophone Chiffre, die während des spanischen Bürgerkriegs (1936-1939) sowohl von Republikanern als auch Nationalisten eingesetzt wurde. Bei der spanischen Strip-Chiffre wird ein Buchstabe des spanischen Alphabets (27 Buchstaben) auf jeweils drei oder vier Geheimtextbuchstaben eines Geheimtextalphabets (Zahlen z.B. von 01 bis 99) abgebildet. In ihrer Arbeit von 2015 haben Sanguino et al. [LABS16] ein neues Verfahren entwickelt, um die spanische Strip-Chiffre mit kombinatorischen und statistischen Methoden zu lösen. Im Zuge dieser Arbeit haben sie mehrere historische, mit der spanischen Strip-Chiffre verschlüsselte, Texte erfolgreich entschlüsselt. Sie konnten aber nicht alle Texte, welche in spanischen Archiven gefunden wurden, dechiffrieren.
Auf der Crypto Challenge Contest Webseite MysteryTwisterC3.org [MTC] gibt es mehrere „Challenges“ mit homophonen Chiffren. Unter anderem sind die originalen Zodiac Cipher Chiffren als auch die originalen Spanish Strip-Chiffre Chiffren auf MysterTwisterC3 verfügbar. Neben den originalen Chiffren gibt es auch mehrere ähnliche Chiffren, die den Zodiac Chiffren und den Spanish-Strip Chiffren nachempfunden sind. Diese sind auch lösbar und wurden bereits gelöst. Von daher können diese Chiffren als Probechiffren, für die in dieser Masterarbeit zu entwickelnden Verfahren, genutzt werden.
Hauptgegenstand dieser Masterarbeit ist es, (hillclimbing-/ und Dictionary-basierte) Analyseverfahren für die Kryptoanalyse von homophonen Chiffren zu entwickeln. Diese Verfahren sollen massiv parallel ausführbar sein. Dadurch können diese auf dem Rechnerpool des FB-16 der Universität Kassel aufgespielt und ausgeführt werden. Als Grundlage hierfür können die in [DAS13] und [LABS16] gezeigten Verfahren dienen. Des Weiteren können und müssen weitere Verfahren durch Recherche gefunden, evaluiert, bewertet und ggf. implementiert werden. Als Grundlage für die massiv-parallele Verteilung soll die am FG AIS entwickelte Verteilungsbibliothek „VoluntLib“ [KK] genutzt werden. Ziel ist die Analyse und ggf. Lösung der Zodiac Chiffren und der Spanish-Strip Chiffren.
Innerhalb dieser Masterarbeit sollen folgende konkrete Aufgaben und Ziele umgesetzt werden:

1. Analyse, Konzeption, Entwicklung und Implementierung von (hillclimbing- und Dictionarybasierten) Kryptoanalysewerkzeugen, die auf dem Rechnerpool des FB-16 der Universität Kassel aufgespielt und (massiv parallel) mittels „VoluntLib“ ausgeführt werden können.

2. Lösen der MysteryTwisterC3 (Übungs) Challenges zu den Zodiac-Chiffren und den Spanish- Strip-Chiffren mit Hilfe der selbst entwickelten Kryptoanalysewerkzeuge.

3. Analyse und (ggf) Lösung der „echten“ noch offenen Zodiac-Chiffren sowie Spanish-Strip- Chiffren. Da eine Lösung als sehr unwahrscheinlich gilt, ist die Lösung der Chiffren kein notwendiges Ziel dieser Arbeit.

Die Arbeit ist vollständig am Fachgebiet für Angewandte Informationssicherheit durchzuführen. Serverhardware wird bereitgestellt. Der Rechnerpool des FB-16 kann in Zusammenarbeit mit dem Betreuer der Masterarbeit genutzt werden. Alle

innerhalb der Arbeit getroffenen (Design-) Entscheidungen sollen in einer Ausarbeitung (der Masterarbeit) diskutiert werden. Neben der Ausarbeitung der Arbeit ist der Quellcode umfangreich zu dokumentieren. Alle in der Arbeit angefertigten Dokumente sowie der erstellte Quellcode sind in das Versionierungsverwaltungstool (Subversion) des Fachgebiets AIS einzuchecken. Abschließend ist eine Präsentation der in der Arbeit erreichten Ziele innerhalb des „Master-Kolloquiums“ vorzubereiten und durchzuführen.

Voraussetzungen für diese Arbeit: Sehr gute Kenntnisse im Bereich Kryptologie. Programmiererfahrung ist eine grundlegende Voraussetzung für diese Arbeit. Vorkenntnisse von .NET, C# und WPF sind notwendig.

## 1.3 Ziele der Arbeit

Das Ziel dieser Arbeit ist es, homophone Verschlüsselungen im Allgemeinen zu erklären und mögliche Angriffe zu erläutern. Weiterhin soll ein Angriff entwickelt werden, der sich auf verschiedene homophone Verschlüsselungen anwenden lässt. Dieser soll anhand bekannter Nachrichten des Zodiac Killers und der Spanish Strip Cipher getestet und anschließend auf bisher unentschlüsselte Nachrichten dieser Verschlüsselungen angewandt werden. Neben der schriftlichen Ausarbeitung, in der dieser vorgestellt und analysiert werden soll, wird der Angriff mit Hilfe der „VoluntCloud“ implementiert, sodass dieser parallel ausführbar ist.

- (R01) Analyse, Konzeption, Entwicklung und Implementierung von (Hillclimbing- und Dictionarybasierten) Kryptoanalysewerkzeugen, die auf dem Rechnerpool des FB-16 der Universität Kassel aufgespielt und (massiv parallel) mittels „VoluntLib“ ausgeführt werden können.
- (R02) Lösen der MysteryTwisterC3 (Übungs)challenges zu den Zodiac-Chiffren und den Spanish-Strip-Chiffren mit Hilfe der selbstentwickelten Kryptoanalysewerkzeuge.
- (R03) Analyse und (ggf) Lösung der „echten“ noch offenen Zodiac-Chiffren sowie Spanish-Strip-Chiffren. Da eine Lösung als sehr unwahrscheinlich gilt, ist die Lösung der Chiffren kein notwendiges Ziel dieser Arbeit.

Im Anschluss sollen die Stärken und Schwächen des Angriffes anhand durchgeführter Messungen ausgearbeitet und analysiert werden. Wenn während der Erarbeitung dieser Arbeit mögliche Verbesserungen des Algorithmus entdeckt werden, sollen diese ebenfalls dargelegt werden.

## 1.4 Aufbau der Arbeit

Die Arbeit ist in acht Kapitel unterteilt. Im auf die Einleitung folgenden Kapitel 2 werden die zum Verständnis der Arbeit erforderlichen Grundlagen der Kryptographie und Kryptologie erläutert. Weiterhin wird in diesem Kapitel eine kur-

ze Erklärung zur für den programmiertechnischen Teil der Arbeit notwendigen VoluntCloud gegeben. Kapitel 3 handelt von homophonen Verschlüsselungen im Allgemeinen, ihrer Funktionsweise und kryptoanalytischen Gesichtspunkten. Weiterhin wird in diesem Kapitel auf historische homophone Verschlüsselungen, die Nachrichten des Zodiac Killers und die spanische Streifenchiffre, eingegangen. In Kapitel 4 wird ein Ansatz für einen Angriff auf homophone Verschlüsselungen im Allgemeinen vorgestellt. Im Verlauf des Kapitels wird auf Verbesserungen dieses Angriffes sowie Besonderheiten im Hinblick auf die Spanish Strip Cipher eingegangen. Die parallele Implementierung des Angriffs, optimiert um die Nachrichten des Zodiac und mit der spanischen Streifenchiffre verschlüsselten Texte zu brechen, wird in Kapitel 5 vorgestellt. Die Evaluation der Implementierung folgt in Kapitel 6, welches die Qualität des Angriffs einordnet. Im darauffolgenden Kapitel 7 wird ein Überblick über andere Arbeiten zum Thema homophone Chiffren gegeben. Abschließend werden mögliche zukünftige Verbesserungen des Angriffs in Kapitel 8 vorgestellt und diskutiert.

# 2 Grundlagen

Dieses Kapitel soll einen Überblick über die zum Verständnis der Arbeit notwendigen Grundlagen geben. Dazu werden zuerst die allgemeinen Begriffe Kryptographie und Kryptoanalyse erläutert. Weiterhin wird auf verschiedene Formen von Angriffen auf Verschlüsselungen eingegangen. Zuletzt folgt eine Erklärung der „VoluntCloud“, welches eine für die parallele Ausführung der Angriffe im Rahmen der Masterarbeit nötiges Programm ist.

## 2.1 Kryptographie

Die Kryptographie ist eine Wissenschaft, die sich mit der Verschlüsselung von Daten beschäftigt. Das Ziel ist es, eine sichere Kommunikation zwischen zwei Parteien zu ermöglichen. Dies bedeutet, dass Nachrichten, die von Partei A an Partei B gesendet werden, nur von Partei B gelesen werden können. Sollte es einer dritten Partei C gelingen, die Nachricht abzufangen, so ist die Nachricht für diese Partei wertlos, da sie nicht gelesen werden kann. Dies gelingt im Rahmen der Kryptographie dadurch, dass die Nachricht von Partei A vor dem Versenden mit einem geheimen Schlüssel, der nur Partei A und Partei B bekannt ist, verschlüsselt wird. Ein ähnliches Ziel verfolgt die Steganographie, welche sich jedoch in der Umsetzung von der Kryptographie unterscheidet.
Während die Kryptographie darauf zielt, dass eine Nachricht, auch wenn sie jedem Menschen bekannt ist, nur von den Menschen, an die sie adressiert war, gelesen werden kann, ist das Ziel der Steganographie, die Nachricht so zu verstecken, dass sie nur von eingeweihten Personen gefunden werden kann. Ein klassisches Beispiel für ein steganographisches Verfahren ist unsichtbare Tinte. Während das Ziel der Kryptographie früher hauptsächlich die sichere Übermittlung geheimer Nachrichten war, so kamen im Laufe der Zeit neue Ziele hinzu. Heute umfasst die Kryptographie auch folgende Ziele: [Wac14]

- Datenintegrität: Dies bedeutet, dass eine Nachricht von Partei A an Partei B nicht von einer dritten Partei abgefangen und verändert werden kann, ohne dass dies dem Empfänger auffällt. Der Empfänger kann also prüfen, ob eine Nachricht unterwegs manipuliert wurde.
- Verantwortlichkeit: Dies bedeutet, dass der Empfänger dem Verfasser einer Nachrich nachweisen kann, dass er der Sender der Nachricht ist. Es ist nicht möglich, zu behaupten, ein Dritter und nicht man selbst habe eine Nachricht verfasst.

- Authentizität: Der Empfänger kann nachweisen, dass die Nachricht von einer bestimmten Person verfasst wurde. Es ist also nicht möglich, eine Nachricht unter dem Namen eines Dritten zu versenden, ohne dass der Empfänger dies bemerken kann.
- Zugriffskontrolle: Dies bedeutet, dass eine Ressource, zum Beispiel eine Datei oder eine Nachricht, so abgesichert ist, dass nur bestimmte Personen, denen dies gestattet ist, Zugriff auf die Ressource haben.

Im Rahmen dieser Arbeit wird der Fokus auf das Ziel der Geheimhaltung gelegt, da das Ziel der Arbeit ist, Angriffe auf bestimmte Verschlüsselungen zu entwerfen. In der Kryptographie werden in der Regel folgende Abkürzungen verwendet:

- P - Plaintext: Der Klartext, der einem Text in einer menschlichen Sprache entspricht
- C - Ciphertext: Der Geheimtext, der durch die Anwendung einer Verschlüsselung aus dem Klartext erzeugt wird. Dieser wird vom Sender an den Empfänger übermittelt, welcher diesen wieder in Klartext übersetzen kann.
- K - Key: Der Schlüssel, der verwendet wird, um einen Klartext zu verschlüsseln und einen Geheimtext anschließend wieder zu entschlüsseln.
- $\mathcal{K}$ - Keyspace: Der Schlüsselraum, aus dem ein Schlüssel entnommen werden kann. Dies entspricht der Menge aller möglicher Schlüssel, die bei Verwendung einer bestimmten Verschlüsselung genutzt werden können.

Im Allgemeinen lassen sich Verschlüsselungen in symmetrische und asymmetrische Verschlüsselungen unterteilen. Während bei symmetrischen Verschlüsselungen derselbe Schlüssel genutzt wird, um die Nachricht zu ver- und entschlüsseln, werden bei asymmetrischen Verfahren verschiedene Schlüssel genutzt. Diese verschiedenen Schlüssel werden als öffentlicher und geheimer Schlüssel bezeichnet. Der öffentliche Schlüssel darf jedem bekannt sein. Ein Beispiel für eine asymmetrische Verschlüsselung ist der RSA Algorithmus. Mit dem öffentlichen Schlüssel werden Nachrichten verschlüsselt, die anschließend nur durch den dazugehörigen geheimen Schlüssel entschlüsselt werden können. Durch diese Idee wird die Problematik des Schlüsselaustauschs eliminiert. Im Gegensatz dazu verwenden bei einer symmetrischen Verschlüsselung beide Kommunikationspartner denselben Schlüssel, um die Nachricht zu ver- und wieder entschlüsseln. Dies bringt das Problem mit sich, dass sich beide Partner auf einen gemeinsamen Schlüssel einigen müssen und eine Möglichkeit finden müssen, diesen Schlüssel auszutauschen. Auch muss jedes Mal, wenn der Schlüssel kompromittiert wurde, ein neuer Schlüssel festgelegt und ausgetauscht werden. Erst in den 1970er Jahren fanden Whitfield Diffie und Martin Hellman eine Möglichkeit, einen gemeinsamen Schlüssel zu generieren, ohne sich dafür treffen zu müssen. Alle klassischen Chiffren sind symmetrisch, aber zum Beispiel auch der moderne DES Algorithmus funktioniert symmetrisch. Aus Gründen der Performance werden asymmetrische Verschlüsselungen auch heute meist nur genutzt, um einen symmetrischen Schlüssel für ein Verfahren, mit dem der folgende Datenverkehr verschlüsselt wird, auszutauschen.

Weiterhin kann man Verschlüsselungen in Substitutions- und Transpositionschiffren unterteilen. Transpositionschiffren verschlüsseln Nachrichten, indem sie die Position der Buchstaben im Text verändern. So kann man den Klartext

TRANSPOSITIONSCHIFFRE

zum Beispiel in den Geheimtext

ERFFIHCSNOITISOPSNART

übersetzen. Die Buchstaben sind noch die Gleichen, nur die Anordnung hat sich verändert. Der Empfänger muss zum Entschlüsseln die Buchstaben wieder in die richtige Reihenfolge bringen. Ein Beispiel für eine Transpositionschiffre ist die Doppelte Spaltentransposition. [LKW14] Substitutionschiffren funktionieren, indem sie Buchstaben durch andere Buchstaben ersetzen. Dabei kann man zwischen monoalphabetischen und polyalphabetischen Substitutionschiffren unterscheiden. Monoalphabetische Verschlüsselungen bilden einen Buchstaben des Klartextes immer auf exakt denselben Buchstaben des Geheimtextes ab. Ein Beispiel hierfür ist die Caesar-Chiffre. In dieser wird jeder Klartextbuchstabe im Geheimtext durch den Buchstaben ersetzt, der im Alphabet drei Stellen nach ihm folgt. So wird aus dem Klartext

CAESARCHIFFRE

der Geheimtext

FDHVDUFKLIIUH

Zum Entschlüsseln muss anschließend jeder Geheimtextbuchstabe wieder durch den Buchstaben, der im Alphabet 3 Stellen vorher steht, ersetzt werden. In der Gattung der Substitutionschiffren entwickelten sich aus diesen die polyalphabetischen Verschlüsselungen. Diese bilden einen Klartextbuchstaben nicht immer auf den gleichen Geheimtextbuchstaben ab, sondern verwenden verschiedene Abbildungsmuster. Welches Abbildungsmuster für einen Buchstaben verwendet wird, hängt in diesem Fall von der Position des Buchstabens im Text ab. Ein Beispiel hierfür ist die Vigenére Verschlüsselung. Diese verwendet ein Schlüsselwort der Länge $n$. Außerdem werden $n$ verschiedene monoalphabetische Verschlüsselungen verwendet, die zur Erklärung von *0* bis *n-1* durchnummeriert sind. An jeder Stelle x des Textes wird nun zum Ver- oder Entschlüsseln die Monoalphabetische Verschlüsselung mit der Nummer $x \ mod \ n$ verwendet. Zwei aufeinanderfolgende, gleiche Klartextbuchstaben, werden also nicht zwingend auf dieselben Geheimtextbuchstaben abgebildet. So kann es sein, dass der Klartext

HALLO

auf den Geheimtext

KBPQA

abgebildet wird. Ist der Text länger als das Schlüsselwort, so bezeichnet man dies als periodische, polyalphabetische Verschlüsselung. Diese Benennung resultiert daraus, dass nach Ende des Schlüsselworts dasselbe Schlüsselwort wieder verwendet wird. In diesem Fall kann man alle Stellen, an denen dieselbe Stelle des Schlüsselwortes verwendet wurde, zusammenfassen und erhält dadurch n verschiedene Texte, die mit einer monoalphabetischen Substitution verschlüsselt wurden.

## 2.2 Kryptoanalyse

Die Kryptoanalyse ist eine eng mit der Kryptographie verknüpfte Wissenschaft, die sich mit der Analyse und Bewertung von kryptographischen Verschlüsselungen beschäftigt. Sie entstand durch das Bedürfnis, abgefangene verschlüsselte Nachrichten dechiffrieren zu können. Heute beschäftigt sich die Kryptoanalyse weiterhin mit der Bewertung von Verschlüsselungen. Dabei werden kryptographische Verfahren auf ihre Qualität im Bezug auf Sicherheit analysiert. Die Analyse der Algorithmen soll sicherstellen, dass mögliche Schwachstellen von Verschlüsselungen möglichst schnell gefunden und, wenn möglich, repariert werden können. Bevor Algorithmen von der breiten Massen von Endanwendern benutzt werden, vergeht in der Regel ein Zeitraum, in dem die Algorithmen ausführlich analysiert werden. Durch eine qualitativ hochwertige Analyse wird so verhindert, dass unsichere Algorithmen, die Schwachstellen enthalten können, von einer großen Nutzerzahl verwendet werden.
Ein wichtiges Prinzip in der Kryptoanalyse ist das „Kerckhoff'sche Prinzip“. Dieser 1883 von Auguste Kerckhoff formulierte Grundsatz besagt, dass die Sicherheit einer Verschlüsselung nur vom verwendeten Schlüssel und keinem anderen Parameter abhängen darf. Man muss also bei der Kryptoanalyse davon ausgehen, dass die komplette Funktionsweise einer Verschlüsselung dem Angreifer bekannt ist. Die Sicherheit gegen den Angreifer wird lediglich durch die Verwendung eines sicheren Schlüssels gewährleistet. Daher ist die Größe des Schlüsselraums ein wichtiger Faktor für die Qualität einer Verschlüsselung. Der Schlüsselraum ist die Menge aller möglichen, validen Schlüssel, die für eine Chiffre verwendet werden können. Diese muss so groß sein, dass es mit vertretbaren Mitteln nicht möglich ist, den kompletten Schlüsselraum einer Chiffre zu testen. Ist der Schlüsselraum zu klein, kann die Verschlüsselung einfach gebrochen werden, indem alle Schlüssel ausprobiert werden und ist somit nicht sinnvoll einsetzbar. Für Kryptoanalytiker wichtige Werkzeuge zur Analyse verschlüsselter Nachrichten sind statistische Analysen. Eine davon ist der erstmals 1922 publizierte Koinzidenzindex. Dieser lässt sich aus einem Text berechnen und beschreibt, wie nah ein Text einer natürlichen oder einer künstlichen

Sprache ist [Fri79]. Jede natürliche Sprache hat eine eigene Charakteristik. In der deutschen Sprache kommen zum Beispiel die Buchstaben „E“, „N“und „I“ sehr häufig vor, im Englischen die Buchstaben „E“, „T“ und „A“. So hat jede Sprache verschiedene Charakteristiken. Jedoch sind sich alle Sprachen in dem Punkt ähnlich, dass es immer Buchstaben gibt, die häufiger vorkommen und Buchstaben, die seltener vorkommen. Der Koinzidenzindex kann genutzt werden, um zu überprüfen, ob ein Text einer natürlichen Sprache mit unterschiedlich häufig vorkommenden Buchstaben oder einer künstlichen Sprache mit möglicherweise gleich häufig vorkommenden Buchstaben ähnlich ist. Dadurch kann der Analyst darauf schließen, ob eine Transposition, eine monoalphabetische oder eine polyalphabetische Substitution eingesetzt wurden.
Eine Transpositionschiffre ersetzt keine Buchstaben sondern ändert nur die Reihenfolge und hat entsprechend keinen Einfluss auf den Koinzidenzindex. Eine monoalphabetische Substitution ersetzt zwar Buchstaben, allerdings wird ein Klartextbuchstabe immer durch denselben Geheimtextbuchstaben ersetzt. Da der Koinzidenzindex nicht davon abhängt, wie oft ein bestimmter Buchstabe vorkommt, sondern nur von der gesamten Häufigkeitsanalyse aller Buchstaben, hat eine monoalphabetische Verschlüsselung ebenfalls keinen Einfluss auf diesen Wert. Anders gestaltet sich dies bei einer polyalphabetischen Substitution. Da die Ersetzung eines Buchstaben abhängig von der Position dessen im Text ist, ist es wahrscheinlich, dass die Buchstabenhäufigkeit verändert wird. Dies führt dazu, dass sich der Koinzidenzindex ändert. Während eine leichte Abweichung des Koinzidenzindex von dem Wert einer natürlichen Sprache erwartbar ist, besonders bei kurzen Nachrichten, da ein Text immer nur einen Teil einer Sprache repräsentiert, kann man bei größeren Abweichungen feststellen, dass eine Verschlüsselung, die die Buchstabenhäufigkeiten manipuliert, verwendet wurde.
Eine Besonderheit bei periodischen, polyalphabetischen Substitutionen ist, dass man mit Hilfe des Koinzidenzindex auch die Länge der Periode abschätzen kann. Dadurch wird es möglich, auch diese Chiffren mit Hilfsmitteln, die für monoalphabetische Substitutionen gedacht sind, anzugreifen. Dies funktioniert, indem man die Länge der Periode abschätzt. Angenommen, diese ist $n$. Nun kann, beginnend am Anfang des Textes, jeder $n$-te Buchstaben ausgewählt und hintereinander aufgeschrieben werden. Anschließend kann man dies, beginnend beim 2. Buchstaben des Textes bis zum $n$-ten Buchstaben des Textes wiederholen, und erhält so n Listen von Buchstaben. Da alle Buchstaben einer Liste mit derselben monoalphabetischen Substitution verschlüsselt wurden, kann man diese Substitution zum Beispiel mit Hilfe der Häufigkeitsanalyse einzelner Buchstaben analysieren. Je mehr Buchstaben in einer Liste vorhanden sind, umso besser lässt sich eine solche Statistik erstellen. Daher ist diese Art der Verschlüsselung sicherer, je länger die Periode und je kürzer der Text ist.
Ein weiterer Indikator, der mit der Textlänge zusammenhängt, ist die Unizitätslänge. Diese ist abhängig von der Entropie des Schlüsselraums und der Redundanz einer Sprache, die für jede Sprache eindeutig berechenbar ist. Die Unizitätslänge gibt an, wie lang ein Geheimtext mindestens sein muss, damit es einen eindeutigen Schlüssel gibt, der diesen Geheimtext auf einen sinnvollen Klartext abbildet.

Ist ein Geheimtext kürzer als die Unizitätslänge, so ist es nicht eindeutig möglich, einen Schlüssel zu bestimmen, der diesen Geheimtext auf einen sinnvollen Klartext abbildet. In diesem Fall ist es möglich, dass mehrere Schlüssel existieren, die den selben Geheimtext auf verschiedene, sinnvolle Klartexte abbilden. Es wäre dann also für einen Kryptoanalytiker nicht möglich, diese Nachricht ohne gesicherte Kenntnis des Schlüssels zu dechiffrieren.

## 2.3 Angriffe

Je nach Art der verwendeten Verschlüsselung gibt es verschiedene mögliche Angriffe auf diese. Generell lassen sich diese in drei verschiedene Kategorien einteilen. Zu welcher Kategorie ein Angriff gehört hängt davon ab, welche Informationen zum Ausführen des Angriffes notwendig sind.

### 2.3.1 Known-Plaintext Angriff

Bei einem Known-Plaintext Angriff steht dem Angreifer ein Textpaar von Klartext und Geheimtext zur Verfügung. Weiterhin ist bekannt, welche Verschlüsselung verwendet wurde, um den Klartext auf den Geheimtext abzubilden. Dem Angreifer ist bekannt, dass der abgefangene Geheimtext zu einem bestimmten, bekannten Klartext gehört. Das Ziel des Angreifers ist es in diesem Fall, den verwendeten Schlüssel herauszufinden. Zwar bedeutet das Bestimmen des verwendeten Schlüssels keinen direkten Informationsgewinn für den Angreifer, da der Klartext der Nachricht bereits vorher bekannt war. Jedoch kann der so erlangte Schlüssel verwendet werden, um weitere abgefangene Geheimtextnachrichten, deren dazugehöriger Klartext nicht bekannt ist, anzugreifen. Eine Möglichkeit hierfür ist, dass derselbe Schlüssel zum Verschlüsseln mehrerer Nachrichten verwendet wurde. Zum Beispiel können alle Nachrichten eines Tages mit einem Tagesschlüssel, der jeden Tag geändert wird, verschlüsselt werden. Wird durch einen Known-Plaintext Angriff dieser Tagesschlüssel geknackt, so kann der Angreifer alle abgefangenen Nachrichten dieses Tages entschlüsseln. Eine weitere Möglichkeit zur Nutzung eines durch einen Known-Plaintext erhaltenen Schlüssels ist der Aufbau einer Datenbank. Je nachdem, auf welchem Weg die Schlüssel generiert werden, kann es möglich sein, mit Hilfe einer möglichst großen Datenbank von bereits verwendeten Schlüsseln auf mögliche zukünftige Schlüssel zu schließen. Die Bedingung hierfür ist, dass die Schlüssel immer auf einem bestimmten Weg generiert werden, wodurch diese Ähnlichkeiten aufweisen, anhand derer man vorhersehen kann, welche Kriterien ein zukünftiger Schlüssel wahrscheinlich erfüllt. Um einen Known-Plaintext Angriff auszuführen muss der Angreifer das Verschlüsseln des Klartextes mit der bekannten Verschlüsselung simulieren. Je nach verwendeter Verschlüsselung kann der Algorithmus sehr trivial, zum Beispiel bei einer monoalphabetischen Substitution, oder komplex, zum Beispiel bei modernen Chiffren wie dem RSA Algorithmus sein. Bei einer monoalphabetischen Substitution muss lediglich das Mapping von

Klartextalphabet zu Geheimtextalphabet aus den bekannten Nachrichten abgelesen werden. In diesem Fall kann man von den Nachrichten einfach ablesen, welcher Klartextbuchstabe auf welchen Geheimtextbuchstaben abgebildet wird. Sehr komplex wäre hingegen ein Known-Plaintext Angriff auf eine RSA Verschlüsselung. Bei dieser ist es mit vertretbaren Mitteln nicht möglich, aus einem Textpaar auf den verwendeten Schlüssel zu schließen.

### 2.3.2 Partially Known-Plaintext Angriff

Das Prinzip des Partially Known-Plaintext Angriffes ist im Ansatz ähnlich wie das des Known-Plaintext Angriffs. Auch in diesem Fall sind dem Angreifer die verwendete Verschlüsselung sowie eine abgefangene, verschlüsselte Nachricht bekannt. Allerdings ist im Gegensatz zum Known-Plaintext Angriff nicht die komplette Bedeutung der Nachricht bekannt. Dem Angreifer ist in diesem Fall nur ein Teil des Klartextes bekannt. Dies kann zum Beispiel durch das Knacken vieler Nachrichten und daraus resultierendem Erkennen von Regelmäßigkeiten in diesen passieren. Eine Möglichkeit ist, dass der Angreifer erkennt, dass Nachrichten immer mit demselben Text beginnen oder enden. Die Ziele eines Partially Known-Plaintext Angriffs umfassen die eines Known-Plaintext Angriffs. Weiterhin zielt ein solcher Angriff auf einen direkten Informationsgewinn, das vollständige Entschlüsseln der Nachricht, ab. Diese Art des Angriffs funktioniert in zwei Stufen. In der Ersten wird ein normaler Known-Plaintext Angriff auf den bekannten Klartext und das dazugehörige Stück Geheimtext ausgeführt. Dieser Angriff führt im Idealfall dazu, dass Teile des Schlüssels festgelegt werden können. Je länger der bekannte Klartext ist, umso wahrscheinlicher wird es, Teile des Schlüssels bestimmen zu können. Diese bekannten Teile werden nun als feststehend markiert. Anschließend wird auf den gesamten Geheimtext ein Ciphertext-Only Angriff ausgeführt. Allerdings wird dieser in dem Punkt eingeschränkt, dass die bereits als feststehend markierten Teile des Schlüssels nicht mehr verändert werden. Die konkrete Funktionsweise des Angriffs hängt, wie schon beim Known-Plaintext Angriff, von der Art der verwendeten Verschlüsselung ab.

### 2.3.3 Chosen Plaintext Angriff

Dieser Angriff kann angewendet werden, wenn dem Angreifer keine Paare von Klar- und Geheimtext zur Verfügung stehen, dieser jedoch die Möglichkeit hat, selbst gewählten Klartext zu verschlüsseln und somit frei gewählte Textpaare erstellen kann. Das Ziel des Angriffs ist, die Verschlüsselung durch Reverse Engineering zu brechen. Zur Durchführung eines solchen Angriffs gibt es zwei verschiedene Möglichkeiten. Die Erste ist, dass der Angreifer Zugang zu einer Verschlüsselungsmaschine wie die Enigma bekommt, zum Beispiel durch die Erbeutung im Gefecht. Mit dieser Maschine können nun beliebig viele Klartexte verschlüsselt und die Chiffre analysiert werden. Auf diesen Weg kann der Angreifer

gute Grundlagen für statistische Analysen schaffen, indem er möglichst viel Text verschlüsselt. Auf diesem Weg kann herausgefunden werden, welche Art von Verschlüsselung die Maschine verwendet. Durch das häufige Verschlüsseln desselben Buchstabens kann der Angreifer außerdem eine periodische polyalphabetische Verschlüsselung analysieren. Die zweite Möglichkeit ist auch anwendbar, wenn kein direkter Zugriff auf eine Verschlüsslungsmaschine besteht. In diesem Fall wird der Feind provoziert, Nachrichten zu versenden, deren Inhalt dem Angreifer mit einer hohen Wahrscheinlichkeit bereits bekannt ist. Dies kann zum Beispiel durch das Antäuschen eines Angriffs auf den Feind erfolgen, den dieser wahrscheinlich melden wird. Wurde vorher bereits eine entsprechende Nachricht dechiffriert, so kann der Inhalt der nun versendeten Nachricht mit hoher Wahrscheinlichkeit vorhergesagt werden. Eine andere Möglichkeit ist die Streuung falscher Informationen. So kann zum Beispiel ein Funkspruch abgesetzt werden, der mit einem Schlüssel verschlüsselt wird, von dem bekannt ist, dass der Feind diesen knacken kann. Da die Nachricht vom Feind wahrscheinlich abgefangen und entschlüsselt wird ist davon auszugehen, dass der Inhalt der Nachricht der nächsten Instanz gemeldet wird. Dieser Funkspruch wiederum kann abgefangen werden, wobei ein großer Teil der Nachricht als bekannt angenommen werden kann, nämlich der absichtlich schlecht verschlüsselte eigene Funkspruch.

### 2.3.4 Ciphertext-Only Angriff

Der Ciphertext-Only Angriff ist die Art von Attacke, die mit den wenigsten bekannten Informationen ausgeführt werden kann. Dem Angreifer muss lediglich eine abgefangene, verschlüsselte Nachricht vorliegen. Weiterhin ist es hilfreich, wenn dem Angreifer bekannt ist, welche Chiffre verwendet wurde, um den Geheimtext zu erstellen. Ist dies nicht bekannt, muss dieser versuchen, die Menge möglicher verwendeter Chiffren durch Merkmale wie den Koinzidenzindex einzuschränken. Das Hauptziel des Ciphertext-Only Angriffs ist es, den zum Geheimtext gehörenden Klartext herauszufinden. Für einen Ciphertext-Only Angriff gibt es verschiedene Herangehensweisen. Welche sinnvoll ist, hängt von den bekannten Informationen und der verwendeten Chiffre ab.

#### 2.3.4.1 Brute-Force Angriff

Ein Brute-Force Angriff ist ein Ciphertext-Only Angriff, der funktioniert, indem jeder mögliche Schlüssel ausprobiert und das Ergebnis ausgewertet wird. Für die Erfolgswahrscheinlichkeit dieser Art von Angriff ist die Schlüsselraumgröße von Bedeutung. Je größer der Schlüsselraum einer Chiffre ist, umso mehr verschiedene Schlüssel muss der Angreifer testen, um den Richtigen zu finden. Entsprechend dauert das Durchsuchen des Schlüsselraums länger. Getestet auf einem Intel i7 4771 Prozessor mit 4 Kernen, jeweils mit Hyperthreading wodurch 8 parallele Threads möglich sind, und getestet an der Implementierung des Brute-Force Angriffs auf eine AES-128 Verschlüsselung in CrypTool2, ergibt sich für einen 38 Bit

langen Schlüssel eine Wartezeit von einem Tag, bis der Schlüssel gefunden wird. Bei einem 47 Bit langen Schlüssel erhöht sich diese Wartezeit auf über ein Jahr. Nimmt man ein acht Zeichen langes, nur aus Kleinbuchstaben bestehendes Passwort, so gibt es

$$\mathcal{K} = 26^8 \approx 2 \cdot 10^{11} \approx 2^{37} \tag{2.1}$$

mögliche Kombinationen, wie das Passwort aussehen kann, was 37 Bit entspricht. Nimmt man nun auch Großbuchstaben hinzu, so ergeben sich

$$\mathcal{K} = 52^8 \approx 5 \cdot 10^{13} \approx 2^{46} \tag{2.2}$$

Möglichkeiten, was einem 512 mal so großen Schlüsselraum entspricht. Je länger und komplexer Passwörter sind, umso unwahrscheinlicher ist es also, dass sie mit Hilfe eines Brute-Force Angriffs gebrochen werden können. Entsprechend wichtig ist, dass eine Verschlüsselung ausreichend viele Möglichkeiten bietet, den Schlüssel zu variieren. Ist die Schlüsselraumgröße einer Chiffre zu klein und damit schnell durchsuchbar, so kann die Verschlüsselung einfach gebrochen werden, indem ein Angreifer mit Hilfe eines Brute-Force Angriffs den kompletten Schlüsselraum durchsucht. Bei der Konzeption neuer Verschlüsselungen ist es entsprechend essentiell, den Schlüsselraum dieser abzuschätzen, da die beste Verschlüsselungsmethode trivial zu brechen ist, wenn der komplette Schlüsselraum durchsuchbar ist.

#### 2.3.4.2 Heuristische Verfahren

Heuristische Verfahren nutzen die Eigenart klassischer Verschlüsselungen, dass ausgehend von einem festgelegten Klartext eine kleine Änderung des Schlüssels auch nur eine kleine Änderung des Geheimtextes zur Folge hat. Im Gegensatz zu modernen Verschlüsselungen, bei denen die Änderung einer Stelle des Schlüssels einen komplett anderen Geheimtext zur Folge hat, kann man Klassische Chiffren so angreifen, indem man versucht, sich dem Schlüssel anzunähern. Dies geschieht durch die wiederholte Veränderung einzelner Stellen des Schlüssels. Ein Heuristisches Verfahren ist der Hill-Climbing Algorithmus. Hierfür muss der Angreifer eine Kostenfunktion entwickeln, die die Qualität eines Textes misst. Diese nutzt die spezifischen Kennzeichen einer Sprache. Daher sollte der Angreifer im besten Fall wissen, in welcher Sprache der Klartext verfasst ist. Diese Kostenfunktion kann zum Beispiel das Vorkommen von drei oder vier aufeinanderfolgenden Buchstaben, Tri- und Tetragramme, nutzen. Der Angreifer wählt nun einen zufälligen Startschlüssel aus. Ausgehend von diesem wird der Schlüssel immer wieder leicht verändert. Nach jeder Veränderung des Schlüssels wird der Geheimtext mit dem neuen Schlüssel dechiffriert. Anschließend wird der so erhaltene Klartext mit der Kostenfunktion analysiert. Diese bewertet, wie stark der Klartext dem Text einer Sprache ähnelt. Je eher der Text der gesuchten Sprache entspricht, umso besser wird er von der Kostenfunktion bewertet. Dies wird wiederholt, bis kein besser bewerteter Schlüssel mehr gefunden werden kann. Dies bedeutet jedoch nicht, dass es keinen besseren Schlüssel gibt. Dies bedeutet nur, dass ein lokales Maximum gefunden wurde. Je nach gewählten Startschlüssel kann dieses lokale Maximum

jedoch weit von dem korrekten Schlüssel entfernt sein. Wie in Abbildung 2.1 zu erkennen ist wird das globale Maximum nur gefunden, wenn der zufällige Startschlüssel im Bereich zwischen 13 und 17 liegt. Bei Startschlüsseln außerhalb dieses Bereichs werden andere, lokale Maxima gefunden, jedoch nicht das globale Maximum. Daher wird der Algorithmus nicht nur einmal gestartet, sondern mehrmals,

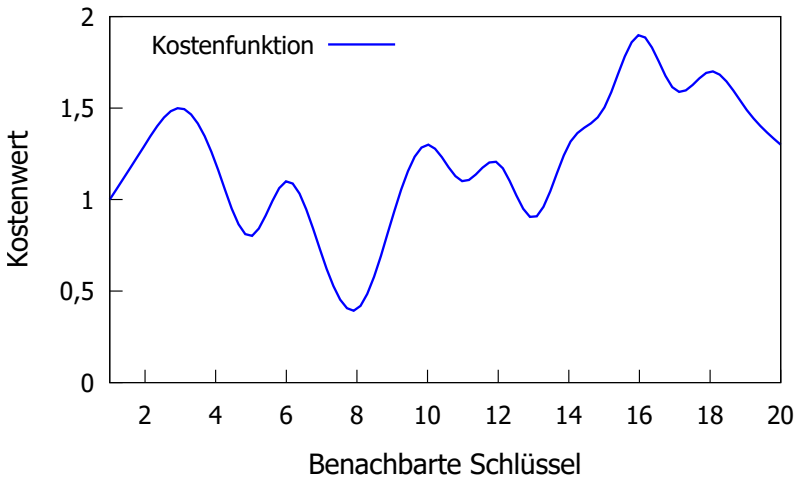



Abbildung 2.1: Beispiel eines Schlüsselraums. Auf der x-Achse befinden sich die Elemente des Schlüsselraums, die y-Achse entspricht der Bewertung durch eine Kostenfunktion.

um möglichst viele verschiedene Startschlüssel zu testen. Das Ziel hierbei ist es, die Startschlüssel möglichst gleichmäßig über den Schlüsselraum zu verteilen, sodass alle Maxima der Kostenfunktion gefunden werden können. Eine mögliche Optimierung heuristischer Verfahren ist das sogenannte „Simulated Annealing“. Dies ermöglicht es, dass mit einer bestimmten Wahrscheinlichkeit auch ein schlechteres Zwischenergebnis erlaubt wird. Das Ziel der Idee ist, dass der Algorithmus in diesem Fall auf der Suche nach dem globalen Maximum lokale Maxima wieder verlassen kann, während das globale Maximum nicht wieder verlassen wird. Dies soll realisiert werden, indem die Wahrscheinlichkeit, dass ein schlechteres Ergebnis akzeptiert wird, mit der Laufzeit des Algorithmus sinkt.

## 2.4 VoluntCloud

Die VoluntCloud ist ein von Studenten, unter anderem dem Autor dieser Arbeit, im Fachgebiet Angewandte Informationssicherheit der Universität Kassel entwickeltes Programm, das die Verteilung von Berechnungen in einem Peer-to-peer Netzwerk ermöglicht. Dieses basiert auf der ebenfalls von einem Studenten entwickelten und auch in CrypTool2 zur verteilten Berechnung genutzten VoluntLib. Jeder Interessent kann sich in dieser einen Account erstellen. Diese werden vom Fachgebiet verwaltet, der Nutzer erhält ein signiertes Zertifikat, das mit seinem


Abbildung 2.2: Login Interface der VoluntCloud.

gewählten Passwort verschlüsselt ist und mit dem er sich anmelden kann. Ebenfalls wird im Fachgebiet eine Liste mit Zertifikaten verwaltet, deren Besitzer als Entwickler freigeschaltet sind. Nur Entwickler sind in der Lage, Jobs in der VoluntCloud hochzuladen. Hat sich der Nutzer eingeloggt, so erhält er eine Übersicht aller in der VoluntCloud angebotenen Jobs. Diese Jobs sind Aufgaben, an denen Freiwillige mitrechnen können. Dabei kann der Nutzer einsehen, wann der Job von welchem Entwickler eingestellt wurde und wie weit die Berechnung des Jobs fortgeschritten ist. Auf der rechten Seite erscheint außerdem eine Übersicht über den gerade angewählten Job, die eine kurze Beschreibung des Jobs beinhalten kann. Aus dieser Liste kann ein Job ausgewählt und geöffnet werden. Weiterhin hat der Ersteller eines Jobs hier die Möglichkeit, den Job wieder zu löschen. Die Joblist wird im Hintergrund regelmäßig aktualisiert, sodass neue Jobs schnell für alle Nutzer sichtbar sind.
Nur für eingetragene Entwickler sichtbar ist die Schaltfläche, über die man zu der Seite gelangt, auf der man einen neuen Job erstellen kann. Über diese gelangt man auf eine Seite, in der man den Namen des Jobs, die zum Job gehörende Datei und eine Beschreibung des Jobs angeben kann. Während der Name und die Beschreibung des Jobs optional sind, muss immer eine zu dem Job gehörende Datei angegeben werden. Da die Identifikation der Jobs über automatisch generierte IDs und nicht über den Namen funktioniert ist ein Job ohne Namen technisch kein Problem, auch wenn dies für den Nutzer verwirrend sein kann. Die Datei muss

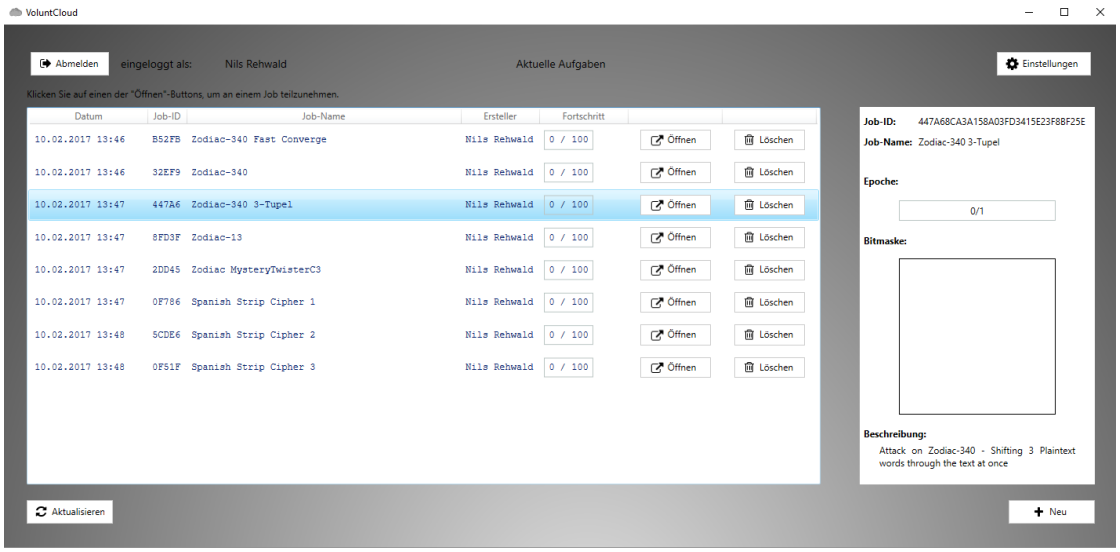



Abbildung 2.3: Jobübersicht der VoluntCloud.

eine Windows Dynamic-link library, kurz dll, sein. Dafür bietet die VoluntCloud eine Schnittstelle an. Diese enthält eine abstrakte Klasse „ACalculationTemplate“, die der Entwickler implementieren muss. Ebenso muss der Entwickler die abstrakte Klasse „AWorker“, die die für die eigentliche Berechnung und Verteilung zuständige VoluntLib anbietet, implementieren. Wurden beide Klassen korrekt implementiert, so kann der Entwickler die resultierende dll Datei in der VoluntCloud hochladen. Dabei wird geprüft, ob die dll auch wirklich beide Klassen enthält und diese instanziierbar sind. Eine Limitierung, die durch die Netzwerkkommunikation der VoluntLib bedingt ist, ist die Größe eines Jobs. Die VoluntLib nutzt zur Netzwerkkommunikation das User Datagram Protocol, kurz UDP. Dies schränkt die Größe eines Jobs auf die Größe eines UDP Pakets, also circa 65 Kilobyte, ein. Daher stellt die VoluntCloud dem Entwickler die Möglichkeit zur Verfügung, Resourcen von einem Webserver nachzuladen. Diese Funktion ermöglicht es dem Entwickler, größere Datein wie Wörterbücher oder externe Bibliotheken auf einem eigenen Server hochzuladen. Zusätzlich muss er eine XML Datei anlegen, in der die Ressourcen mit Namen und URL aufgelistet werden. In seinem Code muss der Entwickler anschließend nur den Link zu der XML Datei angeben. Die VoluntCloud überprüft automatisch ob ein Link zu einer XML Datei angegeben wurde sobald ein Job geöffnet wird. Ist dies der Fall, wird diese geparst und die in der Datei aufgelisteten Dateien werden heruntergeladen. Dadurch kann der Entwickler auch auf größere Ressourcen zugreifen, ohne dass seine dll zu groß wird.
Öffnet ein Nutzer einen Job aus der Liste, so öffnet sich eine Ansicht mit Details zu diesem. In dieser Ansicht hat der Nutzer die Möglichkeit, die Berechnung des Jobs zu starten oder zu stoppen. Startet der Nutzer die Berechnung, so wird ihm von der VoluntLib eine Block ID zugewiesen. Diese entspricht der Nummer des Blocks, der berechnet werden soll. Der Entwickler kann dabei in der dll angeben, auf wieviele Blöcke seine Berechnung aufgeteilt werden soll. Die Methode, in der die Berechnung erfolgt, bekommt mit dem Aufruf die ID des zu berechnenden

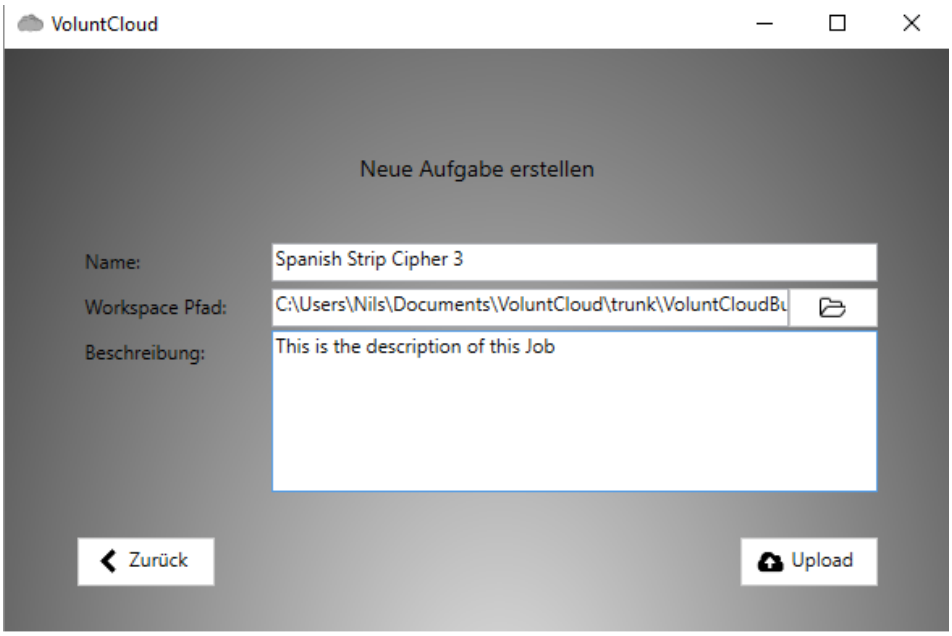



Abbildung 2.4: Interface zum Erstellen eines neuen Jobs.

Blocks übergeben. Der Entwickler muss anhand dieses Parameters definieren, was die Methode berechnen soll. Ist die Berechnung eines Blocks abgeschlossen, so wird diese Information über die VoluntLib an alle anderen Nutzer weitergegeben. Ebenso werden die Ergebnisse der Berechnung verteilt. Dazu muss der Entwickler ebenfalls eine vorgegebene Methode zum Zusammenfassen der Ergebnisse implementieren. Die Berechnung des Jobs erfolgt in zwei Schritten. Der Job wird von der VoluntLib in die vom Entwickler angegebene Anzahl Blöcke unterteilt. Mehrere Blöcke werden zu Epochen zusammengefasst. Startet ein Nutzer die Berechnung, so berechnet er einen zufälliger Block der aktuellen Epoche. Dadurch soll möglichst verhindert werden, dass ein Block von zwei Nutzern berechnet wird. Eine zentrale Verteilung der Blöcke, sodass dies ausgeschlossen ist, ist in einem Peer-to-peer Netzwerk nicht möglich. Sind alle Blöcke einer Epoche berechnet, so wird die nächste Epoche gestartet. Sind alle Epochen eines Jobs fertig berechnet, so kann in der VoluntCloud das Gesamtergebnis der Berechnung angezeigt werden. Dafür steht dem Entwickler eine Fläche in der Detailanzeige eines Jobs zur Verfügung, die er frei mit Informationen füllen kann.

# 3 Homophone Verschlüsselungen

Homophone Verschlüsselungen sind eine besondere Form der monoalphabetischen Substitutionschiffren. Der Unterschied ist, dass bei homophonen Verschlüsselungen das Geheimtextalphabet größer sein kann als das Klartextalphabet. So kann es für jeden Klartextbuchstaben mehrere Geheimtextbuchstaben geben, auf die dieser abgebildet werden kann. Die Geheimtextbuchstaben heißen in diesem Fall „Homophone“. Dies erschwert die statistische Analyse verschlüsselter Texte. Während es bei einem mit monoalphabetischer Substitution verschlüsselten Text gut realisierbar ist, durch Zählen der Häufigkeit von Geheimtextbuchstaben zu erkennen, welchen Klartextbuchstaben diese entsprechen, wird diese statistische Verteilung bei homophonen Chiffren zerstört, da ein Klartextbuchstabe nicht mehr genau einem, sondern mehreren Geheimtextbuchstaben zugeordnet werden kann.

## 3.1 Funktionsweise

Das Ver- und Entschlüsseln von Texten gestaltet sich ähnlich wie bei anderen Substitutionschiffren. Durch Ersetzen der Buchstaben nach bestimmten Regeln, die dem Schlüssel entsprechen, erhält man den entsprechenden Klar- oder Geheimtext. Der Vorteil homophoner Chiffren ist, dass sie genauso einfach anzuwenden sind wie normale Substitutionschiffren, jedoch deutlich schwerer zu brechen sind als andere monoalphabetische Substitutionschiffren.

### 3.1.1 Verschlüsseln

Zum Verschlüsseln schreibt sich der Benutzer das Klartextalphabet und das dazugehörende Geheimtextalphabet auf. Für jeden Klartextbuchstaben wählt er jetzt eine beliebige Anzahl Geheimtextbuchstaben aus, die diesem entsprechen. Besteht das Klartextalphabet aus den Lateinischen Buchstaben und das Geheimtextalphabet aus Zahlen von 01 bis 99, so kann ein Mapping von Klartext und Geheimtext wie folgt aussehen: Aus der Tabelle 3.1 folgt, dass dem Klartextbuchstaben „F“ die Homophone „50“, „52“ und „03“ zugeordnet sind. Zum Verschlüsseln eines Textes kann nun diese Tabelle als Hilfe verwendet werden. Es gibt zwei Möglichkeiten, welcher Geheimtextbuchstabe für einen Klartextbuchstaben genutzt wird. Die einfachere ist, immer zufällig einen passenden Geheimtextbuchstaben aus der Tabelle zu wählen. Die etwas aufwändigere Methode ist, die Geheimtextbuchstaben

| **Klartext** | A | B | C | D | E | F | G | ... |
|---|---|---|---|---|---|---|---|---|
| **Homophone** | 19 | 93 | 17 | 33 | 42 | 50 | 02 | ... |
| | 77 | 09 | 11 | 88 | 31 | 52 | 05 | ... |
| | 19 | | 89 | | 70 | 03 | 97 | ... |

Tabelle 3.1: Mögliche Darstellung eines Schlüssels

abwechselnd zu nutzen. Bei dem ersten Vorkommen eines Buchstabens im Klartext wird der erste zu diesem Buchstaben passende Geheimtextbuchstabe gewählt. Kommt der Buchstabe danach erneut vor, wird der zweite Geheimtextbuchstabe gewählt. Der Vorteil dieser Methode ist, dass ausgeschlossen wird, dass durch Zufall manche Geheimtextbuchstaben gar nicht genutzt werden. Der Nachteil dieser Methode besteht darin, dass beim Verschlüsseln jeweils darauf geachtet werden muss, welcher Geheimtextbuchstabe gerade für welchen Klartextbuchstaben verwendet werden muss. Außerdem ermöglicht dieses Vorgehen statistische Analysen, welche Homophone zu demselben Buchstaben gehören, die in der Arbeit von Luis Alberto Sanguino [LABS16] näher beschrieben werden.

### 3.1.2 Entschlüsseln

Das Entschlüsseln eines Textes ist sehr einfach. Dazu erstellt man sich eine Tabelle wie 3.2, in der zu jedem Homophon der dazugehörige Klartextbuchstabe aufgelistet ist. Für jeden Geheimtextbuchstaben muss nun einfach der entsprechende Klartextbuchstabe eingesetzt werden. Im Vergleich zum Verschlüsseln ist

| **Homophon** | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Klartext** | F | R | Y | Q | F | A | R | P | J | F | Y | A | E | ... |

Tabelle 3.2: Alternative Darstellung eines Schlüssels

das Entschlüsseln auch dann trivial, wenn die zweite Methode, bei der die Homophone abwechselnd verwendet werden, genutzt wurde. Die Abbildung eines Klartextbuchstabens auf einen Geheimtextbuchstaben ist zwar nicht eindeutig, die Gegenrichtung jedoch schon. Zu jedem Geheimtextbuchstaben gibt es jeweils genau einen dazugehörigen Klartextbuchstaben.

## 3.2 Kryptoanalyse

Auch homophone Verschlüsselungen können kryptologisch analysiert werden. Während Metriken wie der Koinzidenzindex nicht sehr hilfreich sind, da eine homophone Chiffre von Natur aus monoalphabetisch ist und Frequenzanalysen durch die Abbildung eines Klartextbuchstabens auf mehrere Geheimtextbuchstaben zerstört

werden existieren einige Metriken, die hilfreich zur Analyse der Chiffren sind. Dazu gehören die Schlüsselraumgröße und die Unizitätslänge um zum Beispiel abzuschätzen, ob ein Brute-Force Angriff erfolgversprechend ist.

### 3.2.1 Schlüsselraum

Die Schlüsselraumgröße einer Chiffre gibt an, wie viele mögliche valide Schlüssel für diese existieren. Je mehr Schlüssel existieren, umso schwieriger wird es für einen Angreifer, mit Hilfe eines Brute-Force Angriffs alle Schlüssel zu testen. Die Berechnung des Schlüsselraums für homophone Chiffren kann unter zwei Annahmen erfolgen:

- Es wird davon ausgegangen, dass beim Erstellen der Chiffre die Homophone feststanden und anschließend jedem Homophon ein Klartextbuchstabe zugewiesen wurde. In diesem Fall kann es passieren, dass ein Klartextbuchstabe von der Chiffre nicht abgebildet werden kann, da er keinem Homophon zugewiesen wurde.
- Es wird davon ausgegangen, dass beim Erstellen der Chiffre das verwendete Alphabet feststand. Anschließend wurden jedem Klartextbuchstaben eine Menge von Homophonen zugewiesen. In diesem Fall existiert für jeden Klartextbuchstaben mindestens ein Homophon, jeder Klartextbuchstabe kann also garantiert abgebildet werden.

Nach einigen Überlegungen wurde für diese Arbeit angenommen, dass nur der zweite Fall zu berücksichtigen ist. Der erste Fall ist zwar möglich, allerdings verändert sich dadurch das zu Grunde liegende Klartextalphabet. Wird zum Beispiel ein deutscher Text verschlüsselt, in dem die Buchstaben „X“ und „Z“ nicht vorkommen, so muss man diese Klartextbuchstaben bei dem Erstellen des Schlüssels nicht zwingend berücksichtigen. Allerdings muss jeder Buchstabe, der in dem Text vorkommt, mindestens einem Homophon zugeordnet sein. Daher ändert sich durch die Nichtberücksichtigung von nicht vorkommenden Buchstaben lediglich das Klartextalphabet, da dieses kleiner wird. Da jedoch jeder vorkommende Buchstabe zugeordnet sein muss, kann man für eine Abschätzung der oberen Grenze des Schlüsselraums den zweiten Fall der Berechnung heranziehen. Dadurch ergibt sich folgende Schlüsselraumgröße:

$$\mathcal{K} = \binom{n}{i} \cdot i! \cdot i^{n-i} \tag{3.1}$$

Hierbei steht $n$ für die Anzahl der Geheimtextbuchstaben und $i$ für die Anzahl der Klartextbuchstaben. Die Einzelteile der Formel lassen sich wie folgt erklären:

- Die Anzahl der Möglichkeiten, $i$ von $n$ Homophonen auszuwählen. Dies sind die Homophone, von dem jedes einem Buchstaben zugeordnet wird, damit jeder Buchstabe mindestens ein zugeordnetes Homophon hat.

- Die Anzahl der Möglichkeiten, die $i$ ausgewählten Homophone auf die Buchstaben zu verteilen, dass jedem Buchstaben genau ein Homophon zugeordnet wird.
- Die Anzahl der Möglichkeiten, die verbleibenden $n - i$ Homophone zufällig auf $i$ Klartextbuchstaben zu verteilen.

An der Gleichung wird sichtbar, dass der Schlüsselraum mit einer steigenden Zahl Geheimtextbuchstaben exponentiell wächst. Dies lässt vermuten, dass eine homophone Verschlüsselung mit einer ausreichend großen Menge Homophoner nicht durch einen Brute-Force Angriff zu brechen ist. Den Schlüsselraum kann man durch einen Vergleich mit dem Triple Data Encryption Algorithm, einem vom National Institute of Standards and Technology als sicher eingestufen Algorithmus, vergleichen. [WCB12] Dieser hat eine Schlüsselraumgröße von circa $2^{168}$ im Fall eines Ciphertext-Only Angriffs. Ausgehend von einem Klartextalphabet mit 26 Zeichen kann man abschätzen, wie viele Homophone verwendet werden müssen, um eine ähnliche Schlüsselraumgröße zu erhalten. Nach der gewählten Berechnungsmethode ergibt sich folgende Formel:

$$2^{168} = \binom{n}{26} \cdot 26! \cdot 26^{n-26} \tag{3.2}$$

$$\Rightarrow n \approx 37 \tag{3.3}$$

Durch die Verwendung von 26 Klartext- und 37 Geheimtextbuchstaben lässt sich also bereits eine Schlüsselraumgröße in der Dimension des Triple-DES schaffen.

### 3.2.2 Unizitätslänge

Die Unizitätslänge einer Verschlüsselung gibt an, wie lang ein Geheimtext mindestens sein muss, sodass genau ein Schlüssel existiert, der diesen Geheimtext auf einen sinnvollen Klartext abbildet. Versucht man einen Geheimtext zu entschlüsseln, der kürzer als die Unizitätslänge der verwendeten Verschlüsselung ist, so kann es passieren, dass mehrere Schlüssel gefunden werden, die den Text auf sinnvolle, aber verschiedene Klartexte abbilden. Ohne den Schlüssel zu kennnen ist es in diesem Fall nicht möglich, sicher festzustellen, welches der dem Geheimtext entsprechende Klartext ist. Die Berechnung der Unizitätslänge einer Chiffre ist sinnvoll um abzuschätzen, ob ein Ciphertext-Only Angriff auf eine abgefangene Nachricht sinnvoll ist oder nicht. Die Unizitätslänge $U$ berechnet sich aus folgenden Parametern:

- H(k) - Der Entropie des Schlüsselraums. Dies ist der Logarithmus der Schlüsselraumgröße zur Basis 2.
- D - Die Redundanz der Klartextsprache, die den Informationsgehalt einer Sprache beschreibt. Dieser Wert ist für Englisch $3,2$.

Die allgemeine Formel zur Berechnung der Unizitätslänge lautet:

$$U = \frac{H(k)}{D} \tag{3.4}$$

Ausgehend von einem Klartextalphabet mit 26 Buchstaben ergibt sich für eine Chiffre mit 30 Homophonen eine Unizitätslänge von etwa 38 Buchstaben, für eine Chiffre mit 50 Homophonen beträgt die Unizitätslänge etwa 77 Buchstaben. Die Unizitätslänge steigt dabei linear mit der Entropie des Schlüsselraums, wie aus (3.4) ersichtlich ist.

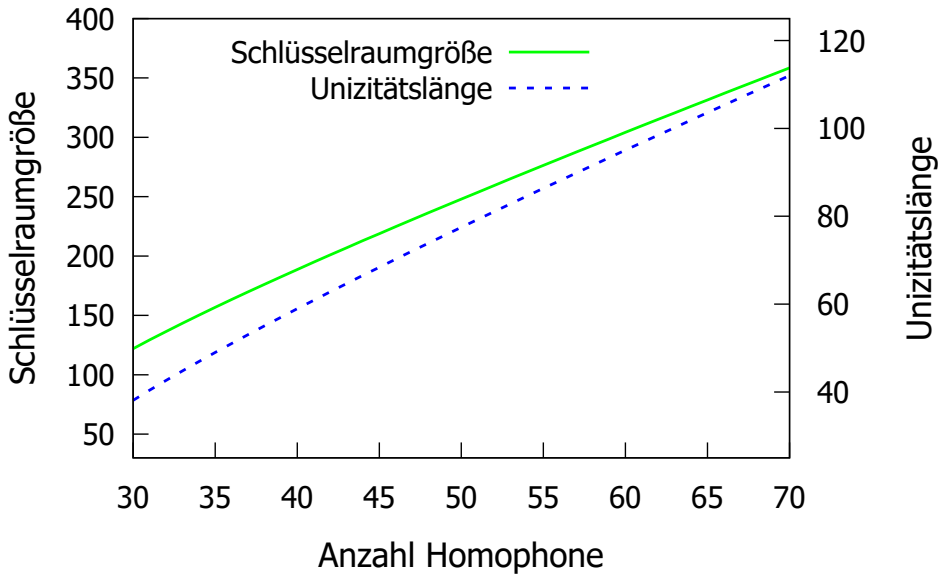



Abbildung 3.1: Die Entwicklung von Entropie und Unizitätslänge in Abhängigkeit der Anzahl der genutzten Homophone bei 26 Klartextbuchstaben.

### 3.2.3 Manuelle Angriffe

Eine Möglichkeit, homophone Chiffren mit Stift und Papier anzugreifen, ist zu versuchen, Wörter in dem Geheimtext zu finden. Ist ein Wort gefunden, so kann man die weiteren Vorkommen der somit festgelegten Homophone betrachten und versuchen, weitere Wörter zu finden. Dadurch wird der Geheimtext Stück für Stück entschlüsselt. Sobald erkannt wird, dass ein Wort nicht passt, da dadurch Homophone festgelegt werden, durch die an anderen Stellen des Textes sinnloser Klartext entsteht, muss das komplette Wort und alle durch dieses Wort festgelegte Homophone zurückgesetzt werden. Dies ist vergleichbar mit dem Lösen eines Kreuzworträtsels. Sobald man erkennt, dass eine Lösungsannahme die korrekte Lösung einer anderen Frage verhindert, muss dieses Wort gelöscht und alle auf diese Annahme aufbauenden Lösungen überdacht werden. Für diesen Angriff ist es nötig, entweder Glück

bei der Wahl des ersten Wortes zu haben. Je mehr über den Verfasser des Geheimtextes und die Umstände der Entstehung bekannt ist, umso wahrscheinlicher ist es, erahnen zu können, welche Worte in dem Text vorkommen. Diese Abschätzung und auch das Erkennen der Worte im Text erfordern menschliche Fähigkeiten, wodurch dieser Angriff nicht einfach auf Computersysteme übertragbar ist. Eine andere Möglichkeit ist die Suche nach Wiederholungen von Sequenzen innerhalb des Geheimtextes. Diese können darauf hindeuten, dass an diesen Stellen dasselbe Wort steht und einige Teile dieses Wortes gleich verschlüsselt wurden. Auch die Suche nach Buchstabenpaaren wie „ch“ oder „st“ ist weiterhin möglich. Zwar ist die Wahrscheinlichkeit, dass diese an zwei verschiedenen Stellen gleich abgebildet werden, durch die Verwendung verschiedene Homophone gesenkt, jedoch immer noch vorhanden.

## 3.3 Historische Nutzung

Homophone Chiffren waren einer der ersten Versuche, die statistische Analyse von Geheimtexten zu erschweren. Eine der bekanntesten historischen Anwendungen homophoner Verschlüsselungen ist die „Beale-Chiffre“ aus dem 19. Jahrhundert. Diese besteht aus drei Teilen, von denen bis heute nur einer gebrochen werden konnte. Auch wenn die anderen beiden Teile wahrscheinlich eine Fälschung sind [Sch12], da auch die Geschichte zu der Chiffre nicht schlüssig ist, konnte der zweite Teil der Chiffre als homophone Verschlüsselung gelöst werden. Dabei verwendete der Autor die Unabhängigkeitserklärung der USA als Schlüssel und suchte für jeden Klartextbuchstaben ein Wort mit dem selben Anfangsbuchstaben in der Unabhängigkeitserklärung. Anschließend ersetzte er den Buchstaben durch die Stelle des Wortes in der Unabhängigkeitserklärung. Da viele Worte in dieser mit dem selben Buchstaben anfangen entspricht dies einer homophonen Chiffre. Auch in der Moderne wurden Homophone Chiffren weiterhin betrachtet. Neben den beiden in dieser Arbeit vorgestellten Verwendungen gab es auch die Idee, homophone Substitutionen zu verwenden, um bestehende Verschlüsselungsverfahren sicherer zu machen. So brachte Christoph Günther die Idee vor, homophone Substitutionen vor einer Verschlüsselung mit dem damaligen Standard DES durchzuführen, um die statistische Analyse von DES zu erschweren. [Gün88]

### 3.3.1 Der Zodiac Killer

Der Zodiac Killer war ein Serienmörder, der in den 1960er und 1970er Jahren im Amerikanischen Bundesstaat Kalifornien aktiv war und in dieser Zeit mindestens fünf Menschen tötete. Bis heute konnte er weder gefasst noch identifiziert werden. Das erste dem Zodiac zuzuordnende Verbrechen ereignete sich am 20. Dezember 1968 im Bezirk Vallejo, etwa 50km nördlich von San Francisco. Dabei erschoss er zwei Teenager in ihrem Auto. Das zweite dem Zodiac Killer zuzuordnende Verbrechen ereignete sich am 5. Juli 1969. Dabei tötete er eine Person, eine

andere wurde schwer verwundet. Am 31. Juli des selben Jahres gingen in den Redaktionen von drei regionalen Tageszeitungen Briefe ein, die vom Zodiac Killer stammten. Dies wurde von der Polizei als plausibel befunden, da die Briefe Details über die Morde enthielten, die der Öffentlichkeit nicht bekannt waren. Jeder der Briefe enthielt, neben Details über die begangenen Verbrechen, einen jeweils 136 Zeichen langen verschlüsselten Text. Diese drei Textstücke ergeben zusammengesetzt die erste verschlüsselte Nachricht des Killers, die aufgrund ihrer Länge auch als Zodiac-408 bekannt ist. In den Briefen wies er die Zeitungen dazu an, seine Nachrichten am nächsten Tag auf der Titelseite abzudrucken. Ansonsten würde er eine Mordserie beginnen. Während die Polizei die Zeitungen darum bat, die unverschlüsselten Texte nicht abzudrucken, um Details zu den Verbrechen geheim zu halten, druckten alle drei Zeitungen die ihnen zugesandten verschlüsselten Texte ab. Weder die Spezialisten der Polizei noch andere Institutionen wie die NSA schafften es, Zodiac-408 zu entschlüsseln. Im Gegensatz dazu gelang dies dem Ehepaar Hardens. Am 9. August wurde ihre Lösung im San Francisco Chronicles abgedruckt. Beide Ehepartner hatten vorher keine Brührungspunkte mit Kryptographie, lösten jedoch gerne Kreuzworträtsel und kamen auf die Lösung, da sie den Beginn der Nachricht richtig vermuteten und durch Ausprobieren schließlich herausfanden, dass die Nachricht mit den Worten „I like killing“ begann. Daraus ergaben sich weitere Teile des Textes, wodurch sie diesen Stück für Stück lösen konnten. Die entschlüsselte Nachricht enthält zwar einige Rechtschreibfehler wie „Experence“ statt „Experience“ oder „Paradice“ statt „Paradise“, jedoch ist der Inhalt der Übersetzung schlüssig, weshalb die Entschlüsselung als korrekt angenommen wird. Auch erwähnte der Zodiac in seinen folgenden Briefen nie, dass seine erste Chiffre falsch entschlüsselt wurde. Ein Rätsel blieben jedoch die letzten 18 Zeichen des Textes. Entschlüsselt man diese mit dem gleichen Schlüssel wie den Reste der Nachricht, so erhält man den Klartext

EBEORIETEMETHHIPITI

der nicht sinnvoll erscheint. Über die letzten 18 Zeichen von Zodiac-408 gibt es mehrere Theorien. So könnte der Text ein Anagramm des Namens des Mörders enthalten. Immerhin behauptete dieser in dem unverschlüsselten Teil des Briefes, dass seine Identität in dem verschlüsselten Text enthalten sei. Eine Lösung, die sogar die Polizei dazu brachte, Personen mit diesem Namen zu überprüfen, war „Robert Emmet the Hippie“. Allzu schlüssig ist diese Lösung jedoch nicht, da, um auf dieses Anagramm zu kommen, zwei Buchstaben hinzugefügt werden müssen. Eine Möglichkeit, warum viele Menschen damals zu dieser Lösung kamen, könnte in San Francisco liegen. Dort gibt es seit dem Jahr 1919 eine Statue von Robert Emmet, einem Irischen Rebellen aus dem 18. Jahrhundert. [emm] Eine andere Möglichkeit ist, dass der Absender am Ende zufällige Homophone hinzufügte, um die dritte Nachricht ebenfalls auf 136 Zeichen zu bringen. In diesem Fall wäre es nicht möglich, einen sinnvollen Inhalt in den letzten 18 Zeichen zu finden. Die Suche nach einem Namen in diesen 18 Buchstaben ist nicht erfolgversprechend, da es über $10^{13}$ mögliche Anordnungen für diese gibt. Einen Tag, nachdem die Lösung

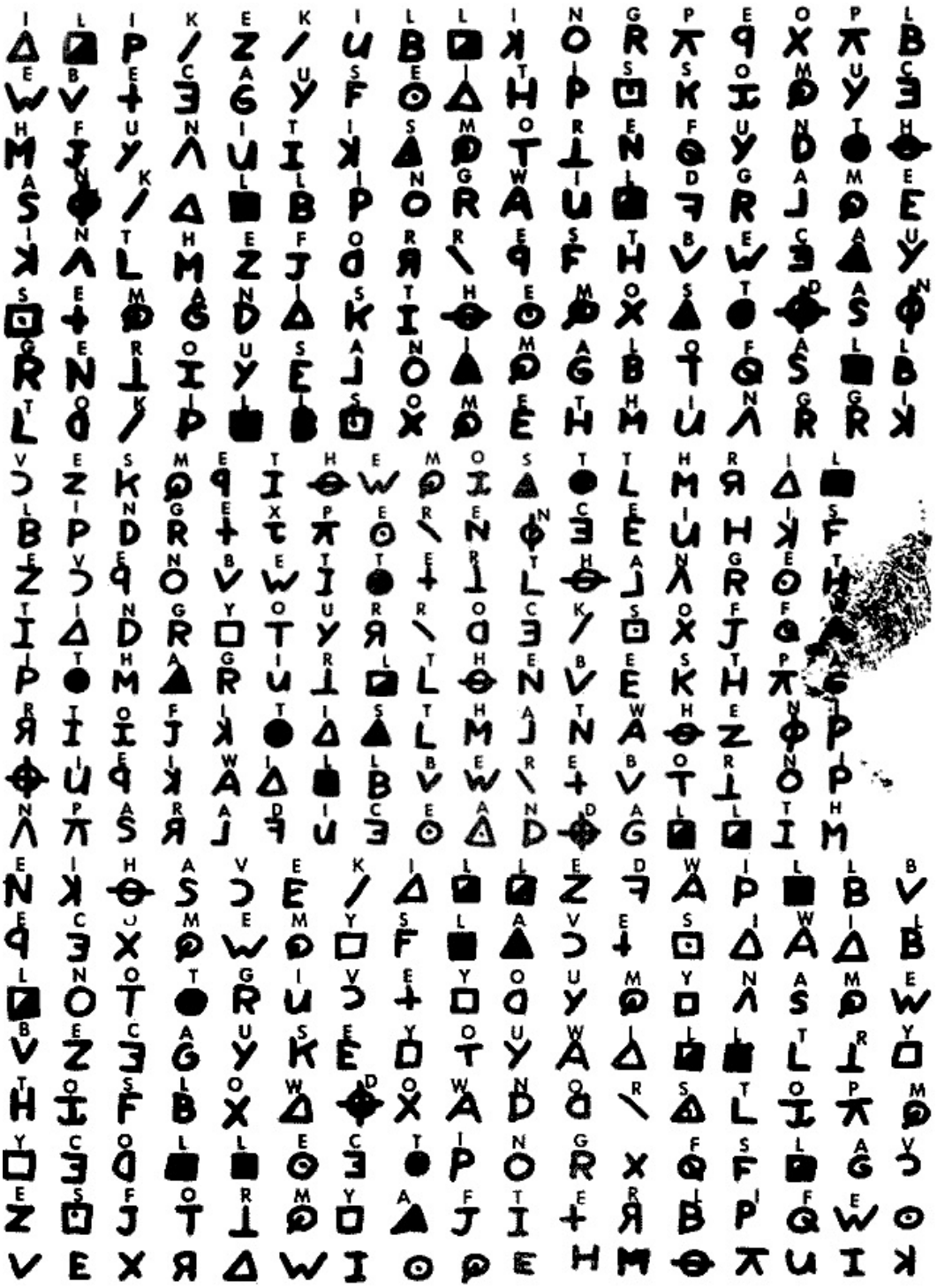


Abbildung 3.2: Die drei Teile von Zodiac-408 zusammengefügt mit entschlüsseltem Text über den Homophonen. Entnommen von [Ora]

der Hardens veröffentlicht wurde, erreichte ein Brief die Polizei. In dieser befand sich ein Papier, auf dem den Klartextbuchstaben die dazugehörigen Homophone zugeordnet waren. Der Brief war unterschrieben mit „concerned citizen“ was dieser Lösung den Namen „Concerned Citizen Key“ gab. Der Absender der Lösung wollte anonym bleiben. Der Schlüssel wurde anschließend vom FBI geprüft und als weitestgehend korrekt eingestuft. Während sich die Lösung des Unbekannten in weiten Teilen mit der der Hardens deckte, so gab es doch einige Ungereimtheiten. [cit] So enthält dieser Schlüssel zum Beispiel für einige Homophone mehrere Klartextbuchstaben zur Auswahl. Dies deutet darauf hin, dass die Nachricht von einer Person verfasst wurde, die das Kryptogramm eigenständig löste und keine Kopie der Lösung der Hardens ist. Da der Zodiac einen ihm zugeordneten Brief im Jahr 1974 ebenfalls mit „a citizen“ unterschrieb existiert die Theorie, dass der Citizen Key vom Zodiac selbst verfasst wurde. Ob dieser absichtlich Fehler enthält, diese entstanden, weil der Verfasser einige ähnlich aussehende Homophone verwechselte oder das Abbilden eines Homophons aus mehrere Klartextbuchstaben auf eine polyalphabetische Verschlüsselung, die der Killer eventuell in seinem zweiten Kryptogramm nutzte, hinweisen, lässt sich nicht klären. Erschwerend kommt hinzu, dass der Citizen Key erst 2011 durch eine Anfrage im Rahmen des „Freedom of Information“ Gesetzes öffentlich wurde. Allerdings wurde nicht der komplette Brief veröffentlicht. Zwei Seiten wurden durch das FBI zensiert. Was diese beiden Seiten enthalten, ist entsprechend nicht bekannt.
Am 11. Oktober 1969 verübte der Zodiac das letzte eindeutig ihm zuzuordnende Verbrechen. Er erschoss dabei einen Taxifahrer. Fast wäre er an diesem Tag gefasst worden. Die von Zeugen gerufene Polizei befragte ihn, da er sich in der Nähe des Tatorts aufhielt. Da die Polizisten jedoch die Information hatten, dass sie nach einem Täter mit schwarzer Hautfarbe suchen müssten, wurden seine Personalien nicht aufgenommen. [Sch12] Entsprechend hämisch äußerte der Killer sich in einem späteren Brief über die Leistung der Polizei an diesem Tag. Zodiac-408 war nicht die letzte verschlüsselte Botschaft des Killers. Es folgten im Lauf der Zeit drei weitere Kryptogramme. Am 9. November 1969 erreichte ein Brief den San Francisco Chronicles. Dieser enthielt ein 340 Zeichen langes Kryptogramm, auch als Zodiac-340 bekannt. Mit dem für Zodiac-408 verwendeten Schlüssel lies sich dieses jedoch nicht entschlüsseln. Bis heute wurde keine anerkannte Lösung für Zodiac-340 gefunden. Die dritte verschlüsselte Nachricht, Zodiac-13, ging am 20. April 1970 bei der selben Zeitung ein. Nur wenige Monate später, am 16. Juni 1970, erreichte den „San Francisco Chronicles“ die bis heute letzte verschlüsselte Nachricht des Zodiac Killers, Zodiac-32. Diese Nachricht bildet das Ende eines Briefes, in dem der Killer angibt, eine Bombe versteckt zu haben. Bis heute sind auch diese beiden Nachrichten nicht entschlüsselt worden. Aufgrund der sehr kurzen Länge beider Nachrichten ist nicht sicher, ob sie als einzelne Kryptogramme überhaupt eindeutig entschlüsselt werden können. Im März 1971 versendete der Zodiac seinen vorerst letzten Brief. Anschließend erreichte den San Francisco Chronicles erst am 29. Januar 1974 wieder ein Brief, der dem Zodiac zugeordnet wird. In diesem Brief bekannte er sich selbst dazu, 37 Menschen getötet zu haben. Eindeutig dem Zodiac zugeordnet werden können jedoch lediglich fünf Morde und zwei versuchte

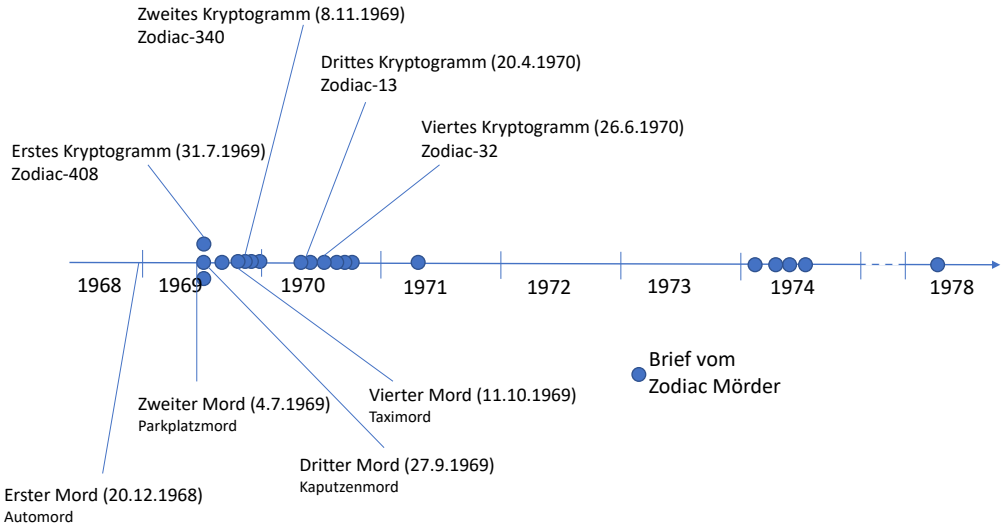



Abbildung 3.3: Die Aktivitäten des Zodiac im Überblick. Idee übernommen aus [Sch12]

Morde. Es ist jedoch möglich, dass der Zodiac noch andere Verbrechen begangen hat, die ihm nicht eindeutig zugeordnet werden können. Der letzte dem Zodiac zugeordnete Mord ereignete sich am 11. Oktober 1969. Bis zum 24. April 1978 folgten noch vier weitere Briefe, die dem Zodiac zugeordnet werden. Allerdings ist diese Zuordnung durch Analysen der Handschrift zumindest teilweise umstritten. Nach dem mit Zodiac-32 abschließenden Brief am 16. Juni 1970 enthielt kein Brief des Zodiac mehr ein Kryptogramm.
Nachdem der Fall zwischenzeitlich geschlossen wurde, so reaktivierte die Polizei von San Francisco im Jahr 2007 den Fall. [zka] Hinweise, die aus der Bevölkerung erhalten werden, werden weiterhin geprüft. Das Lösen von Zodiac-340 könnte hilfreiche Details über den Killer ans Tageslicht bringen.

#### 3.3.1.1 Zodiac-340

Die zweite Verschlüsselte Nachricht ging am 8. November 1969 beim San Francisco Chronicles ein. Sie ist 340 Zeichen lang und besteht aus 63 verschiedenen Homophonen. Somit gibt es einen Ansatz, Schlüsselraumgröße und Unizitätslänge zu berechnen. Um zu prüfen, ob die Nachricht auch im schlechtesten Fall länger als die Unizitätslänge ist, wird eine plausible obere Schranke dieser gesucht. Aus der Formel für die Unizitätslänge ergibt sich, dass diese linear mit der Entropie des Schlüsselraums steigt. Entsprechend werden wir für die Berechnung des

Schlüsselraums Annahmen treffen, die diesen möglichst groß werden lassen. Die Anzahl der Homophone ist bekannt, unbekannt ist lediglich die Anzahl der Klartextbuchstaben. Aus der Formel für die Entropie des Schlüsselraums folgt, dass dieser zuerst mit der Anzahl der Klartextbuchstaben ansteigt, ab einem gewissen Punkt aufgrund der kleiner werdenden Potenz aber wieder sinkt. Der Hochpunkt der Funktion liegt in etwa bei einer Klartextlänge von 40 Zeichen. Für die Berechnung nehmen wir also an, dass das zugrundeliegende Klartextalphabet alle lateinischen Buchstaben sowie die Zahlen von 0 bis 9 umfasst. Außerdem nehmen wir die Satzzeichen „!“, „?“, „.“ und „,“ hinzu. Daraus resultiert ein sinnvolles, 40 Zeichen umfassendes Klartextalphabet. Zu beachten ist, dass durch die Vergrößerung des Klartextalphabets sich auch die Entropie der Sprache $D$ ändert. In diesem Fall nehmen wir nur Zahlen und Satzzeichen zum Klartextalphabet hinzu, die in der Regel verglichen mit Buchstaben selten in Texten vorkommen. Daher sollte sich diese nicht so stark verändern, dass das Ergebnis dadurch unbrauchbar wird. Zur Abschätzung addieren wir deshalb 5% des errechneten Wertes zu der Unizitätslänge hinzu. Weiterhin ist zu bedenken, dass die Hinzunahme von Groß und Kleinbuchstaben zum Klartextalphabet keine Veränderung bewirkt. Zwar vergrößert sich dadurch das Klartextalphabet, jedoch können Homophone beim Angriff auf eine Chiffre zusammengefasst werden. Wurden beim Verschlüsseln zum Beispiel die Homophone „07“ und „77“ für den Buchstaben „A“ und die Homophone „19“ und „42“ für den Buchstaben „a“ verwendet, so kann man dies bei einem Angriff auf die Chiffre zusammenfassen und auf den Buchstaben „A“ abbilden, während Kleinbuchstaben ignoriert werden. Somit ergibt sich folgende Berechnung der Entropie:

$$\binom{63}{40} \cdot 40! \cdot 40^{63-40} \approx 5,4 \cdot 10^{101} \approx 2^{337} \tag{3.5}$$

Daraus ergibt sich folgende Unizitätslänge:

$$U \approx \frac{337}{3,2} \approx 105 \rightarrow 105 \cdot 1.05 \approx 110 \tag{3.6}$$

Dies zeigt, dass unter der Annahme, dass der Killer erneut eine einfache homophone Verschlüsselung verwendete, die Länge der Nachricht deutlich über der Unizitätslänge der genutzten Verschlüsselung liegt. Die meisten Symbole, die als Homophone genutzt wurden, kamen bereits in Zodiac-408 vor. Nur wenige Symbole aus Zodiac-408 kommen nicht mehr vor, dafür enthält Zodiac-340 einige neue Symbole. Trotzdem konnte Zodiac-340 bis heute von niemandem sinnvoll gelöst werden. Daher muss auch die Möglichkeit in Betracht gezogen werden, dass es sich nicht um eine einfache homophone Verschlüsselung handelt. Die Chiffre besteht aus 20 Zeilen mit jeweils 17 Homophonen pro Zeile. Teilt man den Text horizontal in der Mitte, so erhält man zwei Chiffren mit jeweils 10 Zeilen. Auffällig an diesen ist, dass in den ersten drei Zeilen beider Teile kein Homophon doppelt vorkommt. [Ora15] Dies könnte darauf hindeuten, dass der Text in zwei Teilen mit verschiedenen Schlüsseln verschlüsselt wurde. Eine ähnliche Möglichkeit ist, dass der Killer eine polyalphabetische Verschlüsselung nutzte. So könnten verschiedene Schlüssel für

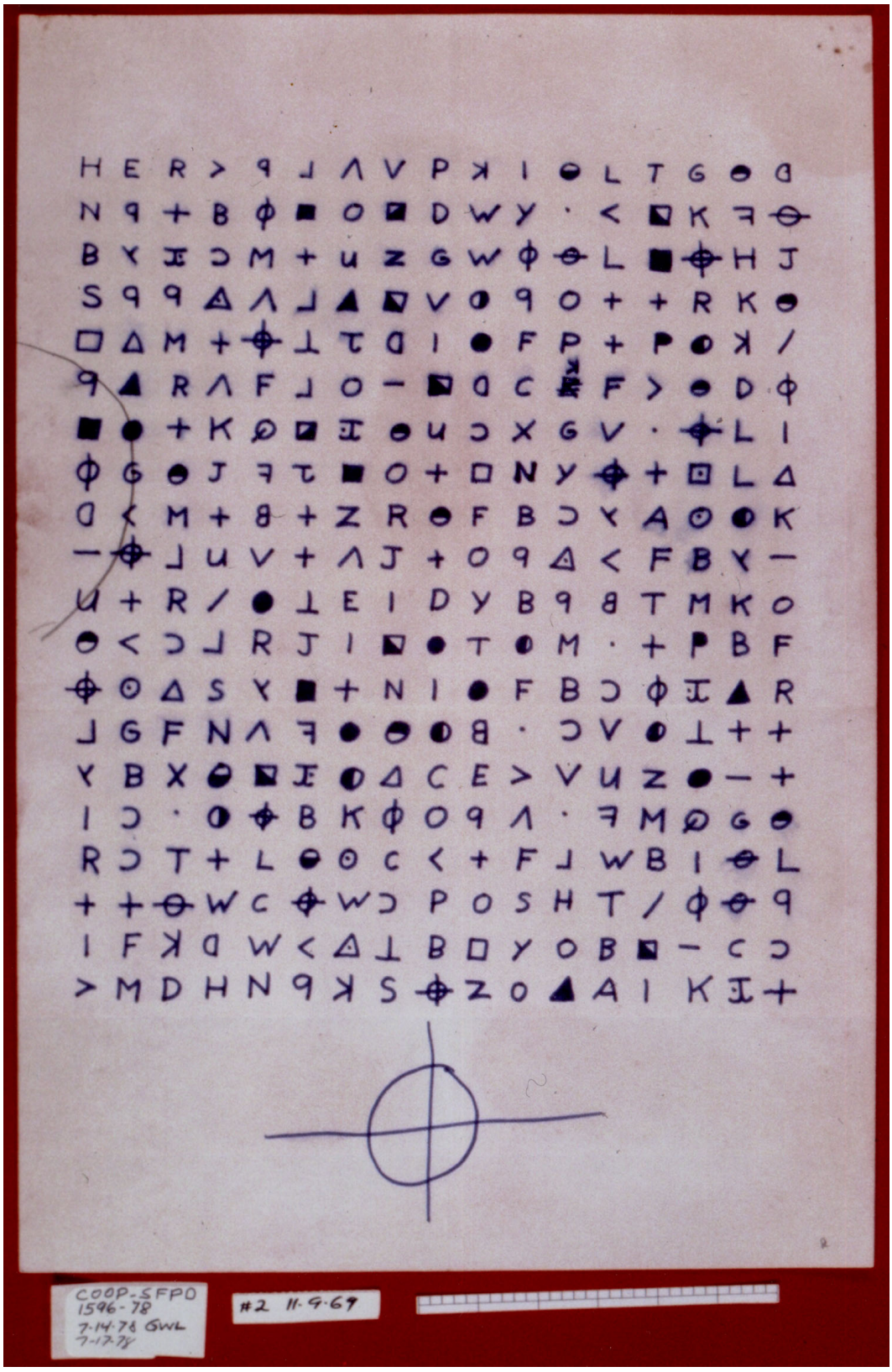



Abbildung 3.4: Die Zodiac-340 Chiffre. Entnommen von [Ora]

Buchstaben an geraden und ungeraden Stellen genutzt worden sein. Auch die Verwendung von mehr verschiedenen Schlüsseln ist denkbar. Ebenso könnte nach der Verschlüsselung durch Homophone zusätzlich eine Spaltentransposition angewandt worden sein. Weiterhin steht nicht fest, dass der Text zeilenweise von links nach rechts gelesen werden muss. Ebenso könnte der Text rückwärts oder spaltenweise von oben nach unten gelesen werden müssen. Bei allen Möglichkeiten ist jedoch zu beachten, dass die Verschlüsselung nicht zu kompliziert gewesen sein sollte. Es ist anzunehmen, dass der Killer die Nachricht von Hand verschlüsselt hat. Daher wäre eine zu umständliche Verschlüsselungsmethode unpraktisch gewesen. Weiterhin kann man aufgrund des psychologischen Profils des Killers, der aufgrund der vielen Briefe, die er versendete, scheinbar nach Aufmerksamkeit strebte, davon ausgehen, dass die Nachricht einen sinnvollen Text enthält und entschlüsselt werden kann. Jedoch könnte der Zodiac, nachdem seine erste Nachricht innerhalb weniger Tage entschlüsselt wurde, versucht haben, die zweite Nachricht komplizierter zu entschlüsseln zu machen. Deshalb ist nicht auszuschließen, dass er eine andere Form der Verschlüsselung als in der ersten Nachricht nutzte.

#### 3.3.1.2 Zodiac-13

Die dritte Nachricht ging am 20. April 1970 beim San Francisco Chronicles ein. Der 13 Zeichen lange, verschlüsselte Text, ist Teil einer längeren Nachricht. In dieser fragt der Zodiac unter anderem, ob sein letztes Rätsel bereits gelöst wurde und gibt an, bis zu diesem Datum zehn Menschen getötet zu haben. Der verschlüsselte Teil befindet sich direkt unter einer Zeile mit dem Inhalt „My name is -“. Der 13 Zeichen

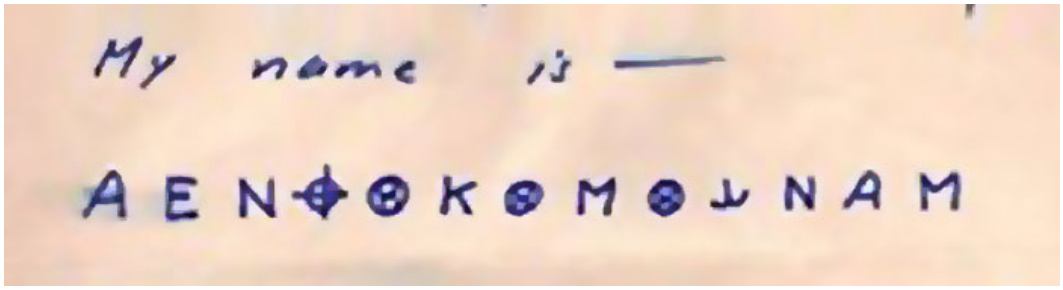



Abbildung 3.5: Die Zodiac-13 Chiffre. Entnommen von [Col]

lange Geheimtext besteht aus 8 unterschiedlichen Homophonen. Lediglich eines der vorkommenden Homophone ist neu, alle anderen Zeichen kommen bereits in Zodiac-340 vor. Genau dieses eine neue Homophon kommt direkt dreimal vor, was bei einem Text dieser Länge nicht zu erwarten wäre. Da alle anderen Zeichen nur ein bis zweimal vorkommen ist es bei der Länge des Textes unwahrscheinlich, dass dieser eigenständig geknackt werden kann. Sollte Zodiac-340 gebrochen werden, so könnte man versuchen, denselben Schlüssel auch auf Zodiac-13 anzuwenden. Sollte dies nicht gelingen, so ist es unwahrscheinlich, dass die Nachricht jemals gebrochen werden kann. Die Berechnung von Unizitätslänge und Schlüsselraum ist in diesem

Fall nicht sinnvoll. Zur Berechnung des Schlüsselraums fehlen Informationen über das genutzte Klartextalphabet. Da weniger Homophone genutzt werden als Lateinische Buchstaben existieren ist eine Berechnung mit 26 Klartextbuchstaben nicht sinnvoll. Zwar kann man für eine Annäherung annehmen, dass eine bestimmt Zahl an Klartextbuchstaben genutzt wurden, jedoch beeinflusst dies massiv die Entropie der Klartextsprache, was die Berechnung der Unizitätslänge verhindert. Dabei bleibt das Problem bestehen, dass nicht bekannt ist, welche Buchstaben der Text wirklich enthält. Daher ist eine eigenständige Lösung der Chiffre ohne dieses Wissen sehr unwahrscheinlich.

#### 3.3.1.3 Zodiac-32

Dies ist die letzte Chiffre, die der Zodiac versandte. Sie erreicht am 26. Juni 1970 die Redaktion des San Francisco Chronicles. Sie bildet das Ende einer längeren, unverschlüsselten Nachricht. In dieser beschwert er sich über die Bewohner San Franciscos, die seine Forderung, Buttons mit dem Symbol des Zodiac zu tragen, nicht nachgekommen sind. Außerdem bekennt er sich zu einem weiteren Mord. Der Brief endet mit der Drohung, er habe eine Bombe versteckt. Zusammen mit ei-

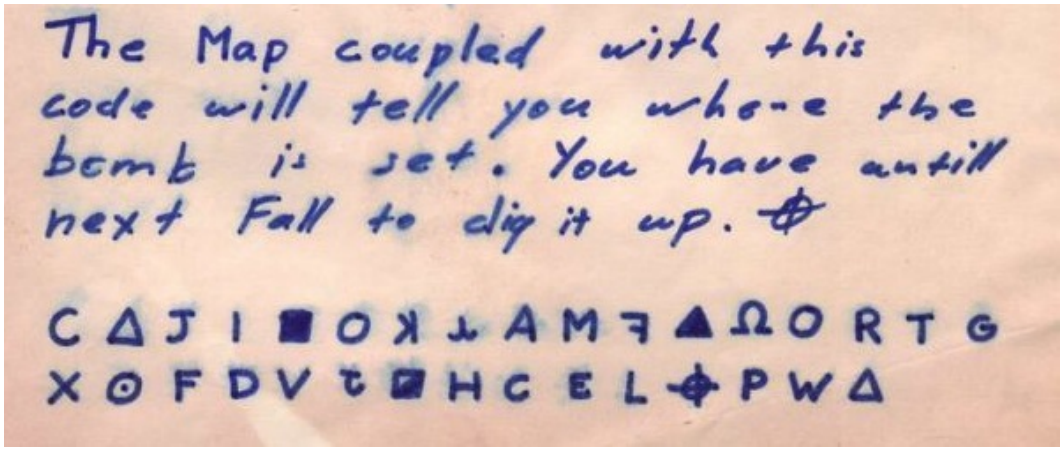



Abbildung 3.6: Die Zodiac-32 Chiffre. Entnommen von [Ora]

ner Karte und dem verschlüsselten Text könne diese gefunden werden. Dies müsse laut ihm bis zum darauffolgenden Frühling geschehen. Die erwähnte Karte, die dem Brief beilag, ist eine Karte von San Francisco und Umgebung. In einem späteren Brief erwähnte der Zodiac, die Nachricht enthielte Radianten und Angaben in Inch. Man kann also davon ausgehen, dass die Chiffre Zahlen enthält, entweder direkt übersetzt in Homophone oder in ausgeschriebener Form. Das größte Problem beim Entschlüsseln dieser Nachricht ist, dass sich nur drei Homophone wiederholen. Die Chiffre besteht aus 32 Zeichen und 29 verschiedenen Homophonen. Das bedeutet, dass sich aus dieser Chiffre fast jeder Klartext der entsprechenden Länge bilden lässt. Auch tauchen in dieser Chiffre zwei neue Homophone auf, die weder in Zodiac-340 noch in Zodiac-13 verwendet wurden. Selbst wenn Zodiac-340

gebrochen werden kann ist es unwahrscheinlich, dass der Killer erneut denselben Schlüssel nutzte und damit auch Zodiac-32 zu einem großen Teil gebrochen werden kann. Eine Lösung von Zodiac-32 zu finden ist daher mindestens so unwahrscheinlich wie eine Lösung für Zodiac-13 zu finden.

### 3.3.2 Die Spanish Strip Cipher

Die Spanish Strip Cipher, im Folgenden als spanische Streifenchiffre bezeichnet, ist ein Verschlüsselungsverfahren, das im Spanischen Bürgerkrieg der Jahre 1936 bis 1939 sowohl von Republikanern als auch von Nationalisten genutzt wurde. Dieser Bürgerkrieg, der einer der Schritte auf dem Weg zum zweiten Weltkrieg war, wurde vom 17. Juli 1936 bis zum 1. April 1939 zwischen den Republikanern, die der Regierung anhangen und unter anderem von der Sowjetuniun und Mexiko unterstützt wurden, und den Nationalisten, die vom späteren Führer Spaniens, General Francisco Franco geführt und unter anderem von Italien, Protugal und Deutschland unterstützt wurden, geführt. Der Krieg endete mit dem Sieg der Nationalisten. Beide Seiten nutzten im Krieg unter anderem die Spanische Streifenchiffre, eine homophone Verschlüsselungstechnik. Diese funktioniert, indem jedem Klartextbuchstaben eine gewisse Zahl von Homophonen zugewiesen wird. Die verwendeten Homophone sind im Fall dieser Chiffre die Zahlen „01“ bis „99“. Dabei gibt es bei der Chiffre die Einschränkung, dass jedem Klartextbuchstaben mindestens drei und höchstens vier Homophone zugewiesen werden. Allerdings sind

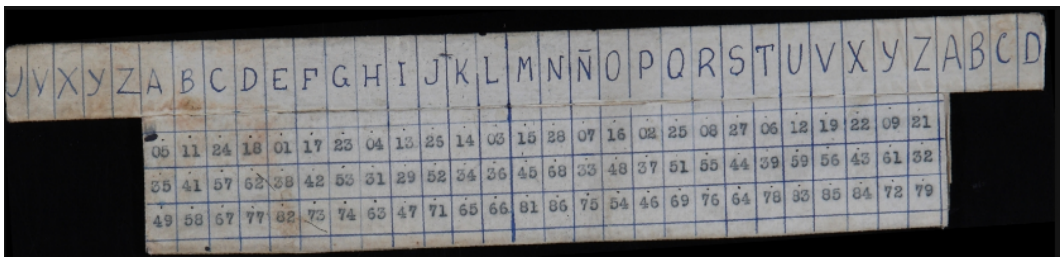



Abbildung 3.7: Historischer Schlüssel der spanischen Streifenchiffre. Entnommen von [Nat]

diese Regeln nicht eindeutig nachvollziehbar, da auch Dokumente existieren, in denen Klartextbuchstaben nur zwei oder sogar fünf Homophone zugeordnet sind. Weiterhin ermöglicht es dieser Aufbau der Chiffre, eine Homophone Chiffre mit einem Codebuch zu verbinden. Weist man jedem der 27 Buchstaben des spanischen Alphabets drei Homophone zu, so bleiben noch 18 Homophone übrig, denen man entweder Buchstaben oder beliebige andere Bedeutungen wie Satzzeichen oder ganze Wörter zuweisen kann. Bei historischen Telegrammen ist nicht immer bekannt, welche Art der Verschlüsselung genutzt wurde. Weiterhin ist auch das zugrunde liegende Klartextalphabet nicht immer eindeutig. So kann es sein, dass spanische Buchstaben wie „CH“ oder „Ll“ eigene Homophone zugewiesen bekommen haben oder nicht. Auch Satzzeichen können, müssen aber nicht mit verschlüsselt worden

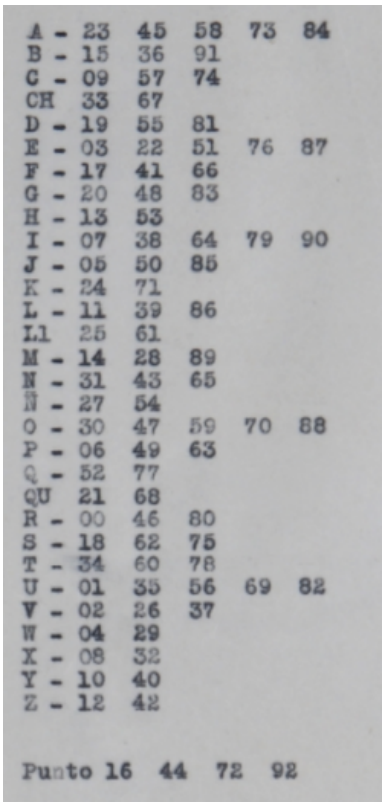
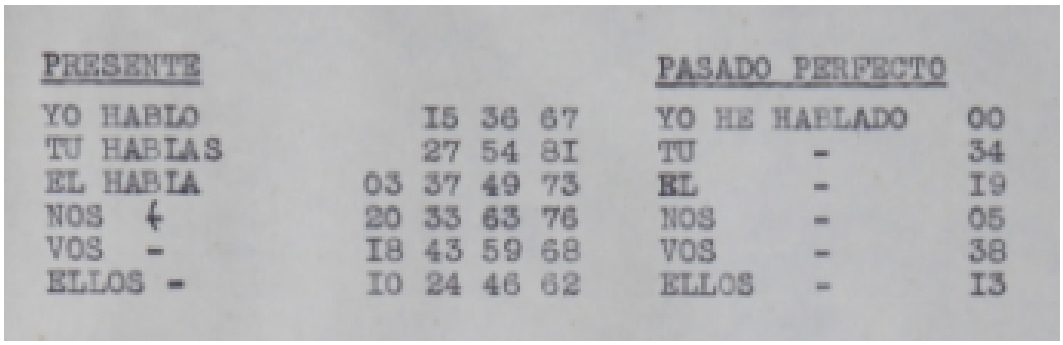



Abbildung 3.8: Alternative Darstellung eines Schlüssels. Entnommen von [Nat]

sein. So taucht zum Beispiel bei vielen historischen Dokumenten der Punkt zum Satzende als Teil des Klartextalphabets auf. Für die Analyse der Verschlüsselung im Rahmen dieser Arbeit wird zuerst die Spanische Streifenchiffre mit den Regeln, dass jeder Klartextbuchstabe des 27 Buchstaben umfassenden spanischen Alphabets mindestens drei und maximal vier Homophone zugewiesen bekommt. In den auf MysteryTwisterC3 gestellten Aufgaben wird weiterhin vor der Verschlüsselung durch die Homophone eine monoalphabetische Verschlüsselung durchgeführt. Dies erhöht zwar intuitiv die Schlüsselraumgröße, stellt sich jedoch schnell als irrelevant heraus. In einem Angriff fällt diese monoalphabetische Substitution nicht weiter auf, da es für den Angreifer keinen Unterschied macht, ob ein Buchstabe erst auf einen anderen Buchstaben abgebildet wird und dann auf ein Homophon oder ob er direkt auf ein Homophon abgebildet wird. Beim Herausfinden des Schlüssels muss das entsprechende Homophon einfach direkt dem ursprünglichen Klartextbuchstaben zugeordnet werden. Dies ist vergleichbar mit der mehrfach hintereinander ausgeführten Anwendung einer Caesar Chiffre. Dies erhöht nicht den Schlüsselraum, sondern ändert nur den Schlüssel. Unsere Voraussetzungen für die Berechnung des Schlüsselraums sind also, dass das Klartextalphabet 27 Buchstaben enthält, es 99 Homophone gibt und jedem Klartextbuchstaben mindestens drei und höchstens vier Homophone zugeordnet sind. Dadurch ergibt sich folgende Berechnung:

$$\binom{99}{27} \cdot 27! \cdot \binom{72}{27} \cdot 27! \cdot \binom{45}{27} \cdot 27! \cdot \frac{27!}{(27-18)!} \approx 4 \cdot 10^{162} \approx 2^{540} \tag{3.7}$$



| PRESENTE | | | | | PASADO PERFECTO | | |
|---|---|---|---|---|---|---|---|
| YO HABLO | | I5 | 36 | 67 | YO HE HABLADO | | 00 |
| TU HABLAS | | 27 | 54 | 8I | TU | - | 34 |
| EL HABLA | 03 | 37 | 49 | 73 | EL | - | I9 |
| NOS ↓ | 20 | 33 | 63 | 76 | NOS | - | 05 |
| VOS - | I8 | 43 | 59 | 68 | VOS | - | 38 |
| ELLOS - | I0 | 24 | 46 | 62 | ELLOS | - | I3 |

Abbildung 3.9: Zuordnung von Homophonen zu kompletten Ausdrücken. Entnommen von [Nat]

Das Ergebnis der Berechnung des Schlüsselraums weicht signifikant von dem von Luis Alberto Sanguino berechneten Ergebnis [LABS16] ab. Jedoch fürt auch sein Resultat, was eine Schlüsselramgröße von $2^{505}$ ist, zu der Erkenntnis, dass ein Brute-Force angriff nicht sinnvoll durchführbar ist. Die Bedeutung der einzelnen Teile der vom Autor dieser Arbeit aufgestellten Formel sind wie folgt zu erklären:

- Die Anzahl der Möglichkeiten, den 27 Klartextbuchstaben jeweils 3 zufällige Homophone zuzuweisen. Zuerst wird jedem Klartextbuchstaben eines der 99 Homophone zugewiesen. Anschließend wird jedem Klartextbuchstabe erneut ein Homophon zugewiesen, wobei zu diesem Zeitpunkt nur noch 72 Homophone zur Verfügung stehen. In einem dritten Schritt wird jedem Klartextbuchstaben ein drittes Homophon zugewiesen, wobei noch 45 Homophone zur Auswahl stehen.
- Die übrigen 18 Homophone werden nun so aufgeteilt, dass jedem Klartextbuchstaben maximal ein Homophon zugeteilt wird, sodass am Ende jeder Klartextbuchstabe mindestens drei und höchstens vier Homophone zugewiesen hat.

Dieser Weg der Berechnung erscheint sinnvoll und ist deutlich einfacher als der Ansatz von Sanguino. Das Ergebnis zeigt, dass ein Brute-Force Angriff auf die Spanische Streifenchiffre aufgrund des zu großen Schlüsselraums nicht sinnvoll ist. Während die beiden leichteren Challenges bei MysteryTwisterC3 feste Vorgaben zur Verschlüsselung haben, gibt es bei Originalnachrichten Informationsdefizite wie zum Beispiel das Klartextalphabet, die eine Analyse erschweren. Bei historischen Nachrichten ist zusätzlich nicht bekannt, ob diese in spanischer oder baskischer Sprache verfasst wurden. Daher müssen bei einem heuristischen Angriff zwei Durchgänge durchgeführt werden, jeweils mit den Statistiken der vermuteten Klartextsprache.

# 4 Konzept und Design

Dieses Kapitel beinhaltet verschiedene Angriffsmöglichkeiten auf Homophone Chiffren im Allgemeinen und die beiden vorgestellten Chiffren im Besonderen. Die Aufteilung möglicher Angriffe in Untergruppen kann anhand der Grafik 4.1 nachvollzogen werden.

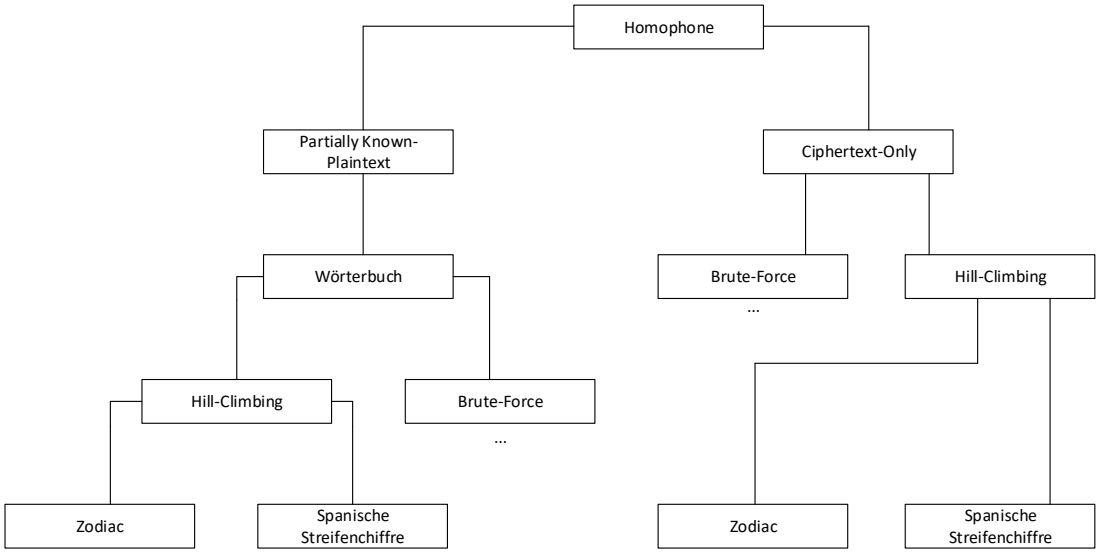



Abbildung 4.1: Überblick verschiedener Angriffsmöglichkeiten auf homophone Chiffren

Man unterscheidet zuerst zwischen Ciphertext-Only und Partially Known-Plaintext Angriffen zu unterscheiden. Ein Known-Plaintext Angriff ist trivial, da in diesem Fall lediglich die verwendeten Homophone den entsprechenden Buchstaben zugeordnet werden können. Diese Art von Angriff wird daher nicht näher betrachtet. Ohne Kenntnis des Klartextes muss ein Ciphertext-Only Angriff durchgeführt werden. Dieser kann auf verschiedene Arten erfolgen. Die einfachste und rechenintensivste Möglichkeit ist ein Brute-Force Angriff. Bei diesem werden alle möglichen Schlüssel einer Chiffre ausprobiert. Dabei muss zwischen den Zodiac Chiffren und den mit der spanischen Streifenchiffre verschlüsselten Nachrichten unterschieden werden. Während für die vom Zodiac angewandte Verschlüsselung keine Regeln bekannt sind kann man den Schlüsselraum der spanischen Streifenchiffre durch die vorgegebenen Regeln eingrenzen. Diese besagen, dass jedem Buchstaben mindestens drei Homophone zugewiesen werden müssen. Nicht ganz klar definiert ist die Obergrenze der Homophone pro Klartextbuchstaben, dies können vier oder fünf sein. Wie in Formel (3.5) berechnet beträgt der Schlüsselraum der Zodiac-340

Nachricht in etwa 337 Bit, was einen Brute-Force Angriff mit Hilfe von dem Stand der Technik entsprechenden Mitteln unmöglich macht. Auch der Schlüsselraum der spanischen Streifenchiffre ist zu groß, um mit einem Brute-Force Angriff eine realistische Erfolgsaussicht zu haben.
Ein alternativer Angriff, bei dem nicht der komplette Schlüsselraum durchsucht wird, ist ein Hill-Climbing Algorithmus. Dieser wählt einen zufälligen Startschlüssel aus und versucht, diesen durch kleine Veränderungen des Schlüssels so lange zu verbessern, bis der bestmögliche Schlüssel gefunden wird. Eine Mischung dieser stellt der Partially Known-Plaintext Angriff dar. Dieser basiert darauf, dass dem Angreifer ein Teil des zu einer Chiffre gehörenden Klartextes bekannt ist. Ausgehend von diesem bekannten Teil der Nachricht können Teile des Schlüssels festgelegt werden, wodurch ein anschließender Ciphertext-Only Angriff eingeschränkt werden kann. Dieses Prinzip kann jedoch auch angewendet werden, wenn dem Angreifer kein Klartext bekannt ist. Anstatt den Schlüssel teilweise durch einen bekannten Klartextabschnitt festzulegen kann dafür auch ein Wörterbuch verwendet werden. Dies funktioniert, indem jedes Wort des Wörterbuches genommen und einmal an jeder Stelle des Geheimtextes getestet wird. Mit dem daraus resultierenden teilweise festgelegten Schlüssel kann anschließend ein Ciphertext-Only Angriff durchgeführt werden. Dies ist zwar kein Known-Plaintext Angriff, jedoch ist die Funktionsweise ähnlich. Der Unterschied besteht darin, dass dem Angreifer in diesem Fall *kein* Klartext bekannt ist sondern dieser geraten wird. Daher zählt dieser Angriff als Ciphertext-Only Attacke. In Grafik 4.1 wird dieser jedoch als Partially Known-Plaintext Angriff dargestellt da die Funktionsweise diesem sehr ähnlich ist.

## 4.1 Generelles Hillclimbing

Ein allgemeiner Hillclimbing Algorithmus für homophone Chiffren ist leicht darzustellen. In einem ersten Schritt muss ein zufälliger Schlüssel generiert werden. Dafür wird zuerst jedem Buchstaben des Klartextalphabets ein zufälliges Homophon zugewiesen. Die restlichen Homophone werden zufällig auf alle Klartextbuchstaben verteilt. Dieser Schlüssel wird nun immer weiter verändert. Für jeden veränderten Schlüssel wird der entsprechende Klartext entschlüsselt. Dieser wird von einer Kostenfunktion bewertet. Zur Bewertung verwendet die Kostenfunktion Statistiken, die aus einer Sprache generiert wurden. Geeignet hierfür sind zum Beispiel Tri- und Tetragramme, welche die Wahrscheinlichkeit des Auftretens bestimmter Buchstabenkombinationen beinhalten. Die Modifikation des Schlüssels geschieht, indem die Bedeutung von Homophonen verändert wird. Dabei kann die Abbildung eines Homophons auf einen Klartextbuchstaben geändert werden, es können aber auch mehrere Homophone gleichzeitig geändert werden. Der Schlüssel wird so lange verändert, bis kein Schlüssel mehr gefunden wird, der von der Kostenfunktion besser bewertet wird.

## 4.2 Hillclimbing mit Wörterbuch

Eine mögliche Verbesserung des allgemeinen Angriffs ist durch die Benutzung eines Wörterbuches zu erzielen. Dadurch werden bestimmte Teile des Schlüssels vor dem Hillclimbing festgelegt. Die Annahme ist in diesem Fall, dass ein im Wörterbuch enthaltenes Wort in dem Text vorkommt. Dazu wird ein Wörterbuch der vermuteten Klartextsprache genutzt. Aus diesem werden zuerst alle Wörter, die einer gewissen Längenbeschränkung entsprechen, ausgewählt. Dies soll flexibel sein, sodass eine minimale und eine maximale Länge angegeben werden kann. Weiterhin sollte auswählbar sein, wie viele Wörter gleichzeitig getestet werden sollen. Zu Beginn des Algorithmus werden die ausgewählten Wörter an den Anfang des Geheimtextes gesetzt. In einer Schleife werden nun alle möglichen Positionen, die diese Worte im Geheimtext einnehmen können, getestet. Dadurch entsteht jedesmal ein Startschlüssel, in dem die Bedeutung einiger Homophone durch die Platzierung der Worte festgelegt ist. Die restlichen Stellen des Schlüssels werden zufällig gesetzt. Der so entstandene Schlüssel wird zum Start des Hillclimbings genutzt. Dabei wird während dem Hillclimbing darauf geachtet, dass durch die Worte festgelegte Homophone nicht mehr verändert werden.
Zur Repräsentierung des Schlüssels sind zwei Ansätze vorstellbar. Entweder wird jedem Homophon genau ein Klartextbuchstabe zugewiesen, oder jedem Klartextbuchstaben wird eine Menge von Homophonen zugewiesen. Dadurch ergeben sich drei verschiedene Ansätze, wie die Schlüsselmodifikation im Hillclimbing Algorithmus durchgeführt werden kann.

- Jedem Homophon ist ein Klartextbuchstabe zugeordnet. Für alle nicht festgelegten Homophone wird schrittweise jeder Klartextbuchstabe ausprobiert und das Ergebnis durch eine Kostenfunktion bewertet. Dies wird wiederholt, bis sich die Bewertung des Schlüssels nicht mehr durch die Änderung der Bedeutung eines Homophons verbessern lässt.

- Jedem Klartextbuchstaben ist eine Menge von Homophonen zugeordnet. Diese Mengen können ausgetauscht werden. So können zum Beispiel alle zum Klartextbuchstaben „A“ gehörenden Homophone mit denen, die zum Klartextbuchstaben „B“ gehören, getauscht werden. Nach jedem Tausch von zwei Mengen wird das Ergebnis wiederum durch eine Kostenfunktion bewertet. Dies wird wiederholt, bis sich durch das Tauschen von zwei Mengen kein besserer Schlüssel mehr finden lässt.

- Es werden nicht die kompletten Mengen der Homophone getauscht, sondern lediglich einzelne Elemente. Dabei können entweder zwei Elemente aus zwei Mengen getauscht werden oder ein Element einer Menge wird in eine beliebige andere Menge verschoben. Auch dies wird durchgeführt, bis sich durch das Tauschen von zwei Elementen kein besserer Schlüssel mehr finden lässt.

Die verschiedenen Ansätze können kombiniert oder nacheinander ausgeführt werden. Entweder werden die Ansätze werden sequentiell nacheinander ausgeführt oder der Algorithmus hat die Möglichkeit, die Ansätze zu mischen. So kann zum

Beispiel Ansatz eins für ein Homophon ausgeführt werden, danach Ansatz zwei auf alle Mengen und anschließend erneut Ansatz eins auf das zweite Homophon.

## 4.3 Hillclimbing für die spanische Streifenchiffre

Für den Angriff auf die spanische Streifenchiffre müssen die den Regeln der Chiffre entsprechenden Einschränkungen berücksichtigt werden. So muss beim Generieren des Startschlüssels darauf geachtet werden, dass jedem Klartextbuchstaben mindestens drei Homophone zugewiesen werden. Weiterhin muss beim Verändern der Bedeutung der Homophone durch den Hillclimbing Algorithmus darauf geachtet werden, dass dadurch keinem Klartextbuchstaben weniger als drei Homophone zugeordnet werden. Die Obergrenze, wie viele Homophone einem Buchstaben zugeordnet werden können, sollte einstellbar sein. Dies ist damit zu begründen, dass in den verschiedenen Aufgaben bei MysteryTwisterC3 verschiedene Voraussetzungen dazu gegeben sind. So ist für die Aufgaben eins und zwei vorgegeben, dass jedem Klartextbuchstaben drei bis vier Homophone zugeordnet sind. In Aufgabe drei können jedoch jedem Buchstaben drei bis fünf Homophone zugeordnet werden.

## 4.4 Verteilung im Rechnernetzwerk

Ein weiterer Aspekt dieser Arbeit ist die Verteilung der Berechnung mit Hilfe der VoluntCloud. Dabei gibt es verschiedene Möglichkeiten, die Berechnungen zu verteilen. Die Erste ist die Verteilung von Startschlüsseln. In diesem Fall muss eine Vorverarbeitung durchgeführt werden. Hierbei wird die Verteilung des Inhalts des Wörterbuchs auf den Text durchgeführt. Die so entstehenden Schlüssel werden gespeichert und die nicht durch das Wörterbuch festgelegten Stellen des Schlüssels durch Platzhalter aufgefüllt. Anschließend werden doppelt vorkommende Schlüssel gelöscht und die Liste der Startschlüssel als zusätzliche Ressource dem Programm in der VoluntCloud übergeben. In diesem wird für die Ausführung eines jeden Blocks ein Startschlüssel ausgewählt. Auf diesen wird anschließend der Hillclimbing Algorithmus ausgeführt. Zur Zuordnung eines Blocks zu einem Startschlüssel kann einfach sequentiell jeder Schlüssel aus der Liste der Reihe nach einem Block zugeordnet werden. Die Anzahl der benötigten Blöcke entspricht in diesem Fall der Anzahl der gefundenen Startschlüssel. Ein zu erwartender Vorteil dieser Methode ist die Reduzierung der zu testenden Schlüssel und somit die Reduzierung der zu verteilenden Aufgaben, da kein Startschlüssel doppelt getestet wird.
Das verwendete Wörterbuch der englischen Sprache enthält circa 80.000 Wörter mit acht Buchstaben. Verteilt man jeweils zwei Wörter der Länge acht auf den Geheimtext, so ergeben sich

$$80.000^2 = 6.400.000.000 \approx 2^{32} \tag{4.1}$$

Möglichkeiten. Zusätzlich gibt es für jede Kombination von zwei Wörtern mehrere Möglichkeiten, diese im Text zu platzieren. Es ist anzunehmen, dass auf diese Weise gleiche Schlüssel mehrfach generiert werden. Dies kann passieren, indem ein Homophone an verschiedenen Stellen des Textes durch unterschiedliche Wörter gleich festgelegt wird. Ein weiterer Vorteil wäre, dass in diesem Fall das Wörterbuch nicht mit verteilt werden müsste, sondern nur die Liste der vorher generierten Schlüssel. Zuletzt würde sich auch die Rechenzeit pro Block verringern, da ein Testen verschiedener Positionen der Wörter nicht mehr durchgeführt werden muss. Der Nachteil dieser Methode ist die Größe der Schlüsselliste. Die am wenigsten Speicherplatz verbrauchende Repräsentation des Schlüssels ist die Darstellung als Byte Array. In diesem Fall wird ein Bytearray erstellt, dessen Größe der Anzahl der im Geheimtext verwendeten Homophone entspricht. Für jedes Homophon wird der entsprechende Klartextbuchstabe als Byte gespeichert und zum Beispiel 255 als Markierung für freie Homophone verwendet. Ausgehend von einem Geheimtext der aus 50 verschiedenen Homophonen besteht, was weniger ist als in den meisten zu analysierenden Chiffren, werden für die Speicherung eines Schlüssels 50 Byte benötigt. Ausgehend davon, dass für zwei Wörter der Länge acht mindestens noch $2^{32}$ verschiedene Schlüssel existieren, benötigt man allein für diese Schlüssel

$$2^{32} \cdot 50\ B \approx 2^{38}\ B \approx 214\ GB \tag{4.2}$$

Speicherplatz. Aufgrund dieses Platzbedarfes kann die Methode als nicht praktikabel angesehen werden, da sie für jede Kombination von Wortlängen und Wortanzahl durchgeführt und die Liste der Schlüssel an jeden mitrechnenden Rechenknoten verteilt werden müsste.
Alternativ kann dem Rechenknoten das Testen der Positionen einer Menge von Wörtern überlassen werden. Dazu wird dem Rechenknoten das Wörterbuch mit übergeben und ein Algorithmus entwickelt, der in Abhängigkeit der Block ID und der Anzahl der zu testenden Wörter eine Menge von Wörtern aus dem Wörterbuch auswählt, die von diesem Block zu testen sind. Das Testen aller Platzierungen dieser Wörter im Text wird von einem Rechenknoten durchgeführt. Für den Fall, dass alle Wortpaare der Länge acht getestet werden sollen, wären somit $2^{32}$ Blöcke nötig, wobei jeder Block alle möglichen Positionen der zwei Wörter im Text ausprobiert und anschließend einen Hillclimbing Algorithmus auf die so entstehenden Startschlüssel ausführt. Unter der Annahme, dass der Rechnerpool der Universität mit circa 100 Rechnern dafür genutzt werden kann, müsste jeder Rechner

$$\frac{2^{32}}{100} \approx 2^{26} \tag{4.3}$$

Blöcke berechnen. Unter der sehr optimistischen Annahme, dass jeder Block in einer Sekunde berechnet werden kann, entspräche dies trotzdem einer Laufzeit von 776 Tagen. Es muss also Entweder die Anzahl der Rechner vergrößert oder die Anzahl der Blöcke verringert werden. Da die Benutzung von signifikant mehr Rechnern im Rahmen dieser Arbeit nicht realistisch ist, muss die Anzahl der Blöcke reduziert werden. Dafür muss entweder die Anzahl der zu testenden Worte verringert werden, was durch eine Einschränkung der Wortlänge möglich wäre. Alternativ

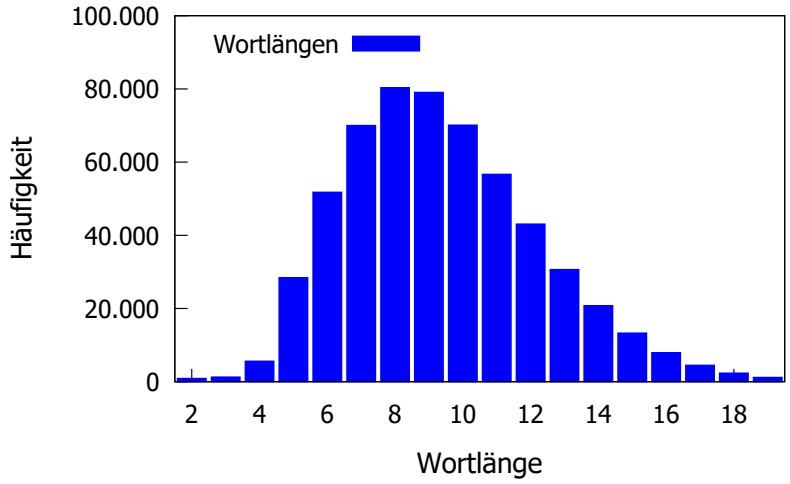



Abbildung 4.2: Verteilung der Wortlängen im Englischen Wörterbuch

könnten statt Wortpaaren nur einzelne Worte getestet werden. Dadurch reduziert sich die Anzahl der Blöcke erheblich. Aus Abbildung 4.2 lässt sich für ein englisches Wörterbuch ablesen, wie viele Worte jeder Länge in diesem enthalten sind. Werden alle Wörter der Längen acht bis zwölf getestet, so sind lediglich 329.316 Blöcke nötig. Unter der Annahme, dass die Berechnung eines Blocks circa eine Minute dauert und jeder Rechner 3294 Blöcke berechnen muss ergibt sich eine Rechendauer von weniger als drei Tagen. Dies ist realistisch mit dem Rechnerpool der Universität durchführbar. Eine Möglichkeit, mehrere Worte gleichzeitig zu testen, soll trotzdem implementiert werden. Liefert der Algorithmus gute Resultate für Testnachrichten wäre es möglich, diesen nach der Fertigstellung dieser Arbeit zu veröffentlichen und auf einer größeren Anzahl von Rechnern laufen zu lassen, sodass die Verringerung der Rechenzeit durch eine Erhöhung der Rechenleistung realisierbar wäre.

# 5 Implementierung

Das folgende Kapitel gibt einen Einblick in den entwickelten Code und getroffene Designentscheidungen. Als Angriff wurde ein Hillclimbing Algorithmus gewählt. Diese Art von Angriff wurde vom Autor dieser Arbeit bereits erfolgreich in seiner Bachelorarbeit angewandt. Weiterhin konnte die originale Zodiac-408 Nachricht bereits mit einem Hillclimbing Algorithmus gebrochen werden. [DAS13] Um diesen Angriff zu verbessern wurde beschlossen, zusätzlich einen Wörterbuchangriff zu nutzen, um vor dem Hillclimbing bereits manche Stellen des Schlüssels festzulegen.

## 5.1 Voraussetzungen

Als Ressourcen für den Angriff werden Wörterbücher und Statistiken für die Bewertung eines Schlüssels im Rahmen des Hillclimbing Algorithmus in den Sprachen Englisch und Spanisch benötigt. Die Englische Statistik und das Wörterbuch wurden aus CrypTool 2 entnommen. Die Spanische Statistik wurde selber generiert. Dafür wurden alle spanischen Texte des „Project Gutenberg“ [gut] heruntergeladen. Anschließend wurden die englischsprachigen Kopf- und Fußabschnitte der Texte entfernt, um nur spanischen Text zur Generierung der Statistik zu erhalten. Im nächsten Schritt wurden alle Sonderzeichen entfernt, sodass nur noch das 27 Buchstaben umfassende spanische Alphabet übrig blieb. Dieser so erhaltene Text wurde nun analysiert, indem er von vorne bis hinten durchgegangen wurde und an jeder Stelle die Kombination der nächsten drei beziehungsweise vier Buchstaben gezählt wurde. Dadurch entstanden zwei Statistiken, die das Vorkommen von Tri- und Tetragrammen beinhalten. Die Resultate wurden später verwendet, um die Qualität eines Schlüssels zu bewerten. Das Wörterbuch wurde durch die Zusammenführung von zwei Wörterbüchern erstellt. Eins wurde aus den aus der Gutenberg Bibliothek extrahierten Texten erstellt, indem alle vorkommenden Wörter aufgelistet und Duplikate entfernt wurden. Ein zweites Wörterbuch wurde aus einem Internetprojekt [spa], das Wörterbücher für den Sublime Editor sammelt, entnommen. Die Wörterbücher wurden weiterhin angepasst, indem alle Wörter in Großbuchstaben umgeformt wurden. Bei den anzugreifenden Chiffren ist eine Unterscheidung zwischen Groß- und Kleinschreibung nicht nötig. Um die Geschwindigkeit beim Einlesen der Wörterbücher zu erhöhen wurden diese außerdem in Binärdateien umgewandelt.

```
string[] Dict = File.ReadAllLines("dictionary.txt");
string Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
List<byte[]> DictWords = new List<byte[]>();
foreach (String s in Dictionary)
{
  byte[] word = new byte[s.Length + 1];
  for (int i = 0; i < s.Length; i++)
  {
    word[i] = (byte)Alphabet.IndexOf(s[i]);
  }
  word[s.Length] = 255;
  DictWords.Add(word);
  [...]
}
```

Dabei wurde jeder Buchstabe in ein Byte umgewandelt. Als Trennzeichen wurde das Byte 255 verwendet. Dies ist nötig, um die Wörter beim Einlesen wieder voneinander trennen zu können.

## 5.2 Architektur

Der Angriff wurde in einer von der VoluntCloud ausführbaren Klassenbibliothek entwickelt. Um dies zu ermöglichen mussten einige von der VoluntCloud vorgegebene Klassen und Methoden implementiert werden. Die Klassen AWorker und AbstractCalculationTemplate sind abstrakte Klassen, die von der VoluntCloud vorgegeben sind und vom Entwickler implementiert werden müssen. Die von AbstractCalculationTemplate erbende Klasse VoluntCloudJob wird beim Starten eines Jobs von der VoluntCloud instanziiert. In dieser kann die URL von nachzuladenden Ressourcen angegeben werden. Außerdem muss in dieser Klasse durch die Variable RequestedBlocks angegeben werden, wie viele Blöcke für diesen Job berechnet werden sollen. Die Methode *MergeResults()* wird von der VoluntLib aufgerufen, wenn ein Rechnerknoten ein Ergebnis liefert. In dieser Methode muss der Entwickler zwei Ergebnisse, zum Beispiel Bestenlisten, zusammenfügen. Im Konstruktor der Klasse VoluntCloudJob wird eine Istanz der Klasse WorkerLogicMock, die von AWorker erbt, erstellt. Zum Start einer Berechnung ruft die VoluntLib die in dieser Klasse implementierte Methode *DoWork()* auf, der als Argument die BlockID übergeben wird. In dieser Klasse muss entsprechend die durchzuführende Berechnung implementiert werden.

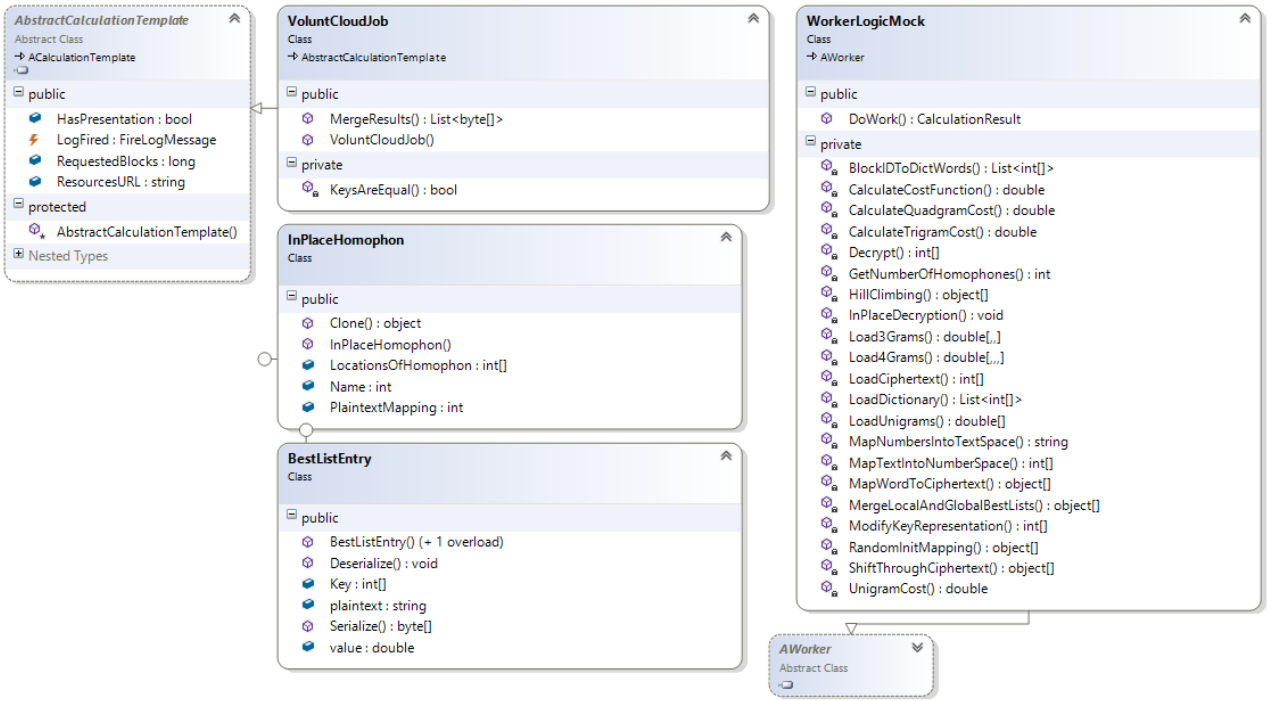



Abbildung 5.1: Klassendiagramm der Implementierung

## 5.3 Laden von Ressourcen

Für die durchzuführenden Berechnungen müssen Ressourcen, die außerhalb des Programms gespeichert wurden, geladen werden. Dies sind das zu verwendende Wörterbuch, der zu analysierende Geheimtext und die zur Bewertung der Schlüssel benötigten Sprachstatistiken. Der Geheimtext wird als Textdatei zur Verfügung gestellt, wobei die einzelnen Homophone als durch Kommata getrennte Zahlen dargestellt sein müssen. Die Homophone müssen von Eins startend durchnummeriert werden. Es darf kein Homophon existieren, dessen Darstellung als Zahl größer ist als die Anzahl der in dem Geheimtext vorkommenden Homophone. Daher war für einige der Challenges zur spanischen Streifenchiffre bei MysteryTwisterC3 eine Vorverarbeitung der Chiffren nötig. Hier wurden die Homophone durch Zahlen von Eins bis n ersetzt, wobei n der Anzahl der in der Chiffre vorkommenden Homophone entspricht. Beim Laden des Geheimtextes wird dieser als Array von Integern gespeichert. Das Wörterbuch wird als Binärdatei zur Verfügung gestellt und als Byte Array eingelesen. Dies wird an den Bytes mit dem Wert 255, der als Trennung verwendet wurde, geteilt und die Wörter als Integer Arrays gespeichert. Dabei entspricht der Zahlenwert eines Wortes an einer Stelle der Position des an dieser Stelle stehenden Buchstabens im Alphabet. Dies wurde implementiert um die Geschwindigkeit des Algorithmus zu erhöhen, da so nicht mit Strings gearbeitet werden muss sondern direkt mit Integer Arrays, die auch für die Berechnung der Kostenfunktion benötigt werden, gearbeitet werden kann. Durch die Variablen MinimumLength und MaximumLength kann eingeschränkt werden, welche Wörter beim Einlesen

des Wörterbuches berücksichtigt werden sollen. So kann zum Beispiel festgelegt werden, dass nur Wörter mit mindestens vier und höchstens sechs Buchstaben berücksichtigt werden sollen. Alle diesen Kriterien entsprechenden Wörter werden schließlich in einer Liste von Integer Arrays gespeichert. Die Sprachstatistiken werden ebenfalls als Binärdateien zur Verfügung gestellt und im Fall der Trigramme als dreidimensionales und im Fall der Tetragramme als vierdimensionales Array von double Werten gespeichert.

## 5.4 Verteilung im Rechnernetz

Die Verteilung der Berechnung geschieht, indem jeder gestartete Job ein anderes Wort oder Wortpaar im Rahmen des Wörterbuchangriffs testet. Welche Wörter in einem Block bearbeitet werden wird in Abhängigkeit von der BlockID berechnet.

```
1  List<int[]> Selector(BigInteger BlockID, List<int[]>
       dict, int nTupel)
2  {
3    BigInteger a = BlockID;
4    int dictLength = dict.Count;
5    List<int[]> res = new List<int[]>();
6    for (int i = 0; i < nTupel; i++)
7    {
8      BigInteger b = a % (BigInteger)dictLength;
9      a = a / dictLength;
10     res.Add(dict[(int)b]);
11   }
12   return res;
13 }
```

Dadurch ist gewährleistet, dass jeder Block ein anderes Wort oder Wörtertupel testet. Außerdem kann eingestellt werden, wieviele Wörter die Funktion auswählen soll. Für jeden Block wird lokal eine Liste der besten Schlüssel erstellt.

## 5.5 Hilfsklassen

Um die Implementierung zu vereinfachen wurden zwei Hilfsklassen erstellt. Eine ist die Klasse InPlaceHomophon. Jedes Objekt dieser Klasse entspricht einem im Text vorkommenden Homophon. In dem Objekt werden der zu dem Homophon gehörende Klartextbuchstabe, eine Liste der Positionen im Text an denen das Homophon verwendet wird und die Bezeichnung des Homophons im Geheimtext gespeichert. Objekte dieser Klasse werden genutzt, um den Schlüssel darzustellen. Diese Darstellung macht es einfacher, Änderungen am Schlüssel durchzuführen, da

somit für jedes Homophon direkt bekannt ist, an welchen Stellen im Geheimtext es vorkommt und die Übersetzung an diesen Stellen direkt geändert werden kann.

```
class InPlaceHomophon
{
  public int PlaintextMapping;
  public int[] LocationsOfHomophon;
  public int Name;
  [...]
}
```

Die zweite Hilfsklasse ist der BestListEntry. Objekte dieses Typs werden genutzt, um die Bestenliste darzustellen. Da die VoluntLib als Rückgabewert für die Do-Work Methode ein Objekt vom Typ CalculationResult fordert, was wiederum die Bestenliste in Form einer Liste von Byte Arrays als Feld beinhalten muss, ist es sinnvoll, eine serialisierbare Klasse zu entwerfen um die einzelnen Elemente der Bestenliste speichern und am Ende in Byte Arrays umwandeln zu können.

```
class BestListEntry : ISerializable
{
  public int[] Key;
  public double value;
  public string plaintext;
  [...]
  public BestListEntry(byte[] b)
  {
    Deserialize(b);
  }
  public void Deserialize(byte[] b)
  [...]
  public byte[] Serialize()
  [...]
}
```

In Objekten dieses Typs werden lokal der Schlüssel, der zum Schlüssel gehörende Klartext und der dazugehörige Kostenwert gespeichert. Zum Umwandeln der Objekte in Byte Arrays kann die Methode Serialize verwendet werden. Dadurch kann die Bestenliste vor dem Versenden des Resultats in eine Liste von Byte Arrays umgewandelt werden, welche wiederum dem CalculationResult zugewiesen werden kann und von der VoluntLib als Ergebnis des Blocks verteilt wird.

## 5.6 Zusammenfügen von Bestenlisten

In der Funktion MergeResults, in der Bestenlisten zusammengefasst werden, können die Listen von Byte Arrays, die der Methode von der VoluntLib als Argumente

übergeben werden, in Listen von Objekten des Typs BestListEntry umgewandelt werden. Dies geschieht, indem für jedes Byte Array der Liste der Konstruktor der Klasse BestListEntry mit dem Byte Array als Argument aufgerufen wird.

```
1  List<byte[]> MergeResults(IEnumerable<byte[]>
       oldResultList, IEnumerable<byte[]> newResultList)
2  {
3    List<BestLEntry> merList = new List<BestLEntry>();
4    List<BestLEntry> oldList = new List<BestLEntry>();
5    List<BestLEntry> newList = new List<BestLEntry>();
6    foreach (byte[] b in oldResultList)
7    {
8          BestLEntry ble = new BestLEntry(b);
9          oldList.Add(ble);
10   }
11   foreach (byte[] b in newResultList)
12   {
13         BestLEntry ble = new BestLEntry(b);
14         newList.Add(ble);
15   }
16   merList = oldList.Concat(newList).ToList();
17   merList = MergedList.OrderByDescending(o => o.value).
       ToList();
18   [...]
19 }
```

Die beiden Listen werden zusammengefügt und die so entstandene neue Liste anschließend nach den Kostenwerten die in den Objekten gespeichert sind sortiert. Im Anschluss muss lediglich geprüft werden, ob Schlüssel doppelt in der Liste vorkommen. Duplikate werden wenn vorhanden entfernt. Zuletzt wird die neue Liste auf 50 Elemente gekürzt, die Objekte wieder serialisiert und die so entstandene Liste von Byte Arrays an die VoluntLib zurückgegeben.

## 5.7 Wörterbuchangriff

Die von dem oben dargestellten Algorithmus ausgewählten Wörter werden im ersten Schritt über den Text geschoben. Dies wird realisiert, indem zu Beginn die ausgewählten Wörter nacheinander an den Beginn des Geheimtextes gesetzt werden. Anschließend wird das letzte Wort Schritt für Schritt nach hinten verschoben, bis es am Ende des Textes steht. In dem Fall wird das davor stehende Wort um eine Position nach hinten Verschoben und das, beziehungsweise bei drei und mehr Worten die nachfolgenden Wörter, werden an das gerade nach hinten geschobene Wort angehängt. Dies wird in einer Schleife ausgeführt bis alle Worte

am Ende des Geheimtextes stehen. Da der Algorithmus zur Auswahl der Worte alle möglichen Kombinationen liefert ist es nicht nötig, beim Verschieben der Worte im Text die Positionen der Worte zu vertauschen. Für jede Positionierung der ausgewählten Wörter im Text wird in einem nächsten Schritt der Teil des Schlüssels festgelegt, der sich durch die Platzierung der Wörter im Geheimtext ergibt. Dabei muss darauf geachtet werden, dass dasselbe Homophon nicht auf zwei verschiedene Klartextbuchstaben abbilden müsste, um die Positionierung der Wörter im Text zu ermöglichen. Dazu wird eine Liste verwendet, in der gespeichert wird, welche Homophone bereits festgesetzt wurden. Jedes Homophon, das durch ein an dessen Stelle im Geheimtext stehendes Wort festgelegt wurde, wird dieser Liste hinzugefügt. Bevor ein Homophon festgelegt wird wird geprüft, ob es bereits in dieser Liste steht. Ist dies der Fall, so wird die Platzierung der Wörter im Text übersprungen und mit dem Verschieben fortgefahren, da sich aus den getesteten Positionen der Wörter kein valider Schlüssel bilden lässt. Die Liste der festgelegten Homophone wird auch weiter verwendet, wenn durch die Positionierung der Wörter ein valider Schlüssel gebildet werden kann. In diesem Fall wird der Hillclimbing Algorithmus gestartet. Diesem wird die Liste der festgelegten Homophone übergeben, damit diese im Lauf des Hillclimbings nicht verändert werden. Ebenso wird dem Hillclimbing Algorithmus der erstellte, teilweise festgelegte Schlüssel übergeben. Für den Fall, dass ausgewählt wurde, dass keine Worte über den Geheimtext geschoben werden sollen, wird der Hillclimbing Algorithmus ohne teilweise vorgegebenen Schlüssel und ohne festgelegte Homophone gestartet.

## 5.8 Hillclimbing

Zu Beginn des Hillclimbing Algorithmus werden zwei Bestenlisten erstellt. In einer werden die besten gefundenen Schlüssel und in einer zweiten die dazugehörigen Bewertungen durch die Kostenfunktion gespeichert. Anschließend beginnt der eigentliche Algorithmus. Wie oft dieser neu gestartet wird kann der Entwickler über eine Variable einstellen. Zu Beginn eines jeden Neustarts wird ein zufälliger Startschlüssel generiert. Dieser basiert auf dem übergebenen, teilweise festgelegten Schlüssel. Die noch nicht festgelegten Homophone werden für jeden Neustart des Algorithmus zufällig festgelegt. Dabei kann angegeben werden, ob die Regeln der spanischen Streifenchiffre beachtet werden sollen oder nicht. Die Speicherung des Schlüssels geschieht auf zwei Arten:

- Als Array von Objekten. Die Länge des Arrays entspricht der Anzahl der vorkommenden Homophone. Jedes Homophon wird durch ein Objekt des Typs „InPlaceHomophon“ dargestellt. Das Homophon mit der Bezeichnung $n$ im Geheimtext wird an der Position $n - 1$ in diesem Array abgespeichert.
- Als Array von Listen von Objekten. Das Array hat in diesem Fall die Länge des genutzten Klartextalphabets. An jeder Position des Arrays ist eine Liste der Objekte, die dem an dieser Position im Klartextalphabet stehenden Buchstaben zugeordnet sind

Die Nutzung von zwei verschiedenen Schlüsseldarstellungen wurde gewählt, da beide Darstellungen die Durchführung verschiedener Aufgaben erleichtern. Die Darstellung als Liste erlaubt ein einfaches Entschlüsseln des Geheimtextes, da in diesem Fall lediglich für jede Stelle des Geheimtextes das entsprechende Homophon betrachtet und dessen Klartextbedeutung übernommen werden muss. Die Darstellung als Array von Listen ist für den zweiten Schritt des Hillclimbings nützlich, in dem Mengen von Homophonen getauscht werden. Durch die Speicherung in Listen kann einfach auf diese Mengen von Homophonen zugegriffen und die Position der Listen im Array getauscht werden.
Die Generierung des Startschlüssels erfogt primär anhand der ersten Darstellung während die zweite Darstellung parallel erstellt wird. Zuerst wird gezählt, ob es mehr Homophone, die noch keinem Klartextbuchstaben zugewiesen sind, als Buchstaben im Klartextalphabet gibt. Ist dies der Fall, so wird zuerst jeder Klartextbuchstabe nacheinander einem zufälligen, noch nicht festgelegten Homophon zugewiesen. Sobald jeder Buchstabe mindestens einmal vorkommt werden den restlichen noch nicht festgelegten Homophonen zufällig Buchstaben zugewiesen. Für den Fall, dass weniger freie Homophone existieren als es Buchstaben im Klartextalphabet gibt, so werden den Homophonen direkt zufällige Homophone zugewiesen. Durch eine Variable hat der Entwickler die Möglichkeit, die Einschränkung der Spanischen Streifenchiffre zu aktivieren. Ebenfalls kann definiert werden, wieviele Homophone mindestens und höchstens einem Klartextbuchstaben zugewiesen werden können.
Die Veränderung des Schlüssels im Laufe des Algorithmus geschieht auf drei verschiedene Arten. Jede dieser Veränderungen kann über eine Variable ein- oder ausgeschaltet werden. Diese Schlüsselmodifikationen werden sequentiell so lange ausgeführt bis sich durch eine erneute Modifikation keine Verbesserung des Schlüssels mehr erreichen lässt.
Im ersten Schritt wird die erste Darstellung des Schlüssels mit einer Schleife durchlaufen. Für jedes Homophon wird zuerst geprüft, ob es durch das Wörterbuch festegelegt ist.

```
1  for(int i = 0; i < Key.Length; i++)
2  {
3    BestFoundKey = Key;
4    for(int j = Alphabet.Length; j++)
5    {
6      Key[i].PlaintextCharacter = j;
7      if(CostValue(Key) > BestKeyValueSoFar)
8      {
9        BestFoundKey = Key;
10     }
11   }
12   Key = BestFoundKey;
13 }
```

Ist dies der Fall, wird es übersprungen. Ansonsten wird für das Homophon jeder Klartextbuchstabe ausprobiert. Für jeden so entstandenen Schlüssel wird getestet, ob dieser besser als der bisher beste gefundene Schlüssel ist. Ist dies der Fall, so wird der Schlüssel in einer temporären Variable gespeichert. Wurden für ein Homophon alle Klartextbuchstaben getestet, so wird der beste gefundene Schlüssel übernommen und das nächste Homophon getestet. Im zweiten Teil des Hillclimbings werden die Listen von Homophonen, die Klartextbuchstaben zugeordnet sind, getauscht. Dazu wird die zweite Darstellung des Schlüssels genutzt. Das Array, in dem die Listen gespeichert sind, wird in zwei verschachtelten Schleifen durchlaufen.

```
1  for(int i = 0; i < Array.Length; i++)
2    {
3    for(int j = i+1; j < Array.Length; j++)
4    {
5      [...]
6      if(settingsWork)
7      {
8        Swap(Array[i], Array[j]);
9      }
10     [...]
11   }
12 }
```

Dabei wird zuerst geprüft, ob in den beiden ausgewählten Listen Homophone enthalten sind, die durch das Wörterbuch festgelegt wurden. Ist dies der Fall, wird das Vertauschen der Listen nicht durchgeführt. Wenn ein Vertauschen möglich ist wird dies signalisiert, indem die Boolean Variable *settingsWork* auf *true* gesetzt wird. Anschließend werden die Listen, die an den Positionen $i$ und $j$ im Array gespeichert sind, vertauscht. Alle Homophone, die vorher dem Klartextbuchstaben $i$ zugeordnet waren sind jetzt dem Klartextbuchstaben $j$ zugeordnet und umgekehrt. Nach jedem Tausch wird ebenfalls geprüft, ob der so entstandene Schlüssel besser ist als alle vorherigen. Ist dies der Fall, so wird der Algorithmus fortgeführt und der Schlüssel in die Bestenliste aufgenommen. Ist dies nicht der Fall, so wird der Tausch rückgängig gemacht.
In der dritten Phase wird die Klartextbedeutung von zwei Homophonen getauscht. Dazu wird die erste Darstellung des Schlüssels mit zwei Schleifen durchlaufen. In der äußeren Schleife wird nacheinander jedes Homophon ausgewählt. Ist dies in der Liste der festgelegten Homophone enthalten, so wird es übersprungen. Wenn nicht wird in der inneren Schleife über alle Homophone iteriert. Dabei wird geprüft, ob das Homophon in der Liste enthalten ist oder dem in der äußeren Schleife ausgewählten Homophon entspricht. Sind beide Bedingungen nicht erfüllt, so wird die Bedeutung der beiden Homophone getauscht.

```
for (int i = 0; i < KeyLength; i++)
  {
  for (int j = 0; j < KeyLength; j++)
  {
    [...]
    int temp = HomophoneKey[i].PlaintextMapping;
    HomophoneKey[i].PlaintextMapping = HomophoneKey[j].
       PlaintextMapping;
    HomophoneKey[j].PlaintextMapping = temp;
    [...]
  }
}
```

Nach allen drei Schritten wird geprüft, ob ein besserer Schlüssel gefunden wurde. Dazu wird nach jeder Veränderung des Schlüssels der dazugehörige Klartext bewertet. Dies geschieht durch eine Kostenfunktion, deren Bewertungsalgorithmus konfigurierbar ist. Dazu kann ausgewählt werden, ob die Bewertung anhand von Unigrammen, Trigrammen, Tetragrammen oder einer Kombination dieser durchgeführt wird. Diese Bewertung des Schlüssels wird jedesmal mit dem besten bis zu dem Zeitpunkt gefundenen Kostenwert verglichen. Ist der Kostenwert des aktuellen Schlüssels besser, so wird der beste Kostenwert auf diesen gesetzt und eine Variable, die markiert dass in diesem Durchgang ein besserer Schlüssel gefunden wurde gesetzt. Wird in einem kompletten Durchgang aller drei Schritte kein besserer Schlüssel gefunden, so endet der Algorithmus und ein neuer Durchlauf beginnt. Sind alle Neustarts beendet, so gibt die Methode die Liste der besten gefundenen Schlüssel zurück. Diese wird an die Methode, die die Wörter im Geheimtext platziert, zurückgegeben und beim nächsten Aufruf des Hillclimbings wieder an diese übergeben, sodass die Liste kontinuierlich aktuell gehalten wird. Neue gefundene Schlüssel werden direkt in diese Liste eingefügt. Wurden alle Positionen der Wörter im Geheimtext getestet, heißt das, dass alle Berechnungen des Blocks durchgeführt wurden. Anschließend wird die Bestenliste wie beschrieben in ein Byte Array umgewandelt und der VoluntLib übergeben.
Eine nachträglich implementierte Optimierung ist die In-Place Entschlüsselung. Vorher wurde nach jeder Veränderung des Schlüssels im Hill-Climbing Algorithmus der komplette Text entschlüsselt um die Kostenfunktion zu berechnen. Durch diese Änderung wird nicht mehr der komplette Text entschlüsselt. In Phase Eins des Hill-Climbing Algorithmus wurde bisher für jede Stelle des Schlüssels geprüft, ob sich durch das Ändern des Klartextmappings dieses Homophons der Klartext verbessert. Statt nach jeder Änderung den Text zu entschlüsseln wird durch die In-Place Entschlüsselung im in 5.8 beschriebenen Algorithmus nur noch in der äußeren Schleife der Text entschlüsselt, also für jede Stelle des Schlüssels einmal. In der inneren Schleife werden nur die Positionen des Klartextes geändert, die von der Änderung des Schlüssels betroffen sind. Das sind die Stellen im Geheimtext, in denen das gerade getestete Homophon steht. Der Rest des Klartextes wird nicht

erneut dechiffriert. Da für jedes Homophon in der äußeren Schleife des Algorithmus einmal eine neue Entschlüsselung durchgeführt wird, muss der Klartext im Fall eines schlechteren Schlüssels auch nicht zurückgesetzt werden, da sobald alle Buchstaben für ein Homophon getestet wurden eine komplette Entschlüsselung durchgeführt wird. In der inneren Schleife, in der die In-Place Entschlüsselung verwendet wird, wird immer dasselbe Homophon verändert, wodurch sich der Klartext immer an denselben Stellen ändert.
Anders verhält es sich im zweiten Teil des Hill-Climbing Algorithmus. In diesem werden Listen von Homophonen getauscht, wodurch sich die Positionen des Klartextes, die modifiziert werden müssen, in jedem Schritt ändern. Daher wird hier vor jeder Veränderung des Schlüssels der bisherige Klartext in einer zusätzlichen Variable gespeichert. Verbessert sich der Schlüssel durch die Änderung, so wird der Klartext beibehalten. Verbessert sich der Schlüssel nicht und der Tausch der beiden Listen wird rückgängig gemacht, so wird auch der Klartext auf den Wert vor dem Tausch zurückgesetzt. Dieselbe Vorgehensweise wird im dritten Schritt des Hillclimbings angewendet.

# 6 Evaluation

Im folgenden Kapitel wird der implementierte Angriff analysiert. Die Evaluation lässt sich dabei in zwei Abschnitte unterteilen. Zuerst lässt sich die Kryptoanalyse selbst evaluieren. Zusätzlich lässt sich die Verteilung des Algorithmus mit Hilfe der VoluntCloud evaluieren.

## 6.1 Evaluation der Kryptoanalyse

In diesem Abschnitt wird die Qualität der Kryptoanalyse evaluiert. Dafür wurde diese mit einem kleineren Wörterbuch auf selber erstellte Geheimtexte angewandt. Die genutzten Chiffren wurden aus einem Textausschnitt des Buches „Alice im Wunderland“ von James Carol generiert.

### 6.1.1 Metriken

Die Qualität der Kryptoanalyse kann daran gemessen werden, ob die Geheimtexte korrekt dechiffriert werden können oder nicht. Dabei können verschiedene Parameter betrachtet werden. Analysiert wird jeweils, mit welcher Wahrscheinlichkeit der Geheimtext korrekt entschlüsselt werden kann.

- Die Textlänge: Wie verändert sich die Wahrscheinlichkeit, dass ein Geheimtext korrekt dechiffriert werden kann, in Abhängigkeit von der Textlänge?
- Der Angriffstyp: Werden die Klartexte unter Zuhilfenahme eines Wörterbuches mit einer höheren Wahrscheinlichkeit gefunden als von einem reinen Hillclimbing Algorithmus?
- Anzahl verwendeter Homophone: Wie beeinflusst die Anzahl der Homophone, die in einem Text vorkommen, die Wahrscheinlichkeit, dass der Geheimtext korrekt dechiffriert werden kann?

Da alle im Rahmen dieser Arbeit untersuchten Nachrichten zwischen 50 und 70 Homophonen enthalten wurde der Fokus auf die Abhängigkeit der Erfolgswahrscheinlichkeit von der Textlänge und der Angriffsart gelegt. Die Abhängigkeit von der Anzahl der vorkommenden Homophone wurde nicht evaluiert.

### 6.1.2 Messungen

Zur Evaluation wurden Geheimtexte verschiedener Längen verwendet. Der kürzeste Geheimtext war 200, der Längste 850 Zeichen lang. Alle Geheimtexte bestanden zufällig aus zwischen 50 und 70 Homophonen. Bei der Generierung der Geheimtexte wurden die Regeln der Spanischen Streifenchiffre beachtet. So wurden insgesamt 104 Homophone genutzt und jedem Klartextbuchstaben vier Homophone zugewiesen. Von diesen 104 Homophonen wurden im Geheimtext jedoch nur zwischen 50 und 70 genutzt. Das verwendete Wörterbuch bestand aus 13 im Klartext vorkommenden Wörtern die zwischen vier und zwölf Zeichen lang waren. Verglichen wurden die Erfolgswahrscheinlichkeit eines Wörterbuchangriffs mit anschließendem Hillclimbing und eines reinen Hillclimbing Angriffs. Ein Angriff wurde als erfolgreich gezählt wenn im Verlauf des Angriffs ein Klartext gefunden wurde, der zu mehr als 80% mit dem originalen Klartext übereinstimmte. Dies ist damit zu begründen, dass ein solcher Klartext dem Nutzer in der Bestenliste auffallen würde. Eine höhere Übereinstimmung ist nicht nötig, da ein zu 80% korrekter Text bereits gut lesbar ist und von Hand nachbearbeitet werden kann. Die Angriffe wurden jeweils 100 mal neu gestartet um die Wahrscheinlichkeit, mit der ein Angriff zum Erfolg führt, evaluieren zu können.
Der Wörterbuchangriff wurde durchgeführt, indem jeweils ein Wort über den Text geschoben und an jeder passenden Stelle ein Hillclimbing Algorithmus mit fünf Neustarts durchgeführt wurde. Dadurch ergaben sich für jedes Wort ungefähr 750 Neustarts, ausgehend von 150 passenden Positionen des Wortes im Geheimtext. Auf die 13 Inhalte des Wörterbuches hochgerechnet wurde der Hillclimbing Algorithmus also circa 10.000 mal ausgeführt. Der reine Hillclimbing Algorithmus wurde mit 500 Neustarts durchgeführt.
Um die Evaluation durchzuführen wurde der Code der Angriffs aus der für die VoluntCloud entwickelten Bibliothek entnommen und manuell auf mehrere Threads aufgeteilt. Diese Berechnungen wurden anschließend auf mehreren Rechnern sowohl lokal als auch auf verschiedenen Servern verteilt. Dieser Weg wurde gewählt, da es im Lauf der Arbeit immer wieder zu Problemen mit der VoluntCloud kam die so umgangen werden sollten.

Wie in 6.1 zu sehen ist benötigt der Wörterbuchangriff deutlich weniger Text um zu einem guten Ergebnis zu kommen als der reine Hillclimbing Algorithmus. Während der Wörterbuchangriff bereits bei Texten mit 450 Zeichen zu 100% das korrekte Ergebnis fand, so war dies bei dem reinen Hillclimbing Algorithmus erst bei Texten die länger als 750 Zeichen waren der Fall. Ein Fazit der Evaluation ist, dass die Verwendung eines Wörterbuches diesen Angriff tatsächlich verbessert und die Erfolgswahrscheinlichkeit signifikant erhöht. Das bedeutet, dass durch die Kombination eines Hillclimbing Algorithmus mit einem Wörterbuchangriff im Rahmen dieser Arbeit eine Verbesserung der Angriffsqualität erzielt werden konnte. Jedoch zeigt dieses Ergebnis auch, dass es mit dem entwickelten Algorithmus nicht möglich ist, kurze Chiffren wie die Nachrichten der spanischen Streifenchiffre bei MysteryTwister oder die originalen Zodiac Nachrichten vollkommen automatisch zu brechen. Besonders die Challenges zur spanischen Streifenchiffre sind mit

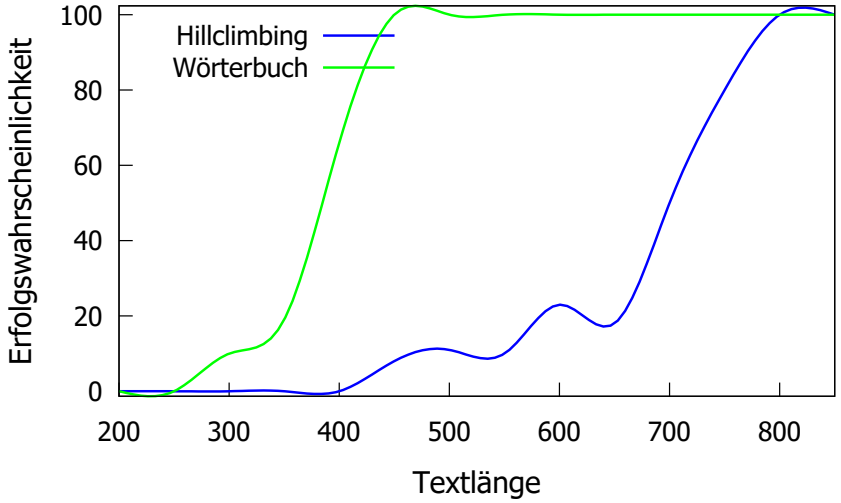



Abbildung 6.1: Erfolgswahrscheinlichkeit des Algorithmus mit und ohne Wörterbuchangriff in Abhängigkeit der Textlänge.

201, 326 und 142 Zeichen deutlich zu kurz um von dem Algorithmus zuverlässig gebrochen werden zu können. Für das Lösen kürzerer Chiffren muss also manuell nachgeholfen werden. Dies kann passieren, indem im Lauf des Angriffs Teile des Klartextes erkannt werden. Diese können dann festgelegt werden, sodass ein Teil des Schlüssels von Anfang an feststeht. Anschließend kann der Algorithmus erneut gestartet werden. Dabei wird versucht, neue Teile des Klartextes zu erkennen. Dieser Vorgang wird so lange wiederholt, bis der komplette Klartext entschlüsselt werden konnte.

## 6.2 Evaluation der Verteilung

In diesem Abschnitt wird auf die Qualität der Verteilung der Berechnung mit Hilfe der VoluntCloud eingegangen.

### 6.2.1 Metriken

Zur Evaluation der Verteilung ist die Beschleunigung der Berechnung durch die Verteilung auf mehrere Rechner zu betrachten. Im Idealfall müsste die Geschwindigkeit linear mit der Anzahl der verwendeten Rechner steigen.

### 6.2.2 Messungen

Da bei der Verteilung mit der VoluntCloud einige Probleme auftraten war es im Rahmen dieser Arbeit nicht möglich, exakte Messungen durchzuführen. Jedoch

ist der durch die Verteilung mit der VoluntCloud entstehende Mehraufwand relativ gering. Daher kann davon ausgegangen werden, dass die Beschleunigung der Analyse proportional zu der zur Verfügung stehenden Rechenleistung ist. Daher wurden für die Evaluation einige Messungen lokal durchgeführt. Die Bedeutung der Messungen wird in diesem Abschnitt interpretiert. Der Testrechner ist ein Intel Core i5 Prozessor mit zwei Kernen die jeweils über Hyperthreading verfügen, also vier virtuelle Prozessoren zur Verfügung stellen. Die maximale Taktfrequenz des Prozessors beträgt 2,5GHz. Gemessen wurden die durchschnittliche Dauer des Hillclimbing Algorithmus mit 500 Neustarts sowie die durchschnittliche Dauer für das komplette Testen eines Wortes. Getestet wurde mit einem Geheimtext der Länge 350.
Die durchschnittliche Dauer eines Hillclimbing Angriffs mit 500 Neustarts betrug mit diesem Setup circa zwei Stunden. Pro Stunde sind mit einem Prozessorkern also in etwa 250 Neustarts realisierbar. Wie in 6.2 erkennbar ist können so bereits

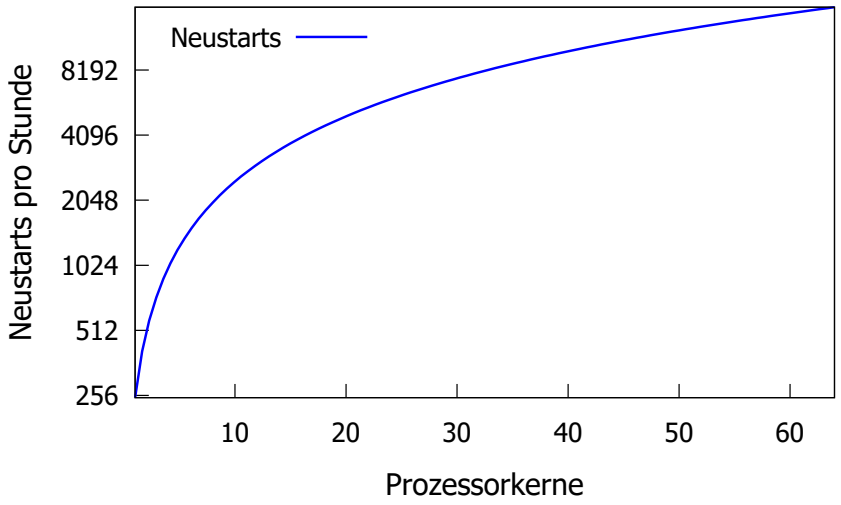



Abbildung 6.2: Theoretisch mögliche Neustarts pro Stunde in Abhängigkeit der verfügbaren Prozessoren.

mit 50 Prozessorkernen mehr als 10.000 Neustarts pro Stunde durchgeführt werden. Zur Lösung der zweiten Challenge zur spanischen Streifenchiffre wurde der Hillclimbing Algorithmus zum Beispiel mit 16.000 Neustarts durchgeführt. Verteilt auf 50 Prozessorkerne ließe sich die Berechnung also in etwa eineinhalb Stunden durchführen.
Das Verschieben eines zehn Buchstaben langen Wortes über den Geheimtext mit jeweils fünf Neustarts des Hillclimbing Algorithmus dauerte im Durchschnitt circa eine Stunde und fünf Minuten. Diese Methode konnte in der Evaluation bei Texten die länger als 450 Zeichen waren immer den korrekten Schlüsel finden. Daher ist diese Methode bei Verwendung eines guten Wörterbuches sehr erfolgversprechend. Ausgehend von der Annahme, dass der Mehraufwand durch die Verteilung zu vernachlässigen ist, kann angenommen werden dass es mit einem Prozessorkern möglich ist, in einer Stunde ein Wort über den Text zu schieben und an jeder Stelle

einen Hillclimbing Algorithmus mit 5 Neustarts auszuführen.
Das genutzte englische Wörterbuch hat insgesamt circa 581.000 Einträge. Zur Vereinfachung nehmen wir an, dass alle Einträge zehn Buchstaben lang sind. Demnach würde ein einzelner Prozessor circa 581.000 Stunden benötigen um alle diese Wörter zu testen. Dies würde einer Laufzeit von circa 66 Jahren entsprechen. Verteilt man diesen Algorithmus auf den Rechnerpool der Universität Kassel mit circa 50 Rechnern mit jeweils 4 Kernen, so erhält man eine theoretische Laufzeit von nur noch ungefähr 60 Tagen. Durch die Verteilung der Berechnung können somit auch für komplette Wörterbücher deutlich realistischere Laufzeiten erreicht werden.

## 6.3 Entschlüsselung der Nachrichten

Eines der Ziele der Arbeit war das Lösen der bei MysteryTwisterC3 veröffentlichten Challenges zur Spanischen Streifenchiffre und dem Zodiac. Während bei MysteryTwisterC3 nur eine Nachricht, die der des Zodiac Killers nachempfunden ist, existiert, gibt es drei verschiedene Challenges zur Spanischen Streifenchiffre. Die Challenges bei MysteryTwisterC3 sind je nach ihrer Schwierigkeit in vier verschiedene Level eingeteilt. Level 1 Challenges sind mit Stift und Papier lösbar, Level 2 Challenges erfordern Programmierkenntnisse, Level 3 Challenges erfordern eine überdurchschnittliche Rechenleistung. Level X Challenges sind Rätsel, die bis heute nicht gelöst werden konnten. Die beiden ersten Challenges der Spanischen Streifenchiffre entsprechen dem Level 2 während die dritte Challenge eine Level X Challenge ist. Die Challenge zur Zodiac Chiffre ist eine Level 1 Challenge und damit nominell die einfachste der zu lösenden Aufgaben. Da das Verschieben eines Wortes auf dem Geheimtext mit einem nachfolgenden Hillclimbing Algorithmus zeitintensiver als erwartet war und bereits bei 50 Neustarts des Hillclimbing Algorithmus in etwa zwei Stunden dauerte musste die Idee, ein komplettes Wörterbuch über den Geheimtext zu schieben, vereinfacht werden. Als Alternative wurde beschlossen, die Wörterbücher zu kürzen und nur Wörter zu nutzen von denen entweder bekannt war dass sie in der Nachricht vorkommen oder die in der Nachricht vermutet wurden.

### 6.3.1 Spanische Streifenchiffre 1

Die erste Challenge zur Spanischen Streifnechiffre besteht aus der Verschlüsselung eines spanischen Telegramms. Als Hinweis ist zusätzlich angegeben, dass der Klartext nicht den Buchstaben Ñ enthält. Der Geheimtext ist 201 Zeichen lang und besteht aus 67 verschiedenen Homophonen. Zuerst wurde ein Wörterbuchangriff auf diese Chiffre angewandt. Das benutzte Wörterbuch wurde aus spanischen Wörtern, die in der Arbeit von Luis Alberto Sanguino [LABS16] erwähnt werden, zusammengestellt. Dies umfasste 29 Wörter. Jedoch konnte durch den Algorithmus kein sinnvoller Text gefunden werden. Zwar fanden sich in der Bestenliste einige Schlüssel die zu Klartexten führten, die spanische Wörter enthielten. Jedoch standen diese

in keinem sinnvollen Zusammenhang. So entstanden Passagen, die zwar aus aneinandergereihten spanischen Wörtern bestanden, jedoch keine spanischen Sätze ergaben. Der Kostenwert des besten gefundenen Schlüssels lag bei -3365, wobei ein höherer Wert einer besseren Bewertung des Schlüssels entspricht. Der beste gefundene Klartext lautete wie folgt:

```
ASMID OSIER RANOS ACONC UEPRE SIDEN TAHOY OUELA BALPR EMBAL
OSECO MODES PARAN ZAAQU ESION DEMAN UNTIS PERYL ABAYC OMOEL
OODEL ESUBA MUYAN STRUN EIANO ELERA CONEB RARAC OPADO LORIN
STAMP UESTE DIRQU ENCIA DOSUR DEREA SINTE LOARA AUERD OYTOMO
```

Im nčhsten Schritt wurde angenommen, dass die Nachricht der Challenge der in der von Sanguino verfassten Arbeit entschlüsselten Arbeit entsprach. Daher wurde der Beginn der Nachricht auf den Text „El Ministro Marina y Aire“ festgelegt. Ausgehend von dieser Vorgabe wurde anschließend der Hillclimbing Algorithmus mit 5000 Neustarts angewandt. Auch dieser Versuch führte jedoch nicht zu einem Erfolg. Zwar konnte ein besser bewerteter Schlüssel gefunden werden, dessen Kostenwert sich auf -3133 belief. Jedoch konnten auch in dem so gefundenen besten Klartext keine Anhaltspunkte gefunden werden, die zu einer kompletten Lösung der Chiffre hätten führen können. Der beste gefundene Klartext lautete wie folgt:

```
ELMIN ISTRO MARIN AYAIR EWDOO HIJPL TEKAM IERFV XABBO OQWAB
ASPKA MINWN PVFAI VEAJE OHTAM PPKER ELPIL UWOTF AFAMR IFAWV
IANOB PNNEE QEQVL STOTI RIAMI OFPFA OARWW OAMER IBANA BAOTI
NPVMU ERHJO PCODJ PRRBE DALEF PWMOA LILEP FIEOV AERON ATTIKA
```

Es gibt mehrere mögliche Gründe, warum der Angriff nicht erfolgreich war. Einer ist, dass der Text zu kurz ist. Die Evaluation zeigte, dass die Erfolgswahrscheinlichkeit des Angriffs bei Geheimtexten mit 200 Zeichen 0% beträgt. Dies gilt sowohl für den reinen Hillclimbing Algorithmus als auch für den Wörterbuchangriff. Auch Texte der Länge 250 konnten von dem Algorithmus nicht gelöst werden, somit ist nicht zu erwarten dass diese Challenge durch den entwickelten Algorithmus gelöst werden kann. Eine Chance besteht jedoch falls dem Nutzer Textteile auffallen, die richtig entdeckt wurden, und diese anschließend manuell festgelegt werden. Dadurch ist es möglich, nach und nach immer mehr Teile des Textes zu entschlüsseln und somit die Chiffre zu brechen. Allerdings müssen bereits am Anfang lange Textstücke erkannt werden, da auch der Wörterbuchangriff in der Evaluation bei dieser Textlänge keinen Erfolg hatte.
Da die Textlänge ein Problem darstellt erscheint es verwunderlich, warum die Challenge bei MysteryTwisterC3 als Level eins Challenge geführt ist. Allerdings findet sich in der Aufgabenstellung die Information, dass die Homophone beim Verschlüsseln zyklisch genutzt wurden. Dies wiederum ermöglicht statistische Analysen der Chiffre, die in der Arbeit von Sanguino näher erklärt sind. Diese wiederum vereinfachen das Lösen der Chiffre signifikant. Ein anderer möglicher Grund ist, dass die spanische Sprachstatistik nicht gut genug ist, echte spanische Texte von generischen zu unterscheiden. Die Qualität der spanischen Statistiken wird später in diesem Kapitel evaluiert.

### 6.3.2 Spanische Streifenchiffre 2

Die zweite Challenge zur Spanischen Streifenchiffre ist die Verschlüsselung eines englischen Telegramms das im spanischen Bürgerkrieg verschickt wurde. Jedem Klartextbuchstaben wurden im Schlüssel entweder drei oder vier Homophone zugeordnet. Der Geheimtext ist 326 Zeichen lang und besteht aus 73 verschiedenen Homophonen. Da die Lösung der Challenge bereits bekannt war konnte der Wörterbuchangriff vereinfacht werden, indem ein verkürztes Wörterbuch verwendet wurde, das nur Wörter enthielt, von denen bekannt war, dass sie in der Lösung vorkommen. Dieses Wörterbuch enthielt 39 Einträge. Jeder Eintrag wurde in einem Block über den Text geschoben und an jeder passenden Position der Hillclimbing Algorithmus mit fünf Neustarts durchgeführt. In der so entstandenen Bestenliste war erkennbar, dass viele gut bewertete Schlüssel zu einem Klartext führten, der mit

```
SHENA BYAND AIRMI NISTE R
```

oder einem ähnlichen Text begann. Dadurch ließ sich annehmen, dass der Klartext mit

```
THENA VYAND AIRMI NISTE R
```

beginnt. Dies schien auch im Kontext des Telegramms, des spanischen Bürgerkrieges, sinnvoll zu sein. Im nächsten Schritt wurde der Algorithmus angepasst, sodass der Beginn der Nachricht festgelegt wurde und auf dem so entstandenen Schlüssel ein Hillclimbing Algorithmus mit 30.000 Neustarts gestartet wurde. Dies wurde auf 30 Blöcke mit jeweils 1000 Neustarts verteilt. Aus der so entstandenen Bestenliste ließen sich neue Teile des Klartextes erkennen. An vielen Stellen war der Text bereits gut lesbar, sodass zwei große Teile des Klartextes festgelegt werden konnten. Der Text begann augenscheinlich mit

```
THENA VYAND AIRMI NISTE
RTOPR ESIDE NTOFT HE
```

und wenige Zeichen später war die Zeichenfolge

```
GOVER NMENT WEARE CONSI
DERIN GTHEI DEATO ESTAB
LISH
```

erkennbar. Diese beiden Textstellen wurden im nächsten Schritt festgelegt. Anschließend wurde der Hillclimbing Algorithmus mit 16.000 Neustarts, die auf 16 Blöcke mit jeweils 1000 Neustarts aufgeteilt wurden, gestartet. Aus der Bestenliste ließen sich anschließend einige Schlüssel entnehmen, die zu einem sehr gut lesbaren Klartext führten. An einigen Stellen war anschließend noch Handarbeit nötig, da

der Algorithmus zum Beispiel *BASQUE* nicht korrekt entschlüsselte sondern zum Beispiel *BASAVE* an diese Stelle setzte. Jedoch ließen sich diese Stellen aus dem Kontext erschließen, wodurch die Chiffre komplett entschlüsselt werden konnte. Der Klartext lautete

```
THE NAVY AND AIR MINISTER TO PRESIDENT OF THE BASQUE
GOVERNMENT WE ARE CONSIDERING THE IDEA TO ESTABLISH
IN BILBAO AN AIRCRAFT FACTORY WHERE THE MACHINE MARTIN
BOMBERS AND THE FOKKER CAN BE BUILT WELL I REQUEST
YOU TO CALL THE AERONAUTICAL ENGINEER IN LAMIACOCERRO
SERVICE AND EXPLAINING HIM THIS IDEA GIVEN THE
CONDITIONS LET US KNOW THE KIND OF MATERIALS THAT
WILL BE EMPLOYED IN BOTH MODELS
```

und ließ sich mit Hilfe des entwickelten Algorithmus erfolgreich entschlüsseln.

### 6.3.3 Spanische Streifenchiffre 3

Die dritte Challenge besteht aus einem Verschlüsselten Telegramm aus dem spanischen Bürgerkrieg, das bis heute nicht gebrochen werden konnte. Die Nachricht ist 142 Zeichen lang und besteht aus 55 verschiedenen Homophonen. Diese Challenge ließ sich ebenfalls nicht lösen. Dies ist nicht verwunderlich, da die Nachricht wie auch die erste ungelöste Challenge wahrscheinlich auf spanisch verfasst ist und zusätzlich noch kürzer als die erste Challenge ist. In den zu den besten gefundenen Schlüsseln gehörenden Klartexten konnten wie erwartet keine sinnvollen Satzteile gefunden werden.

### 6.3.4 Zodiac Challenge MysteryTwisterC3

Die veröffentlichte Challenge zum Zodiac Killer ist eine 340 Zeichen langer Geheimtext der aus 65 verschiedenen Homophonen besteht. Bei der Zodiac Chiffre ist allerdings nicht bekannt, wie viele Homophone einem Buchstaben zugeordnet sind. Daher können bei einem Angriff die Einschränkungen der spanischen Streifenchiffre nicht genutzt werden. Für den Angriff wurde zunächst ein Wörterbuch genutzt das 15 in der originalen Zodiac-408 Nachricht vorkommende Phrasen enthält. Die Länge der Phrasen beträgt zwischen vier und zwölf Buchstaben. Jeder Eintrag wurde einmal über den Text geschoben und die so erhaltenen Schlüsselteile dem Hillclimbing Algorithmus mit fünf Neustarts übergeben. Bereits bei diesem Angriff war anhand der erhaltenen Bestenliste bemerkbar, dass der erarbeitete Algorithmus nicht optimal für allgemeine homophone Chiffren ist. Zwar konnte durch den Angriff erkannt werden, dass der Klartext mit

```
I LIKE KILLING
```

beginnt. Jedoch war bemerkbar, dass der Hillclimbing Algorithmus an stellen, an denen der Schlüssel nicht durch das getestete Wort festgelegt war, den Klartext der Sprachstatistik anpasste. Dies ist möglich, da sich durch die Vielzahl der verwendeten Homophone an vielen Stellen im Text häufig in der natürlichen Sprache vorkommende Tri- und Tetragramme erzeugen lassen ohne an anderen Stellen sehr unwahrscheinliche Textabschnitte entstehen zu lassen. So war auffällig, dass das Wort *the* sehr häufig in den gefundene Klartexten vorkam. Da der Inhalt des Klartextes teilweise bekannt war konnte der Angriff durch das Festlegen einer Phrase im Text weiter getestet werden. Jedoch ließen sich auch durch das Festlegen der Phrase *ILIKEKILLING* am Anfang des Textes und die Durchführung des Hillclimbing Algorithmus mit 10.000 Neustarts keine deutlich besseren Ergebnisse finden. An dieser Stelle müssten vermutlich sowohl der Hillcimbing Algorithmus als auch die Berechnung der Kostenfunktion optimiert werden. Diese scheinen im Fall von allgemeinen homophonen Chiffren ohne Vorgaben zur Verwendung von Homophonen nicht optimal zu funktionieren. Mögliche Verbesserungen werden in Kapitel 8 erwähnt.

### 6.3.5 Original Zodiac-340 Nachricht

Die originale Zodiac-340 Nachricht ist 340 Zeichen lang und enthält 63 verschiedene Homophone. Da der Inhalt des Klartextes dieser Nachricht nicht bekannt ist wurde zum Testen das gleiche Wörterbuch wie für die Zodiac Challenge bei MysteryTwisterC3 verwendet. Erwartungsgemäß konnte mit diesem Wörterbuch kein sinnvolles Ergebnis gefunden werden. Da bereits aus den Angriffsversuchen auf die Challenge die Erfahrung gewonnen wurde, dass der entwickelte Algorithmus nicht gut mit einschränkungsfreien homophonen Chiffren funktioniert wurde nicht weiter versucht diese Nachricht zu brechen. Bevor dies mit einer Aussicht auf Erfolg möglich ist sollte der Algorithmus erst soweit weiterentwickelt werden dass es mit ihm möglich ist, die originale Zodiac-408 Nachricht und die Challenge zur Zodiac Chiffre zu lösen. Dies konnte im Rahmen dieser Arbeit jedoch nicht erreicht werden. Angriffe auf die Nachrichten Zodiac-13 und Zodiac-32 wurden aus diesem Grund nicht durchgeführt.

## 6.4 Diskussion der Ergebnisse

Die Evaluation zeigt, dass der Algorithmus nicht geeignet ist, Nachrichten die kürzer als 450 Zeichen sind komplett eigenständig zu brechen. Möglicherweise erhöht sich die Erfolgswahrscheinlichkeit des Algorithmus, wenn mehr als ein Wort gleichzeitig durch den Wörterbuchangriff getestet wird. Allerdings erhöht sich die Laufzeit des Algorithmus dadurch ebenfalls deutlich. In der Evaluation zeigte sich, dass das Testen eines zehn Buchstaben langen Wortes in einem 350 Zeichen langen

Geheimtext etwa eine Stunde dauert. Dieses Wort muss an ungefähr 340 verschiedenen Stellen im Geheimtext getestet werden. Will man nun zwei Wörter gleichzeitig testen, so gibt es für zwei jeweils zehn Buchstaben lange Wörter circa

$$340 \cdot 330 \approx 100.000 \tag{6.1}$$

verschiedene Möglichkeiten, diese im Text zu platzieren. Geht man nun davon aus, dass die Laufzeit des Hillclimbing Algorithmus in etwa gleich bleibt, so erhöht sich die Laufzeit des Wörterbuchangriffs ungefähr um den Faktor 300. Statt einer Stunde dauert das Testen von zwei Wörtern voraussichtlich also circa 300 Stunden. Dieser Wert konnte durch Tests nicht bestätigt werden. Allerdings fiel auf, dass die Laufzeiten je nach Wortpaar sehr unterschiedlich waren. Dies kann zwei Gründe haben.

- Die reale Anzahl an Möglichkeiten, die beiden Worte im Text zu platzieren. Gerade bei langen Worten oder Worten mit seltenen Buchstaben kann es passieren, dass sich viele theoretisch mögliche Kombinationen der Wörter im Text als unzulässig herausstellen, da durch diese Positionierung ein Homophon auf zwei verschiedene Klartextbuchstaben abgebildet werden müsste.
- Die Laufzeit des Hillclimbing Algorithmus sinkt. Je mehr Teile des Schlüssels durch die Wörter festgelegt sind umso weniger Möglichkeiten hat der Hillclimbing Algorithmus, den Schlüssel zu verändern. Dadurch findet der Algorithmus schneller ein Maximum, die Laufzeit sinkt.

Auch wenn die Laufzeit mit mehreren Wörter unter 300 Stunden lag war sie zu lang um genug Daten für eine Evaluation sammeln zu können.
Der Algorithmus ist zwar nicht geeignet, kurze Chiffren selbstständig zu lösen, jedoch konnte zumindest eine Chiffre geknackt werden. Die zweite Challenge zur spanischen Streifenchiffre konnte erfolgreich gelöst werden. Somit wurde gezeigt, dass der entwickelte Algorithmus prinzipiell geeignet ist, um mit der spanischen Streifenchiffre verschlüsselte Texte anzugreifen. Nicht erfolgreich waren die Angriffe auf spanische Texte. Der Hauptgrund für diesen Misserfolg scheint die schlechte Qualität der generierten spanischen Statistiken zu sein.
Nachdem keine guten Ergebnisse für die erste Challenge zur Spanischen Streifenchiffre gefunden werden konnten wurde die Kostenfunktion für spanische Texte evaluiert. Dafür wurde diese auf vier verschiedene Texte angewendet. Als Vergleichswerte wurden zwei Texte aus zufälligen Artikeln der spanischen Wikipedia genutzt. Außerdem wurden ein komplett zufällig generierter Text verwendet. Als Vergleichswert wurde ein Text, der von dem Algorithmus beim Versuch die erste Challenge zur Spanischen Streifenchiffre zu lösen gut bewertet wurde, genutzt. Dieser Text war der von der Kostenfunktion im Verlauf des Angriffs am besten bewertete gefundene Klartext. Zuvor wurde dieser analysiert und festgestellt, dass zwar einige spanische Worte wie „Presidenta“ oder sogar Wortfolgen wie „como des para“ oder „no el era con“ enthielt, diese jedoch keinen sinnvollen Zusammenhang ergaben und zwischen den Wörtern oder Wortreihen kein sinnvoller Text zu finden war. So bedeuten „como des para“ zum Beispiel frei übersetzt „wie des wegen“ und „no el era con“ in etwa „nicht er war mit“. Entsprechend wurde erwartet,

dass dieser Text von der Kostenfunktion besser als der komplett zufällig generierte, aber schlechter als die echten spanischen Texte bewertet werden sollte. Die Länge der Texte war 201 Buchstaben was der Länge der ersten Challenge entspricht. Alle Texte wurden zuerst nur mit Tetragrammen und anschließend mit Tri- und Tetragrammen bewertet. Wie in 6.1 zu sehen ist werden die beiden Spanischen Texte

| | Spanisch 1 | Spanisch 2 | Generiert | Hillclimbing |
|---|---|---|---|---|
| Tetragramme | -1882 | -1872 | -3847 | -1931 |
| Tri- und Tetragramme | -3364 | -3366 | -6854 | -3392 |

Tabelle 6.1: Kostenwerte der Texte

aus Wikipedia sehr ähnlich bewertet. Während der dritte, zufällig generierte Text, deutlich schlechter bewertet wird, vermag die Kostenfunktion kaum zwischen einem echten spanischen Text und dem vom Hillclimbing Algorithmus gefundenen, nicht sinnvollen Text, zu unterscheiden. Daher ist anzunehmen, dass für eine Verbesserung des Hillclimbing Algorithmus eine Verbesserung der spanischen Statistik erforderlich ist, sodass diese echte Texte zuverlässig besser bewerten kann als Texte, die nur an manchen Stellen spanische Worte enthalten.
Auch für allgemeine homophone Chiffren ohne die einschränkenden Regeln der spanischen Streifenchiffre scheint der Algorithmus nicht optimal geeignet zu sein. Daher wurde die Evaluation auch mit Geheimtexten die nach den Regeln der spanischen Streifenchiffre verschlüsselt wurden durchgeführt.

# 7 Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten vorgestellt. Auch wenn besonders zu den Nachrichten des Zodiac Killers viele Theorien existieren gibt es wenige wissenschaftliche Arbeiten zu homophonen Chiffren. In diesem Kapitel werden drei Arbeiten vorgestellt, die sich mit homophonen Chiffren befassen. Zwei dieser Arbeiten legen einen Schwerpunkt auf die Nachrichten des Zodiac während sich eine exklusiv mit der spanischen Streifenchiffre beschäftigt.

## 7.1 Efficient Cryptanalysis of Homophonic Substitution Ciphers

Diese Publikation von Amrapali Dhavare, Richard M. Low und Mark Stamp [DAS13] beschäftigt sich mit homophonen Substitutionschiffren und Angriffsmöglichkeiten auf diese im Allgemeinen. Dabei wird besonders auf Hillclimbing Algorithmen eingegangen, die sowohl für normale Substitutionschiffren als auch für homophone Verschlüsselungen beleuchtet werden. Die Autoren der Arbeit entwerfen darin einen Hillclimbing Algorithmus, der im Anschluss sowohl auf die unbekannte Zodiac-340 Nachricht als auch auf die auf MysteryTwisterC3 gestellte Zodiac Challenge angewandt wird. Der entwickelte Algorithmus funktioniert in zwei Schritten. Im ersten Schritt wird eine Vorgabe generiert, wieviele Homophone jedem Klartextbuchstaben zugeordnet werden sollen.
Unter Beachtung dieser Vorgabe werden im zweiten Schritt zufällige Startschlüssel generiert. Mit diesen wird anschließend ein Hillclimbing Algorithmus gestartet. In diesem werden die Bedeutungen von Homophonen getauscht. Dabei durchläuft eine Schleife alle Homophone und tauscht die Bedeutung jedes Homophons mit jedem anderen Homophon, das nicht vorher schon die gleiche Klartextabbildung beinhaltete. Sobald durch einen solchen Tausch ein besserer Schlüssel gefunden wurde startet der Algorithmus von vorne, es werden also nicht zwingend in jedem Durchgang alle Homophone getauscht. Innerhalb des Hillclimbing Algorithmus wird die Bedingung, welchem Klartextbuchstaben wieviele Homophone zugeordnet werden, nicht verändert.
Zur Berechnung des Schlüssels werden zwei Matritzen benutzt. Eine beinhaltet eine Statistik zum Vorkommen von Bigrammen in der natürlichen Sprache, die Andere die Statistik der Bigramme des zu einem Schlüssel gehörenden Klartextes. Der Schlüssel wird bewertet, indem der Unterschied zwischen den beiden Matritzen berechnet wird. Je kleiner der Unterschied ist, umso besser ist der Schlüssel.
Die Autoren der Arbeit versuchten, mit Hilfe des Algorithmus die unbekannte

Zodiac-340 Nachricht zu brechen. Dazu wurden sowohl nur die zweite Phase als beide Phasen zusammen getestet. Beide Angriffe brachten keine brauchbaren Ergebnisse. Im Gegensatz dazu gelang es ihnen, die MysteryTwisterC3 Challenge zum Zodiac zu brechen. Allerdings gelang dies nicht mit normalen englischen Statistiken, sondern lediglich mit der Benutzung einer Sprachstatistik, die aus der bekannten Zodiac-408 Nachricht generiert wurde. Während die Erfolgswahrscheinlichkeit mit dieser Statistik über 70% lag, fand der Algorithmus mit einer englischen Statistik nur in weniger als 15% der Fälle den richtigen Schlüssel.

## 7.2 Bayesian Inference for Zodiac and Other Homophonic Ciphers

Das von Sujith Ravi und Kevin Knight [RK11] verfasste Paper beschäftigt sich mit einem anderen möglichen Angriff auf homophone Chiffren. Dieser Angriff arbeitet mit der Bayesschen Statistik und verwendet Informationen aus Wörterbüchern sowie Sprachstatistiken. Zum Entschlüsseln der Nachricht generiert dieser Algorithmus zuerst ein statistisches Modell für die Klartextsprache. Anschließend wird mit einer bestimmten Wahrscheinlichkeit $P_0(p_1)$ ein Klartextbuchstabe $p_1$ generiert. Mit der Wahrscheinlichkeit $P_0(c_1|p_1)$ wird nun dieser Klartextbuchstabe $p_1$ dem Homophon $c_1$ zugewiesen. In den nächsten Schritten werden weiterhin Klartextbuchstaben generiert, die Geheimtextbuchstaben zugewiesen werden. Die jeweiligen Wahrscheinlichkeiten hängen davon ab, wie oft dieser Schritt bereits vom Algorithmus durchgeführt wurde. Nach jedem Durchgang kann der Algorithmus entweder abbrechen oder mit einer bestimmten Wahrscheinlichkeit die letzten beiden Schritte wiederholen.
Mit dieser Methode und den Daten eines Wörterbuches sowie der Statistik von Trigrammen gelang es den Autoren, die Zodiac-408 Nachricht zu mehr als 97% vollautomatisiert zu brechen, was nach ihren Angaben niemandem zuvor gelungen ist. Jedoch konnten auch sie die Zodiac-340 Nachricht nicht brechen.

## 7.3 Spanische Streifenchiffre

Das von Luis Alberto Sanguino und anderen verfasste Paper [LABS16] „Analyzing the Spanish strip cipher by combining combinatorial and statistical methods“ beschäftigt sich ausschließlich mit der Spanischen Streifenchiffre. Der von den Autoren entwickelte Angriff besteht aus drei Teilen. In einem ersten Schritt wird versucht, Homophone die auf den gleichen Klartextbuchstaben abbilden, zu finden. Dies geschieht unter der Annahme, dass die Homophone beim Verschlüsseln sequentiell verwendet werden. Dadurch wird es bei ausreichend langen Texten möglich, die Homophone zwischen den Vorkommen eines Homophons zu analysieren und Überschneidungen zu finden. Dadurch lassen sich Mengen von Homoponen

finden, die möglicherweise auf denselben Klartextbuchstaben abbilden. Im zweiten Schritt wird eine statistische Analyse der Buchstaben durchgeführt. Dafür werden alle Homophone, die auf denselben Klartextbuchstaben abbilden zusammengefasst und ihre Häufigkeit im Geheimtext analysiert. Dabei wird versucht, die in Schritt Eins erstellten Mengen von Homophonen Klartextbuchstaben zuzuordnen. Im letzten Schritt wird ein Wörterbuch genutzt. Dafür wird zuerst der Geheimtext mit dem in Schritt Zwei angenommenen Schlüssel dechiffriert. Dieser so entstandene Geheimtext wird nun aufgeteilt und geprüft, ob sich Teile des Klartextes in einem Wörterbuch finden lassen.
Mit diesem Angriff war es den Autoren möglich, originale Telegramme aus dem spanischen Bürgerkrieg zu entschlüsseln.

# 8 Ausblick und Zusammenfassung

In diesem Kapitel soll rückblickend zusammengefasst werden, ob die gesteckten Ziele erreicht werden konnten. Weiterhin sollen Ideen zu möglichen zukünftigen Verbesserungen dargelegt werden, die im Rahmen dieser Arbeit nicht mehr implementiert werden konnten. Jedoch könnten diese Verbesserungen zukünftig umgesetzt werden um das Produkt der Arbeit weiter zu verbessern.

## 8.1 Zusammenfassung

In diesem Abschnitt wird darauf eingegangen, ob die in der Einleitung genannten Ziele erreicht werden konnten und welche Probleme ein Erreichen der Ziele verhinderten. Dabei soll bewertet werden, ob die an die Arbeit gestellten Anforderungen erfüllt werden konnten.

### 8.1.1 (R1) Analyse, Konzeption, Entwicklung und Implementierung

Das erste Ziel der Arbeit bestand darin, homophone Chiffren zu analysieren und ein Kryptoanalysewerkzeug zu entwickeln, einen Hillclimbing Algorithmus sowie ein Wörterbuch benutzt um homophone Chiffren anzugreifen. Eine ausführliche Analyse homophoner Chiffren im Allgemeinen und besonders der Nachrichten des Zodiac Killers und der spanischen Streifenchiffre ist in Kapitel 3 zu finden. Weiterhin konnte eine Kryptoanalysekomponente entwickelt werden, die Angriffe auf homophone Chiffren ausführen kann. Diese Komponente ist im Quellcode einfach anpassbar, sodass sie sowohl auf allgemeine homophone Chiffren als auch auf die spanische Streifenchiffre mit ihren Besonderheiten anwendbar ist. Weiterhin ist in dieser Komponente einstellbar, ob ein reiner Hillclimbing Algorithmus verwendet werden oder dieser durch einen vorher durchgeführten Wörterbuchangriff unterstützt werden soll. Auch die Anzahl der Wörter und die Wortlängen sind einstellbar. Die Anforderung **R1** konnte also erfolgreich erfüllt werden.

### 8.1.2 (R2) Lösen der MysteryTwisterC3 (Übungs)challenges

Das zweite Ziel beinhaltete die Lösung der bei MysteryTwisterC3 verfügbaren Challenges zur spanischen Streifenchiffre und der Zodiac Nachricht. Von den drei

Challenges zur spanischen Streifenchiffre konnte nur eine gelöst werden. Während die zweite Challenge zur spanischen Streifenchiffre in mehreren Schritten gelöst wurde, konnten weder die erste Level 1 Challenge noch die dritte Level X Challenge erfolgreich mit dem Algorithmus gelöst werden. Auch die bei MysteryTwister veröffentlichte Challenge zur Zodiac Chiffre konnte nicht gelöst werden. Während bei den beiden ungelösten Challenges zur spanischen Streifenchiffre wahrscheinlich die Qualität der spanischen Sprachstatistik eine Rolle spielte, so ist dies kein Argument, warum die Zodiac Challenge nicht gelöst werden konnte. Die englischen Statistiken sind von guter Qualität und das Lösen der zweiten Challenge zur spanischen Streifenchiffre war mit dieser Statistik möglich. Hier ist möglicherweise eine Optimierung des Hillclimbing Algorithmus nötig. So konnte mit dem Wörterbuchangriff herausgefunden werden, dass die Nachricht mit „I like killing“ beginnt, weitere sinnvolle Stellen konnten im Anschluss jedoch nicht gefunden werden. Die Anforderung **R2** konnte nur teilweise erfüllt werden.

### 8.1.3 (R3) Analyse und (ggf) Lösung der noch offenen Zodiac-Chiffren sowie Spanish-Strip-Chiffren

Das letzte Ziel der Arbeit bestand daraus, die noch nicht entschlüsselten Nachrichten des Zodiac Killers sowie originale mit der spanischen Streifenchiffre verschlüsselte Nachrichten zu analysieren und, wenn möglich, zu lösen. Eine Analyse der ungelösten Nachrichten des Zodiac Killers ist in Kapitel 3 zu finden. Die spanische Streifenchiffre wurde im selben Kapitel ebenfalls ausführlich untersucht. Da die Regeln dieser Chiffre relativ klar festgelegt ist eine Analyse der einzelnen Nachrichten nicht nötig, da der einzige ersichtliche Unterschied die Länge dieser ist. Leider konnten weder die Nachrichten des Zodiac Killers noch originale Nachrichten aus dem spanischen Bürgerkrieg entschlüsselt werden. Wie in Kapitel 4 beschrieben ist für einen erfolgversprechenden Angriff mit einem kompletten Wörterbuch deutlich mehr Rechenleistung nötig. Für originale Nachrichten aus dem spanischen Bürgerkrieg ist weiterhin wahrscheinlich eine bessere Sprachstatistik notwendig. Für die Nachrichten des Zodiac Killers muss, wie in Anforderung R2 beschrieben, wahrscheinlich der Hillclimbing Algorithmus an die Freiheiten allgemeiner homophoner Chiffren im Vergleich zur spanischen Streifenchiffre angepasst werden. Die Anforderung **R3** konnte nicht erfüllt werden. Allerdings wurde bereits in der Aufgabenstellung erwähnt dass ein Erfüllen dieser Anforderung unwahrscheinlich und daher kein notwendiges Ziel der Arbeit ist.

## 8.2 Ausblick

Während der Anfertigung der Arbeit ergaben sich einige Möglichkeiten, den implementierten Algorithmus zukünftig zu verbessern oder besser anzuwenden. Diese Möglichkeiten sollen im folgenden Abschnitt dargelegt werden.

### 8.2.1 Bessere Sprachstatistiken

Ein Problem mit den spanischen Telegrammen war möglicherweise die Statistik. Diese wurde ausschließlich aus Büchern der Gutenberg Bibliothek generiert. Verglichen mit der genutzten englischen Statistik lag dieser vergleichsweise wenig Text zugrunde. Eine Möglichkeit wäre es, zusätzlich alle Artikel der spanischen Wikipedia zur Generierung der Statistik zu nutzen. Auch online Archive von Zeitungen wären eine Möglichkeit an mehr spanischen Text zu kommen.

### 8.2.2 Verbesserung der Kostenfunktion

Ein Problem bei dem Angriff auf die Zodiac Challenge war das Verhalten des Hillclimbing Algorithmus. Dies lag daran, dass es für die Nachrichten des Zodiac Killers im Gegensatz zur spanischen Streifenchiffre keine Einschränkung der Nutzung der Homophone gibt. Die MysteryTwisterC3 Aufgabe ist zum Beispiel 340 Zeichen lang und besteht aus 65 verschiedenen Homophonen. Das bedeutet dass jedes Homophon im Schnitt nur circa fünfmal im Text vorkommt. Dadurch ist es möglich, Schlüssel zu finden die den Homophonen Bedeutungen zuweisen die dazu führen dass an einigen Stellen im Text sehr wahrscheinliche Buchstabenkombinationen wie *the* oder *and* vorkommen ohne dass dadurch an anderen Stellen des Textes sehr unwahrscheinliche Buchstabenkombinationen entstehen. Eine Möglichkeit, dies zu verhindern, ist die stärkere Gewichtung der Buchstabenhäufigkeit. Tri- und Tetragramme bewerten zwar das Vorkommen von Wörtern wie *the* und *and* sehr hoch, jedoch könnte dies ausgeglichen werden wenn die Buchstaben „e“ und „t“ häufiger als erwartet vorkommen und dies die Bewertung durch die Kostenfunktion wieder verschlechtert.

### 8.2.3 Nutzung von mehr Rechenleistung

Wie bereits in Kapitel 4 berechnet reicht die zur Verfügung stehende Rechenleistung nicht aus um komplette Wörterbücher in einer annehmbaren Zeit über die Texte zu schieben. Die nötige Rechenleistung für einen Angriff mit einem kompletten Wörterbuch wird in absehbarer Zeit nicht durch die Nutzung der VoluntCloud zu erreichen sein. Eine Alternative wäre es, ein BOINC Projekt [boi] zu erstellen. Im Gegensatz zur VoluntCloud ist BOINC eine ausgereifte Software die schon seit über 10 Jahren existiert und von vielen Projekten genutzt wird. Entsprechend ist die Nutzerbasis von BOINC deutlich größer wodurch zumindest theoretisch deutlich mehr Rechenleistung zur Verfügung steht. Zwar müssten in diesem Fall die Nutzer von BOINC überzeugt werden, ihre Rechenzeit für das Projekt zur Verfügung zu stellen, jedoch sollten sich gerade für die Entschlüsselung der ungelösten Nachricht Zodiac-340 viele Freiwillige finden lassen, da diese sowohl einen kriminologischen als auch einen kryptologischen Reiz hat und somit wahrscheinlich das Interesse vieler Leute wecken kann.

# Literaturverzeichnis


[boi] *Berkeley Open Infrastructure for Network Computing*. https://boinc.berkeley.edu.

[cit] http://www.zodiackillerciphers.com/?p=224. Accessed: February 2nd, 2016.

[Col] Cole, Michael. http://zodiacrevisited.com/name-cipher-motivation/.

[DAS13] Dhavare Amrapali, Richard M. Low und Mark Stamp: *Efficient cryptanalysis of homophonic substitution ciphers*. Cryptologia, 37.3:250–281, 2013.

[emm] https://localwiki.org/sf/Robert_Emmet. Accessed: February 6th, 2016.

[Fri79] Friedman, William Frederick: *The index of coincidence and its applications in cryptography*. Aegean Park Press, 1979.

[Gün88] Günther, Christoph G.: *A Universal Algorithm for Homophonic Coding*, Seiten 405–414. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[gut] https://www.gutenberg.org.

[Hoe] Hoerenberg, Michael: *Breaking German Navy Ciphers*. https://enigma.hoerenberg.com/. Accessed: December 7th, 2016.

[KK] Kopal, Nils und Christopher Konze: *VoluntLib Middleware in the CrypTool 2 Repository*. https://www.cryptool.org/trac/CrypTool2/browser/trunk/LibSource/voluntLib. Accessed: August 29th, 2016.

[LABS16] Luis Alberto Benthin Sanguino, et al.: *Analyzing the Spanish strip cipher by combining combinatorial and statistical methods*. Cryptologia, 40.3:261–284, 2016.

[LKW14] Lasry, George, Nils Kopal und Arno Wacker: *Solving the Double Transposition Challenge with a Divide-and-Conquer Approach*. Cryptologia, 38(3):197–214, 2014.

[MTC] *MysteryTwisterC3 - the Crypto Challenge Contest*. https://www.mysterytwisterc3.org/.

[Nat] *Nachrichten aus dem baskischen Nationalarchiv.* `http://dokuklik.snae.org/badator_zoom.php?cdc=001&cdd=01033`. Accessed: November 17th, 2016.

[Ora] Oranchak, David. `http://zodiackillerciphers.com`.

[Ora15] Oranchak, David: *The Zodiac Ciphers - What Do We Know, and When Do We Stop Tying to Solve Them?* 2015 Cryptologic History Symposium, 2015.

[RK11] Ravi, Sujith und Kevin Knight: *Bayesian Inference for Zodiac and Other Homophonic Ciphers.* In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, Seiten 239–247, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[Sch12] Schmeh, Klaus: *Nicht zu knacken: Von ungelösten Enigma-Codes zu den Briefen des Zodiac-Killers.* Carl Hanser Verlag GmbH Co KG, 2012.

[spa] `https://github.com/titoBouzout/Dictionaries/blob/master/Spanish.dic`.

[Wac14] Wacker, Arno: *Introduction to applied Cryptology.* Vorlesung an der Universität Kassel, 2014.

[WCB12] William C. Barker, Elaine Barker: *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher.* `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-67r1.pdf`, 2012.

[zka] `https://web.archive.org/web/20110824185010/http://www.sanfranmag.com/story/zodiac-killer-who-will-not-die`. Accessed: February 4th, 2016.

# Versicherung an Eides Statt

Ich, Nils Rehwald, Matrikelnummer 31202848, wohnhaft in 34311 Naumburg, versichere an Eides Statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen übernommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe.

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

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

Naumburg, 27. März 2017 — Nils Rehwald