UNIVERSITÄT KASSEL
FACHBEREICH ELEKTROTECHNIK/INFORMATIK
FACHGEBIET ANGEWANDTE INFORMATIONSSICHERHEIT

**Bachelorarbeit**

# Kryptoanalyse der ADFGVX-Chiffre in CrypTool 2

Dominik Vogt
Matrikelnummer: 30210970

UNI KASSEL
VERSITÄT
Fachgebiet Angewandte Informationssicherheit
Fachbereich Elektrotechnik/Informatik
Universität Kassel

14. Oktober 2018

**Prüfer:**
Prof. Dr. Arno Wacker
Prof. Dr. Gerd Stumme
**Betreuer:**
Prof. Dr. Bernhard Esslinger
Dr. Nils Kopal

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

Algorithmen zur Verschlüsselung von Daten sind nicht erst heute ein wichtiges Mittel, um vertrauliche Informationen auszutauschen. Seit Jahrhunderten werden Nachrichten, vor allem in Kriegszeiten, verschlüsselt übermittelt – so auch während des Ersten Weltkrieges in Deutschland. Der Nachrichtenoffizier Fritz Nebel entwickelte die ADFGVX-Chiffre in zwei Schritten, um Funksprüche des Heeres zu verschlüsseln. Die erste Variante beinhaltete die Buchstaben ADFGX, später kam das V als sechster Buchstabe hinzu. Mitte der 60er Jahre kam heraus, dass noch während des Krieges die Nachrichten bereits zum Teil entschlüsselt werden konnten.[Wikb] Aber erst in den letzten paar Jahren konnte die ADFGVX-Chiffre vollständig gebrochen werden. Im Jahr 2016 veröffentlichte George Lasry ein neues Verfahren, das es ermöglichte, auch bis dahin unverschlüsselte Nachrichten zu entschlüsseln.

In dieser Bachelorarbeit wurde für die Open-Source-Software CrypTool 2, die eine Lehr- und Lernplattform für kryptografische und kryptoanalytische Algorithmen und Verfahren bietet, eine neue ADFGVX-Analyse-Komponente implementiert und darin die Verfahren von Lasry integriert. Außerdem wurde die bestehende ADFGVX-Chiffrier-Komponente, die Nachrichten mit der ADFGVX-Chiffre ver- und entschlüsseln kann, so erweitert, dass damit auch 7x7-Matrizen verarbeitet werden können. Darüber hinaus wird das implementierte Verfahren analysiert und die Mächtigkeit des Verfahrens evaluiert.

## 1.1 Motivation

Das Verschlüsselungsverfahren ADFGVX findet im heutigen Alltag keine Verwendung mehr. Dies liegt hauptsächlich daran, dass es aufgrund der verfügbaren Rechenleistung und den inzwischen entwickelten heuristischen Verfahren unsicher ist. Mit heuristischen Verfahren ist es möglich, sich an den korrekten Schlüssel anzunähern. Über die Güte eines Schlüssels entscheidet dabei eine Kostenfunktion. Meines Wissens gibt es kein klassisches Verfahren, das einem heuristischen Angriff standhalten kann – außer der Vernam-Chiffre.[Eil] Trotzdem kann die Implementierung und Diskussion der Kryptoanalyse von klassischen Verfahren wertvoll sein. Bezüglich ADFGVX sind vor allem Historiker daran interessiert, den Inhalt der Nachrichten aus dem ersten Weltkrieg zu entschlüsseln. Dadurch können neue Details zum Ablauf der Geschichte bekannt werden. Darüber hinaus können die Lösungsverfahren für solche Chiffre auch bei anderen gegenwärtigen Problemen hilfreich sein. Zum Beispiel werden heuristische Verfahren eingesetzt für die Routenplanung.[Pat, Sch] Um moderne Verschlüsselungsverfahren anzugreifen sind die heuristischen Verfahren allerdings nutzlos. Das liegt daran, dass man sich bei älteren Verfahren an den richtigen Schlüssel schrittweise annähern kann; bei heutigen Verschlüsselungsverfahren ist das nicht mehr möglich.

## 1.2 Aufgabenstellung

Die Aufgabenstellung aus dem Fachgebiet lautet:

*Die ADFGVX-Chiffre wurde von Fritz Nebel während des Ersten Weltkriegs entworfen und im Juni 1918 vom deutschen Militär eingeführt. Sie war die Erweiterung der ADFGX-Chiffre, welche drei Monate zuvor eingeführt worden war. Die ADFGX und die ADFGVX waren die am weitesten fortgeschrittenen Feldchiffren ihrer Zeit, da mit ihnen erstmals Fraktionierung, neben Substitution und Transposition, eingesetzt wurde. Die Kryptoanalyse der Chiffren gestaltete sich in der damaligen Zeit äußerst schwierig, dennoch konnte der Franzose Georges Painvin die Chiffre letztendlich doch brechen – zumindest in bestimmten, günstigen Fällen. Kurz nach dem Ersten Weltkrieg hat James Rives Childs eine bessere und allgemeine Methode für die Kryptoanalyse entwickelt. Marcel Givierge, der Leiter des französischen „Bureau du Chiffre“, entwickelte parallel zu Painvin eine ähnliche Methode in den frühen 20er Jahren. In heutiger Zeit wurden zusätzliche Methoden, basierend auf modernen Optimierungsalgorithmen, für die Kryptoanalyse von ADFGVX/ADFGX entwickelt. Allen voran die Methode von George Lasry, welche auf Hill-Climbing und speziellen Kostenfunktionen beruht. Mit dieser Methode konnte Lasry eine Vielzahl von originalen, vorher nicht vollständig entschlüsselten Nachrichten brechen.*

*CrypTool 2 – der Nachfolger der bekannten E-Learning-Anwendung für Kryptografie und Kryptoanalyse CrypTool 1 – ist ein Open-Source-Projekt, das Lernenden, Lehrenden und Entwicklern die Möglichkeiten bietet, selbst verschiedene kryptografische und kryptoanalytische Verfahren anzuwenden und auszuprobieren. Mit der modernen Benutzeroberfläche kann man intuitiv per Drag & Drop sowohl einfache als auch komplexe kryptografische Algorithmen erstellen. Der Benutzer kann dabei ohne besondere Programmierkenntnisse die Algorithmen miteinander verbinden und so eigene neue Algorithmen und Abläufe erschaffen und testen.*

*CrypTool 2 (CT2) basiert auf modernen Techniken wie dem .NET Framework (zurzeit 4.7.1) und der Windows Presentation Foundation (WPF). Darüber hinaus ist die Architektur von CrypTool 2 vollständig Plugin-basiert und modular aufgebaut, wodurch die Entwicklung neuer Funktionalitäten stark vereinfacht wird. Im Rahmen dieses Open-Source-Projekts wurden bereits eine Vielzahl von kryptografischen Algorithmen (wie z.B. AES, SHA1 oder Enigma) als Komponenten entwickelt.*

*Bezüglich ADFGX und ADFGVX sind in CrypTool 2 derzeit nur die Chiffren, also die Ver- und Entschlüsselung bei gegebenem Schlüssel, implementiert. Die Kryptoanalyse beschränkt sich auf einen einfachen Wörterbuchangriff. Von daher ist* *das Ziel dieser Bachelorarbeit die vollständige Implementierung der Kryptoanalyse der ADFGX- und ADFGVX-Chiffren innerhalb einer eigenen Kryptoanalyse-Komponente. Als Basis hierfür dient die Publikation von Lasry. Der Quellcode von Lasry, geschrieben in Java, kann auch genutzt werden. Die Analysekomponente soll sich in ihrem Design an bereits bestehenden Analysekomponenten orientieren (bspw. an der Vigenère-Kryptoanalyse). Neben der Implementierung der Analyse soll auch eine Evaluation der Mächtigkeit der Kryptoanalyse-Algorithmen durchgeführt werden. Des Weiteren sollen Vorlagen und Hilfen in deutsch und englisch in CT2 erstellt werden, welche die Thematik „ADFGX und ADFGVX“ beinhalten.*

*Teil der Bachelorarbeit ist außerdem eine Diskussion, welche Teile wie zur Sicherheit des Gesamtverfahrens beitragen. Dazu ist der Substitutionsschritt so flexibel zu implementieren, dass er nicht nur eine 5x5- (ADFGX) und eine 6x6- (ADFGVX)-Matrix unterstützt, sondern auch höhere Längen wie 7x7. Die unterstützten 49 Zeichen könnten bspw. folgende sein:*
"$ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.,:;?()-+><=$".

*Hier wäre dann der Sicherheitsunterschied zwischen ADFGX, ADFGVX und einer auf einer 7x7-Matrix basierenden Variante kurz zu diskutieren. Außerdem soll beim Transpositionsschritt ganz kurz diskutiert (nicht implementiert) werden, wie der Sicherheitsgewinn ist, wenn man statt einer einfachen Spaltentransposition eine doppelte verwendet.*

*Alle in der Umsetzung getroffenen Entscheidungen sind innerhalb der Bachelorarbeit zu diskutieren. Der entwickelte Code sowie weitere Artefakte (cwm-Dateien, Hilfe, etc.) sind in die Versionierungskontrolle (SVN) des CrypTool 2-Projekts einzupflegen.*

## 1.3 Ziele der Arbeit

Der Schwerpunkt dieser Bachelorarbeit liegt nicht in der ADFGVX-Chiffre selbst, sondern in der Umsetzung des von Lasry entwickelten Verfahrens. Daraus ergeben sich folgende Ziele, die an diese Arbeit gestellt und umgesetzt wurden:

Z1 Die Implementierung von Lasrys Quellcode in eine neue Analyse-Komponente für CrypTool 2

Z2 Die Implementierung der geforderten Erweiterungen in der bisherigen Chiffrier- und der neuen ADFGVX-Analyse-Komponente

Z3 Das Erstellen von Vorlagen und Hilfen in CrypTool 2

Z4 Die Evaluation der implementierten ADFGVX-Analyse-Komponente

## 1.4 Aufbau der Arbeit

Die vorliegende Bachelorarbeit ist in acht Kapitel unterteilt.

Auf das Einleitungskapitel folgt eine ausführliche Beschreibung der Grundlagen in Kapitel 2.

In Kapitel 3 werden Konzept und Design der neu implementierten Analyse-Komponente vorgestellt und begründet.

Kapitel 4 befasst sich mit der technischen Umsetzung der Komponente und einer Erklärung, wie diese Komponente und die zugehörigen Vorlagen zu benutzen sind.

In Kapitel 5, Evaluation der ADFGVX-Analyse-Komponente, befassen wir uns mit der Mächtigkeit des Verfahrens und wie gut dieses in CrypTool 2 umgesetzt wurde.

In Kapitel 6 wird das Verfahren von Lasry mit verwandten Arbeiten verglichen.

Im vorletzten Kapitel 7 wird diskutiert, welchen Sicherheitsgewinn eine Vergrößerung des Substitutions-Alphabets und die doppelte Spaltentransposition für die ADFGVX-Chiffre gebracht haben könnte.

Abschließend werden in Kapitel 8 die wichtigsten Ergebnisse zusammen gefasst, ein Fazit der Bachelorarbeit gezogen und ein Ausblick auf mögliche Verbesserungen an der implementierten Komponente gegeben.

# 2 Grundlagen

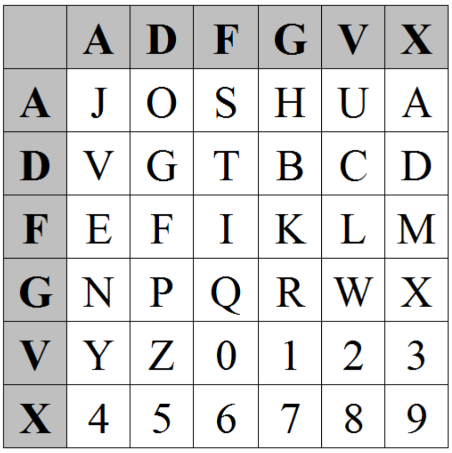
Die *ADFGVX* wurde von den Deutschen auch als *Geheimschrift der Funker 1918* bezeichnet. Während des ersten Weltkrieges wurde hauptsächlich die Telegrafie per Funk für die Kommunikation verwendet. Weil diese Nachrichten aber jeder mitlesen konnte, mussten die Informationen verschlüsselt werden, wozu eine Geheimschrift für den Funkverkehr benötigt wurde.[Ben15]

## 2.1 Geschichte

Im Jahr 1918 erfand der deutsche Nachrichtenoffizier Fritz Nebel die *ADFGX*-Chiffre, mit der die übermittelten Nachrichten nur noch aus den Zeichen A, D, F, G und X bestanden, da diese beim Funken am leichtesten voneinander unterschieden werden konnten. Durch das sogenannte Polybius-Quadrat (siehe Abb. 2.1) konnte man das gesamte Alphabet aus 26 Zeichen abbilden, indem man jedes Zeichen aus dem Klartext mit einer Kombination aus zwei der zur Verfügung stehenden Zeichen ersetzte. Anschließend wurde der Geheimtext mit einer Transposition noch auseinander gerissen. Eingesetzt wurde diese Variante ab dem 01. März 1918. Im April 1918 konnte der französische Offizier Georges Painvin die Verschlüsselung aber bereits brechen und die Nachrichten konnten teilweise mitgelesen werden.[Ben15]

Zum 01. Juni 1918 änderten die Deutschen ihr System und fügten mit dem *V* ein sechstes Zeichen ein. Somit konnte man 36 Zeichen in dem Polybius-Quadrat unterbringen, was dem Alphabet und den Ziffern 0-9 entspricht. Als die Franzosen die erste *ADFGVX*-Nachricht abhörten, konnte Painvin genau diese Tatsache richtig schlussfolgern und bereits nach einem Tag auch einige der neuen Nachrichten entschlüsseln.[Ben15]

Im Jahr 1966 erfuhr Fritz Nebel, dass Georges Painvin die von ihm erfundene Chiffre bereits während des Weltkrieges gebrochen hatte. Zwei Jahre später trafen dann beide aufeinander und Nebel erzählte, dass er ursprünglich eine doppelte Spaltentransposition vorgeschlagen hatte, diese jedoch von seinen Vorgesetzten abgelehnt wurde. Painvin ergänzte, dass er eine doppelte Spaltentransposition nicht hätte brechen können.



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

Abb. 2.1: Polybius-Quadrat einer ADFGVX-Chiffre mit dem Schlüsselwort Joshua Vogt

## 2.2 Das ADFGVX-Verfahren

Die *ADFGVX*-Chiffre ist eine monografische bipartite monoalphabetische Substitution mit einer anschließenden Transposition. Monografisch bedeutet, dass man jedes Zeichen aus einem Klartext einzeln betrachtet, und bipartite bedeutet, dass dieses dann durch zwei Geheimtextzeichen substituiert wird. Da nur ein einziges Alphabet benutzt wird ist es somit auch eine monoalphabetische Substitution. Danach erfolgt, mit Hilfe der Transposition, die Fraktionierung des bereits verschlüsselten Textes.

### 2.2.1 Die Substitution

Da nur fünf bzw. sechs Buchstaben zur Verfügung standen, wurde das Polybius-Quadrat verwendet. So konnten mit einen 5x5-Quadrat 25 Zeichen dargestellt werden. Für das 26 Zeichen lange Alphabet bedeutete dies, dass die Zeichen J und I gleich gesetzt wurden. Analog kann ein 6x6-Quadrat 36 verschiedene Zeichen darstellen, was zusätzlich die Verwendung der Ziffern 0-9 ermöglichte. Der Schlüssel bei dieser Substitution ist die Anordnung der Zeichen in dem Polybius-Quadrat, wobei jedes Zeichen genau einmal vorkommen muss. Bei einem Quadrat der Größe 5x5 beträgt der Schlüsselraum 25! ($\approx 1{,}55 \cdot 10^{25}$ oder $\approx 2^{87}$), bei 6x6 bereits 36! ($\approx 3{,}72 \cdot 10^{41}$ oder $\approx 2^{142}$).

### 2.2.2 Die Transposition

Zusätzlich wurde der Geheimtext noch mit Hilfe eines Transpositionsschlüssels fraktioniert. Dabei wurde der substituierte Text, Zeichen für Zeichen, von links

| k | 17 | 18 | 19 | 20 | 21 |
|---|---|---|---|---|---|
| Formel | $36! \cdot 17!$ | $36! \cdot 18!$ | $36! \cdot 19!$ | $36! \cdot 20!$ | $36! \cdot 21!$ |
| Schlüsselraum | $\approx 2^{186}$ | $\approx 2^{190}$ | $\approx 2^{195}$ | $\approx 2^{199}$ | $\approx 2^{204}$ |

Tab. 2.1: Schlüsselraum bei Länge k des Transpositionsschlüssel

nach rechts und von oben nach unten, unter den Transpositionsschlüssel geschrieben. Dadurch entstanden Zeilen mit gleicher Anzahl Zeichen. Die letzte Zeile ist in den meisten Fällen kürzer. Anschließend wurden die Spalten lexikografisch sortiert und spaltenweise ausgelesen, so dass die Bigramme auseinander gerissen werden. Der Schlüsselraum für die Transposition entspricht k!, wobei k die Länge des Transpositionsschlüssels darstellt. Bei $k = 13$ ergibt sich eine zweistufige Verschlüsselung mit einem Schlüsselraum von $25! \cdot 13! \approx 2^{119}$ oder $36! \cdot 13! \approx 2^{174}$.

In Tabelle 2.1 sieht man, wie sich der Schlüsselraum bei den Längen $17 - 21$ des Transpositionsschlüssels verändert (dies waren übliche Schlüssellängen im Ersten Weltkrieg). Zur Berechnung wurde eine 6x6-ADFGVX-Matrix ausgewählt.

### 2.2.3 Unizitätslänge

Neben der Schlüssellänge und der Entropie ist die Unizitätslänge [Sha49] ein weiteres Merkmal zur Beurteilung der Güte einer Chiffre. Diese gibt an, wie viele Zeichen ein Geheimtext mindestens haben muss, um daraus wieder einen eindeutigen Klartext zu generieren. Ein einfaches Beispiel hierfür ist die transponierte Zeichenfolge BEEINNNVTUSO, vermutlich ist das Wort SUBVENTIONEN verschlüsselt worden. Aber es könnten auch die Wörter VENUS IN BETON gewesen sein. Entscheidend für die Unizitätslänge ist die Größe des Schlüsselraums und welches Alphabet bzw. welche Sprache zum Verschlüsseln verwendet wurde. Weil die Größe des Schlüsselraums durch die Länge des Transpositionsschlüssels variabel ist, ist die Unizitätslänge ebenfalls keine Konstante.

Die Unizitätslänge $U$ berechnet sich aus: $U = H(ks)/D$, wobei $H(ks)$ die Entropie des Schlüsselraums $ks$ (119 bei $2^{119}$) und D die Klartext-Redundanz in Bits pro Zeichen ist.[MvOV96, Pom] Ein Alphabet mit 26 Buchstaben kann theoretisch $log_2(26) \approx 4,7$ Bits an Informationen übermitteln. Praktisch beträgt die durchschnittliche Menge tatsächlicher Informationen nur 1,5 Bits pro Zeichen in der englischen Sprache (1,4 deutsche Sprache). Die Klartext-Redundanz beträgt also $D = 4,7 - 1,5 = 3,2$ (Deutsch: $D = 4,7 - 1,4 = 3,3$).

Wird die englische Sprache und die *ADFGX*-Chiffre mit einem dreizehn Zeichen langen Transpositionsschlüssel verwendet, ergibt sich eine Unizitätslänge von circa 37 Zeichen: $U = 119/3,2$ (Deutsch: $U = 36 = 119/3,3$). Weitere Berechnungen zur Unizitätslänge sind in Tabelle 2.2 auf Seite 9 zu finden.

### 2.2.4 Vergleich von ADFGVX – Bifid – Playfair – Four-Square

Um diese Werte besser vergleichen zu können, werden andere Chiffren, die ebenfalls auf dem Polybius-Quadrat aufbauen, betrachtet. Dies kann unter [Pra] detaillierter nachgelesen werden.

*Bifid* basiert auf einer 5x5-Matrix mit den Zahlen eins bis fünf und ein Zeichen wird immer auf zwei Zahlen abgebildet, die untereinander aufgeschrieben werden. Anschließend wird zeilenweise ausgelesen. Diese Substitution, das 5x5-Polybius-Quadrat alleine, ist hier für die Berechnung des Schlüsselraums und der Unizitätslänge ausschlaggebend, da die Transposition ein festes Schema hat.

*Playfair* ist eine bigrafische, bipartite monoalphabetische Substitution, es werden also Zeichenpaare aus dem Klartext zu Zeichenpaaren in einen Geheimtext verschlüsselt. Dies erfolgt ebenfalls wieder über ein 5x5-Polybius-Quadrat, hier wird aber der Geheimtext nicht auf bestimmte Zeichen reduziert. Die Auswahl der zum Klartextzeichen gehörenden Geheimtextzeichen ist abhängig davon, ob das Paar in derselben Zeile, Spalte oder weder noch liegt. Trotz des verschiedenen Vorgehens ist es eine monoalphabetische Substitution.

*Four-Square* arbeitet ähnlich wie die Playfair-Chiffre, allerdings mit insgesamt vier 5x5-Polybius-Quadraten, wobei zwei davon mit idealerweise unterschiedlichen Passwörtern generiert werden und zwei die Klartext-Matrix beinhalten. Die Quadrate werden in einer 2x2-Matrix angeordnet, wo die Klartext- und Schlüssel-Matrizen sich diagonal gegenüber stehen. So kann über ein Zeichenpaar aus dem Klartext ein Quadrat über beide Klartext-Matrizen gespannt werden und die beiden Ecken in den Schlüssel-Matrizen ergeben das Zeichenpaar des Geheimtextes.

Tabelle 2.2 zeigt, dass die *ADFGX-Chiffre* mit einem dreizehn Zeichen langen Transpositionsschlüssel k einen größeren Schlüsselraum und eine längere Unizitätslänge besitzt als die Vergleichsverfahren Bifid und Playfair. Die Unizitätslänge der *Four-Square-Chiffre* ist fast identisch zur *ADFGVX-Chiffre.* Da die Größe des Schlüsselraums und die Unizitätslänge aber nur notwendige und keine hinreichenden Bedingungen sind, beschreibt das noch nicht die kryptografische Sicherheit (Gesamtsicherheit) des Verfahrens. Die *ADFGVX*-Chiffre ist allen deutlich überlegen, da die anderen, betrachteten Verfahren lediglich mit einer monoalphabetischen Substitution verschlüsselt werden. Die Transposition bei Bifid wird wegen des Kerckhoffs'schen Prinzips[1] nicht berücksichtigt.

---

[1] Die Sicherheit des Verfahrens sollte auf Geheimhaltung des Schlüssels beruhen, nicht auf Geheimhaltung des Verfahrens.

| Name | ADFGX | ADFGVX | Bifid | Playfair | FourSquare |
|---|---|---|---|---|---|
| Schlüsselraum | $25! \cdot k!$ | $36! \cdot k!$ | $25!$ | $25!$ | $(25!)^2$ |
| | $(k = 13) \approx 2^{119}$ | $(k = 13) \approx 2^{174}$ | $\approx 2^{84}$ | $\approx 2^{84}$ | $\approx 2^{167}$ |
| Unizitätslänge | $119/3,2$ | $174/3,2$ | $84/3,2$ | $84/3,2$ | $167/3,2$ |
| | (k=13) 37 | (k=13) 54 | 27 | 27 | 53 |

Tab. 2.2: Vergleich von Chiffren mit Polybius-Quadrat (englische Sprache)

## 2.3 Heuristische Verfahren

Heuristische Verfahren werden in der Kryptoanalyse eingesetzt, um nicht den gesamten Schlüsselraum durchsuchen zu müssen. Diese Verfahren nähern sich dem richtigen Schlüssel an.

Es gibt eine Vielzahl von heuristischen Verfahren – die wohl bekannteste Heuristik ist *trial and error*. Für die Umsetzung der Bachelorarbeit wird das heuristische Näherungsverfahren *Simulated Annealing* verwendet.

Beim Simulated Annealing darf sich das Zwischenergebnis auch verschlechtern, was dazu führt, dass das Verfahren ein lokales Maximum wieder verlassen kann, um später ein noch Besseres zu finden (siehe Abb. 2.2). Übertragen auf das Finden eines Schlüssels bedeutet das, dass die Nachbarschlüssel betrachtet werden und dann davon, mit einer gewissen Wahrscheinlichkeit, einer ausgewählt werden kann, der ein etwas schlechteres Ergebnis liefert.[Wikc]

## 2.4 Häufigkeitsverteilung

Die Häufigkeitsverteilung beschreibt die Gesamtheit der Häufigkeiten der einzelnen Buchstaben eines Textes im Verhältnis zum Gesamttext.[Krya] In der deutschen Sprache kommen die Buchstaben *E* und *N* mit Abstand am häufigsten vor – zusammen machen sie knapp 27% eines Textes aus.[Krya]

Für unsere Untersuchungen nutzten wir das Buch Pinocchio [Col] – hier exemplarisch der erste Abschnitt aus dem Buch:

```
Es war einmal ... Ein Koenig! - meinen gleich die klugen kleinen Leser.
Aber diesmal, Kinder, habt ihr weit daneben geraten. - Es war einmal:
ein Stueck Holz, ja, ein ganz gewoehnliches Holzscheit! Draußen lag es
im Wald mit vielen andern Stuecken auf der Beige. Ein Fuhrmann kam, lud
sie alle auf den Wagen und fuhr damit zur Stadt dem Schreiner-Toni vor
das Haus. Das Holz ward gesaegt und gespaltet; denn im kalten Winter
sollte es im knisternden Ofen die Stube waermen. - Ein Glueck, daß Toni
das eine Scheit bemerkte. Es war so huebsch gerade und hatte keinen Ast;
```

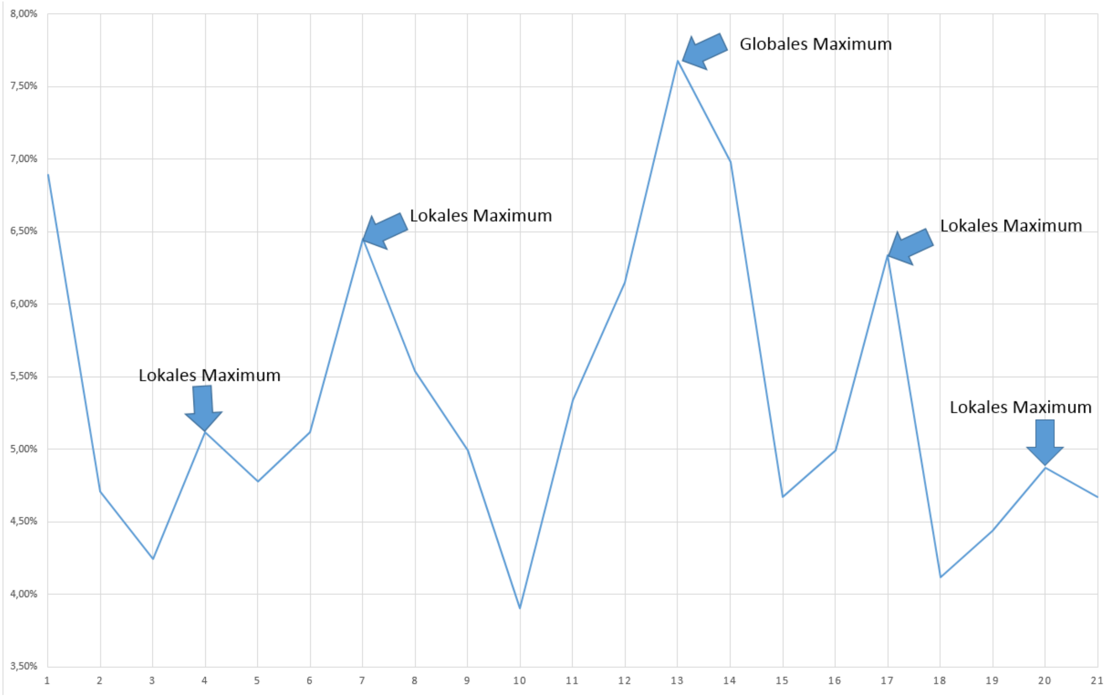


Abb. 2.2: Die Abbildung zeigt auf der x-Achse die Schlüssel und auf der y-Achse die Kostenfunktion (Koinzidenzindex). Dieser Graph dient nur zur Veranschaulichung, wie ein heuristisches Verfahren vorgeht.

```
drum stellte es der Schreiner in eine Ecke seiner Werkstatt und dachte:
Ein gutes, glattes Stueck, 's waer schade, es zu verbrennen.
```

Abb. 2.3 stellt die Häufigkeit der zehn häufigsten Einzelbuchstaben gegenüber und vergleicht den obigen Text mit der statistischen Häufigkeit der deutschen Sprache. Diese zehn Zeichen reichen in der deutschen Sprache bereits aus, um Texte lesbar zu machen.

Eine solche Häufigkeitsverteilung kann auch über Bi-, Tri- oder Tetragramme erstellt werden. Zum Beispiel kommen im Deutschen die Bigramme **er**, **en** und **ch** am häufigsten vor. Außerdem ergeben die 30 häufigsten Wörter zusammen 31,8% aller Wörter im Deutschen. Diese sind unter anderem: die, der, und, in, zu, den oder das.[Krya]

## 2.5 Koinzidenzindex

Den Koinzidenzindex erhält man durch eine statistische Auswertung der Zeichen in einem Text. Er gibt die Wahrscheinlichkeit an, mit der zwei zufällig aus einem Text gezogene Zeichen übereinstimmen. Die Häufigkeitsverteilung der Zeichen ist abhängig von der Sprache. Dies wird genutzt, um verschlüsselten Text auf dessen

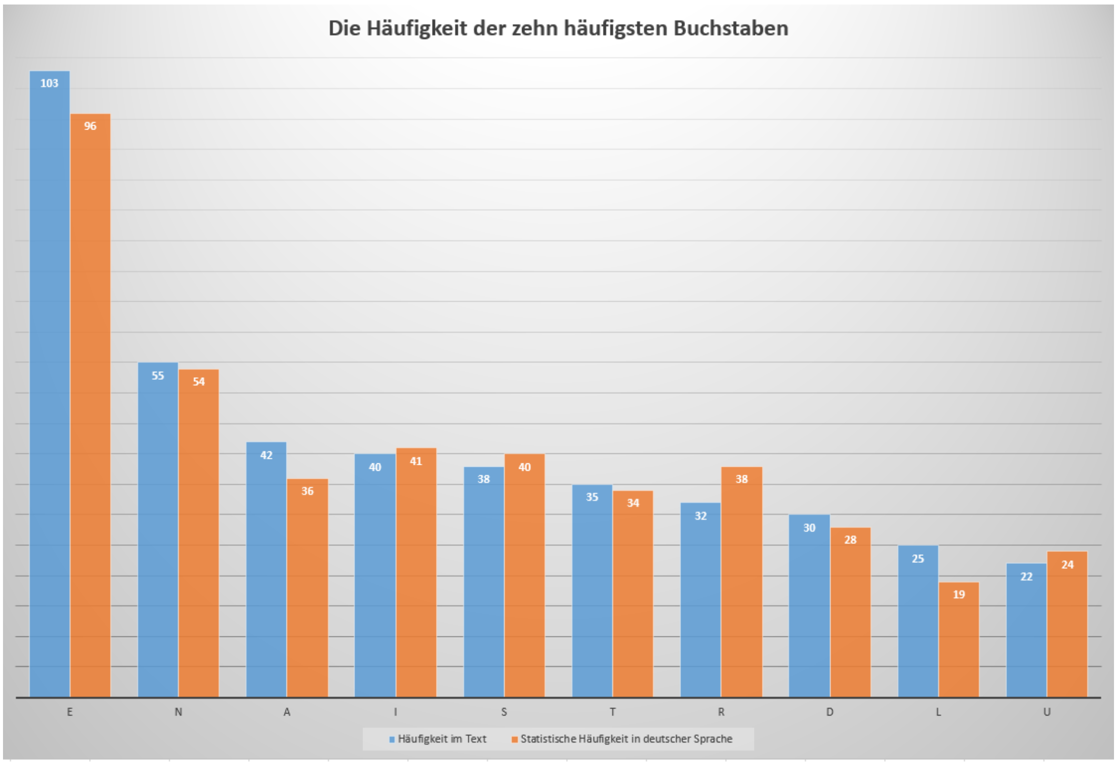


Abb. 2.3: Häufigkeiten einzelner Buchstaben für den ersten Abschnitt aus Pinocchio

sprachliche Eigenschaften zu untersuchen. Außerdem kann man mit dem Koinzidenzindex erkennen, ob ein verschlüsselter Text auf einer monoalphabetischen Substitution beruht oder nicht.[Kryb] Bei einer monoalphabetischen Substitution verändert sich die Häufigkeit der einzelnen Zeichen nicht, da diese einfach nur untereinander ausgetauscht werden.

Den Koinzidenzindex, abgekürzt $IC$, erhält man folgendermaßen: Zuerst wird gezählt, wie oft jedes Zeichen in einem Text vorkommt. Anschließend wird für jedes Zeichen die `Anzahl · (Anzahl-1)` berechnet und die Summe dieser Produkte gebildet. Zum Schluss wird dividiert durch `Textlänge · (Textlänge-1)`. Als Formel [Kryb] sieht das wie folgt aus:

$$IC = \frac{\sum_{i=A}^{Z} n_i(n_i - 1)}{N \cdot (N - 1)}$$

Der Koinzidenzindex des Textes aus Kapitel 2.4 beträgt 7,93%. Zum Vergleich der Koinzidenzindex für bestimmte Sprachen:

- `deutsch = 7,6%`
- `französisch = 7,8%`
- `italienisch = 7,6%`

- `spanisch = 7,5%`
- `englisch = 6,5%`
- `gleichverteilt = 3,85%`

Texte in einer natürlichen Sprache haben einen bis zu doppelt so hohen Koinzidenzindex wie ein Text mit gleichverteilten Zeichen. Ein solcher nahezu gleichverteilter Text kann sich aus einer polyalphabetischen Substitution ergeben.[Kryb]

Chiffren, wie die ADFGVX, die nach einer monoalphabetischen Substitution noch transponiert werden, weisen eine ähnliche Verteilung auf, wie eine polyalphabetische Substitution. Die Tatsache, dass der Koinzidenzindex einer monoalphabetischen Substitution deutlich von dem der polyalphabetischen Substitution abweicht, kann benutzt werden, um die Transposition mit statistischen Mitteln zu brechen, ohne die Substitution selbst angreifen zu müssen.

## 2.6 Kryptoanalyse von ADFGVX

Obwohl die ADFGVX-Chiffre bereits 100 Jahre alt ist, ist es auch mit der heutigen Rechenleistung nicht praktikabel, einen puren Brute-Force-Angriff auf diese durchzuführen. Zur Zeit geht man davon aus, dass ein Schlüsselraum von $2^{88}$ zu groß ist, um ihn komplett zu durchsuchen. Dieser besteht aus

$$309.485.009.821.345.068.724.781.056 \approx 3,1 \cdot 10^{26}$$

Schlüsseln. Um den Schlüsselraum in einem Jahr (1 Jahr = 31.536.000 Sekunden ) komplett zu durchsuchen, müsste man $\approx 9,8 \cdot 10^{18}$ bzw $\approx 2^{63}$ Schlüssel pro Sekunde berechnen. Bei einer Rechenzeit von zehn Jahren wären es immer noch $\approx 9,8 \cdot 10^{17}$ bzw $\approx 2^{59}$ Schlüssel pro Sekunde.

Das Notebook, auf dem die Bachelorarbeit geschrieben wurde, hat eine Intel Core i3-2330M-CPU. Eine realistische Annahme wäre, dass eine solche CPU bis zu 1.000.000 Schlüssel pro Sekunden berechnen kann, was $\approx 2^{20}$ entspricht. Dieses Notebook bräuchte somit $2^{68}$ Sekunden (295.147.905.179.352.825.856), was wiederum über neun Milliarden Jahren (genauer 9.359.078.677.681) entspricht, um den gesamten Schlüsselraum zu durchsuchen.

Bei der folgenden Kryptoanalyse wird aber recht schnell deutlich, an welcher Stelle ein Angriff ansetzen muss, um dieses Verfahren zu brechen.

### 2.6.1 Schwachstellen der monoalphabetischen Substitution

Den Schlüsselraum einer monoalphabetischen Substitution (in unseren beiden Fällen 25! und 36!) können wir dank der Häufigkeitsanalyse vernachlässigen, wie in Kapitel 2.5 beschrieben (zur Analyse genügen 1-Gramm-Statistiken). Die Zeichen in jeder Sprache kommen unterschiedlich oft vor. Durch einfaches Zählen der Zeichen wird somit deutlich, welches Zeichen durch welches ersetzt wird. Hinzu kommt, dass in der deutschen Sprache bereits die zehn häufigsten Zeichen ausreichen, um einen Text lesbar zu machen.

### 2.6.2 Schwachstelle der Transposition

Die Transposition hat eine der Substitution ähnliche Schwachstelle, die auch auf einer Häufigkeitsanalyse beruht. Hierbei werden N-Gramme mit $N > 1$ benutzt. Gesucht werden zum Beispiel die häufigsten 3-Gramme im Deutschen: *sch*, *ein* oder *aus*. Mit Hilfe solcher N-Gramme kann man erkennen, ob ein Teil der permutierten Spalten bereits richtig angeordnet ist. Dieses Verfahren funktioniert natürlich umso besser, je mehr Zeilen man bei einer Spaltentransposition zur Verfügung hat.

### 2.6.3 Auswirkung der Schwachstellen auf ADFGVX

Da ein Durchgreifen, die obere Verschlüsselungsschicht zu ignorieren und mit der tieferen zu beginnen, in diesem Fall nicht möglich ist, muss zuerst die Spaltentransposition gelöst werden. Dies liegt daran, das ein Zeichen durch ein Bigramm dargestellt wird, diese Bigramme wurden aber durch die Fraktionierung zerstört. Die Herausforderung eines Angriffs ist also, zwischen Transposition und Substitution zu kommen. Das kann man daran erkennen, ob der zurück transponierte Text einen mono- oder polyalphabetischen Charakter hat. Dies kann durch den Koinzidenzindex ermittelt werden, siehe Kapitel 2.5. Die darunter liegende Substitution kann dann am Ende mit einfachen Hilfsmitteln wie z.B. Stift und Papier, gelöst werden.

### 2.6.4 Angriffsarten

Man unterscheidet drei Angriffsarten: *ciphertext-only*, *(partially-)known-plaintext* und *chosen-plaintext*. Eine Chiffre die als sicher gilt muss allen drei Arten stand halten, die *ADFGVX-Chiffre* fällt allerdings bei den *known-plaintext* und *chosen-plaintext* Angriffen mit aktueller Technik in wenigen Minuten, teilweise sogar in Sekunden.

Wenn bereits bekannt ist, dass eine Nachricht mit

`Montag 0600 Wetterbericht sonnig zehn Grad`

anfängt, handelt es sich um einen partially-known-plaintext-Angriff. Dies bietet einem Angreifer bereits so viele Informationen, dass sich viele Transpositionen (Fraktionen) ausschließen lassen. Immerhin muss der Zwischentext, der Text nach der Substitution aber vor der Transposition, Zeichenpaare oder -folgen für *00* , *ette , nn* enthalten. Hinzu kommt noch, das diese Folgen auch in einem bekannten Abstand zueinander auftauchen werden.

Der *chosen-plaintext* ist sogar noch um einiges stärker, da beliebige, bekannte Klartexte mit der Chiffre verschlüsselt werden können. Zum Beispiel sendet man einfach ein $A$, der verschlüsselte Text dazu könnte lauten $AG$. Es gibt also nur zwei mögliche Positionen für unser $A$, nämlich $AG$ oder $GA$. Jedes Zeichen kann somit auf zwei Felder eingegrenzt werden. Fünf/Sechs Zeichen, die auf *AA, DD, FF, GG, (VV), XX* abbilden sogar auf genau ein Feld. Durch Ausschlussverfahren kann sich das Polybius-Quadrat somit fast komplett rekonstruieren lassen. Danach können ganz gezielte Angriffe auf die Transposition durchgeführt werden.

*Ciphertext-only*-Angriffe, bei denen dem Angreifer nur der verschlüsselte Text zur Verfügung steht, sind die mächtigsten Angriffe. Georges Painvin auf französischer Seite und William F. Friedman auf amerikanischer Seite fanden noch während des ersten Weltkrieges unabhängig voneinander einen Weg, solche Nachrichten teilweise zu entschlüsseln.

## 2.6.5 Angriff von G. Painvin / W. F. Friedman

Eine generelle Lösung, die Chiffre zu brechen, fanden beide damals jedoch nicht. Ihr Angriff beruht darauf, verschlüsselte Nachrichten zu analysieren, die mit demselben Schlüsselpaar verschlüsselt wurden. Friedman schilderte drei Ansätze. Diese machten es unter Umständen möglich, die Verschlüsselung zu brechen. Painvins Vorgehen ähnelte der ersten und zweiten Methode:

1. `Special solution by the means of identical endings`
2. `Special solution by the means of identical beginnings`
3. `Special solution by the exact factor method`

### 2.6.5.1 Special solution by the means of identical endings

Das Verfahren kann in drei Teilschritte unterteilt werden. Im ersten und auch aufwendigsten Schritt wird herausgefunden, ob zwei Nachrichten überhaupt ein identisches Ende haben, wie lang der Transpositionsschlüssel ist und welche der Spalten länger und welche kürzer sind. Diese Informationen greifen so ineinander, dass diese nicht nacheinander, sondern eigentlich zeitgleich ermittelt werden.

Betrachten wir die beiden Geheimtexte Text 1 sowie Text 2:

Text 1:

```
XVAAX VDDAG DADVF ADADA FXGFV XFAXA XVAVF AVXDA GFFXF FGAGF DGDGD
DGAFD AADDD XDAVG GAADX ADFVF FDFXF GFGAV AFAFX FFXFX FVDGX AFFGX
AAAVA VAFAG DDFAG VFADV FAVVX GVAAA FDFAX XFAAG DX
```

Text 2:

```
FDFFF FVFAD DVFVD GAFDF DAGAD FDFAF GAXGD VXGFX VXDXV AAAAD GXFFD
VFAAG VGVFF FDAFF FXDAF XGAFD VFGXV DDFAD DAAAX AAFFA FVFXF FAXXA
XDGXA VDAVF DFAVX VADXF AXFFX XAAVX XADXA AAVVG AGDXX FDFAX FDGDF
FXDGX FAGDF FDDVD DXDAF AGXXA FGAV
```

Zuerst betrachtet man das letzte Tetra- oder Trigramm beider Nachrichten und versucht, es in dem jeweilig anderen Geheimtext an einer anderen Stelle wiederzufinden. In diesem Beispiel finden wir das letzte Tetragramm AGDX aus Text 1, an der Stelle 151-154, in Text 2, sowie das Tetragramm FGAV aus Text 2, an der Stelle 87-90, in Text 1 wieder. Ist dies bereits nicht der Fall, eignet sich das Nachrichtenpaar nicht für dieses Verfahren. Wenn doch, kann mit diesen Informationen bereits die Schlüssellänge für die Transposition bestimmt bzw. sehr stark eingegrenzt werden. Der Grund dafür liegt in der Eigenschaft der Spaltentransposition selbst: Wenn das Ende des Klartextes beider Nachrichten identisch ist, bleiben auch die Zeichen, die am Ende der Spalten untereinander stehen, identisch. Diese stehen maximal in einer anderen Spalte, da der Text zuvor unterschiedlich lang ist. Das gefundene Tetragramm ADGX in Text 2 und FGAV in Text 1 muss also in beiden Fällen am Ende einer Spalte stehen. Unter Berücksichtigung der Länge der beiden Geheimtexte und der Position der sich mitten im Text befindlichen Tetragramme kann angenommen werden, dass die Schlüssellänge 18 oder 20 beträgt.

Dies wird deutlich in Tabelle 2.3, wo man sieht, wie viele Spalten welcher Länge bei welcher Schlüssellänge möglich sind (weil beide Texte ein identisches Ende haben und mit demselben Schlüssel verschlüsselt wurden, wird die Menge möglicher Schlüssellängen stark eingegrenzt). Bei der Länge 19 wird sofort ersichtlich, dass diese nicht in Frage kommen kann, da alle Spalten von Text 1 die Länge acht haben müssen. Bei dieser Länge würde das Tetragramm FGAV, welches an der Stelle 87-90 steht, nicht am Ende einer Spalte liegen.

| Länge | 18 | 19 | 20 | 21 |
|---|---|---|---|---|
| Text 1 | 8x9 & 10x8 | 19x8 | 12x8 & 8x7 | 5x8 & 16x7 |
| Text 2 | 14x11 & 4x10 | 4x11 & 15x10 | 14x10 & 6x9 | 5x10 & 16x9 |

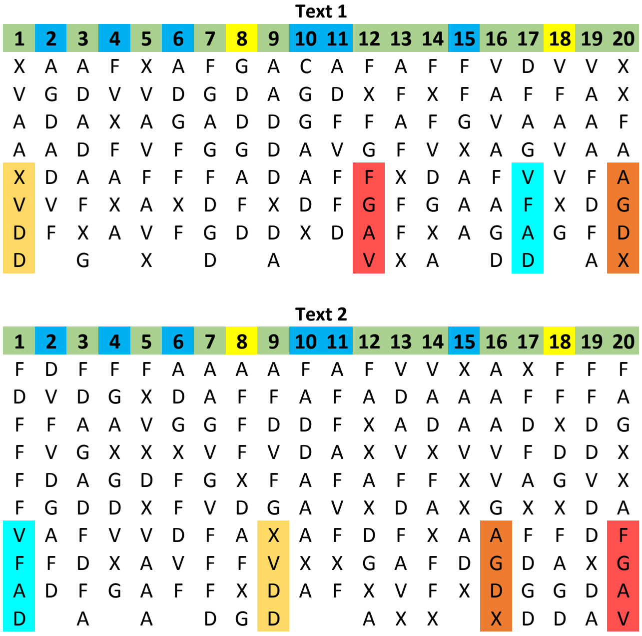
Tab. 2.3: Mögliche Schlüssellängen für die Texte 1 und 2, um die Transposition zu lösen

Wenn man von einer Schlüssellänge 20 ausgeht, muss es also 20 Tetra- oder Trigrammpaare geben, die sich in beiden Texten finden lassen. Da aber im seltensten Fall eine reguläre[2] Spaltentransposition vorliegt, sondern meist eine irreguläre[3]

---

[2] Alle Spalten sind gleich lang.

[3] Einige Spalten sind um ein Zeichen länger.



Text 1

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

Text 2

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

Abb. 2.4: Beide Nachrichten mit validen Spaltenenden

und man zu diesem Zeitpunkt noch nicht weiß, welche der Spalten länger und welche kürzer sind, ist das Finden dieser Paare nicht trivial. Betrachtet man Text 1 endet die erste Zeile also entweder mit AXVD bzw. XVD (7 Zeichen) oder XVDD bzw. VDD (8). In Text 2 finden wir an der Stelle 84-87 ebenfalls ein XVDD und an Stelle 179-181 ein VDD. Das VDD in 179-181 kann aber direkt ausgeschlossen werden, da dies unter der aktuellen Annahme in Text 2 nicht an einem Spaltenende liegen kann. Das Tetragramm XVDD ($6x10 + 3x9 = 87$) kommt dagegen in Frage, allerdings könnte auch das XVD($84$-$86$ ; $5x10 + 4x9 = 86$) das korrekte Trigramm und das folgende *D* nur durch Zufall entstanden sein. Durch die Einschränkung der Zeichen beträgt die Wahrscheinlichkeit hierfür immerhin 1 zu 6. Der durch solche Situationen entstehende große Entscheidungsbaum macht diesen Ansatz komplex und aufwendig. Das Ziel ist es, alle passenden N-Gramme an validen Spaltenenden zu finden. Siehe Abb. 2.4.

Im zweiten Schritt können die Spalten bereits in unterschiedliche Gruppen eingeteilt werden, die Zeilen 1,3,5,7,9,12,13,14,16,17,19 und 20 sind in beiden Texten lang (in Abb. 2.4 grün markiert), die Zeilen 8 und 18 (in Abb. 2.4 gelb) sind in Text 1 kurz und in Text 2 lang und die Zeilen 2,4,6,10,11 und 15(in Abb. 2.4 blau) sind in beiden Texten kurz. Zusätzlich können anhand der Positionen der gefundenen N-Gramm-Paare aus den Texten Äquivalenzketten hergeleitet werden. Zum Beispiel das letzte Tetragramm aus Text 1 (AGDX) ist das Ende der Spalte 16 aus Text 2.

N-Gramm-Paare aus den beiden Texten

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

1 zu 9 , 9 zu 19 , ... , 7 zu 17 , 17 zu 1 und die zweite Kette 4 zu 10 , 10 zu 15 , ... , 16 zu 18 , 18 zu 4. Diese beiden Ketten müssen anschließend wieder zusammengeführt werden. Durch die vorherige Einteilung der Spalten in Gruppen wissen wir, dass die ersten zwölf Spalten durch 1,3,...,20 dargestellt werden. Nachfolgend in Orange gekennzeichnet.

Beide Ketten in initialer Reihenfolge:

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

Beide Ketten in ablesbarer Reihenfolge:

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

So lässt sich die richtige Permutation der Transposition direkt ablesen, indem die Ketten wie im Reißverschlussverfahren ineinander greifen:

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

Mit der Häufigkeitsanalyse kann das Polybius-Quadrat rekonstruiert werden, womit sich folgende Klartexte ergeben:

`Have ordered commanding brigade to counter attack without delay with all arms`

und

`Expect enemy attack at daylight stop hold your sector without fail stop counter attack without delay with all arms.` [Fri39, 116-123]

#### 2.6.5.2 Special solution by the means of identical beginnings

Im Gegensatz zur vorherigen ist der Anfang dieser Methode weniger komplex. Bereits an den ersten Zeichen zweier verschlüsselten Nachrichten kann man erkennen, ob diese mit dem gleichen Klartext beginnen, da die beginnende Zeichenfolge identisch sein muss. Anschließend sucht man alle weiteren identischen Zeichenfolgen. Ein großer Vorteil dabei ist, dass diese Zeichenfolgen auch in der gleichen Reihenfolge auftauchen. Darüber hinaus lässt sich durch die Anzahl an gefundenen, identischen Zeichenfolgen bereits erkennen, wie lang der gesuchte Schlüssel ist.

Text 3:

```
XDAAA GXDDX VFFVD GADFD XAAAG DFADG AFDAD GVGDV FDFXA GFXAF AFAXD
DDDFD XAXVA DXFXF DGAGF GGADD AGDGX AVGDG ADAFA XFAAG VAAGA FDVDV
DXFDA XFDFF GDXDV DADAV DADDD GADAG AAAFG GDXAX FGVXD DGDDF AFAGV
AFGXG VDDAX XDVFF FFDXG VGDFG AVADA XDAFA AFDGF VFXXX AAGAG AFDGX
AFAFX XGGAG AAFFA AFDGA GAFVX DGGFG DAAAF DADAD XVVAX FVADD GAFFF
GXAXD FDDFX AAAAA
```

Text 4:

```
XDAAV DXDGF XVGDD AVGXA DXAAD XGGAA GDFDA AAGAX DVFDF DFFDD FDDFX
FXXFD FDXAX GAXFF VDVAF GVDVD DDAGD GGDAA GGFDD DVFFV VAGVA XAAGG
XGXDD DADXF ADFFG DGFDA AFGAX FFDVD DDAGA FADAV DDDAV GAVAD FGDDF
FDGDV DGGXA XAXDA DXDVF FXVAX GFDAG XFFFF AAXDA FVDXG XFDAG AGAVD
VAGAF DGDAV VDDDD DFXGV AFFAA FFFDV DFFAF DAGDG FAAAF DXAXA VAXDA
GADXD VFAFF FGDDA DDDFA GDFAX DG
```

In Text 3 und Text 4 lassen sich zum Beispiel 15 identische Zeichenketten finden. Für Text 3 bedeutet das, bei 290 Zeichen, 5 Zeilen á 20 Zeichen und 10 Zeilen a 19 Zeichen. Für Text 4, bei 302 Zeichen, 2 Zeilen á 21 Zeichen und 13 Zeilen á 20 Zeichen.

Zu diesem Zeitpunkt ist allerdings noch nicht offensichtlich, welche der Zeilen lang und welche kurz sind. Einige lassen sich sofort identifizieren, andere hingegen können durch ein zufällig identisches Zeichen, das vor oder hinter der richtigen identischen Zeichenfolge steht, falsch interpretiert werden. Auffällig sind in diesem Fall Zeichenfolgen, die atypisch länger sind als andere. Hat man wie in diesem Beispiel identische Zeichenfolgen mit vier, fünf und sechs Zeichen, kann davon ausgegangen werden, dass in den sechs Zeichen langen Folgen ein Zeichen zufällig sein muss. In Abb. 2.5 sind die richtigen Spaltenlängen rekonstruiert.

Im zweiten Teil dieser Methode ist die korrekte Reihenfolge der Spalten zu rekonstruieren. Dabei ist entscheidend, ob die Länge des Schlüssels gerade oder ungerade ist.

Ist die Länge des Schlüssels gerade, führt das dazu, dass die senkrechten und waagerechten Buchstaben des Polybius-Quadrats jeweils eine komplette Spalte belegen. Kombiniert man diese bipartiten Komponenten durch eine Gegenüberstellung jeweils einer dieser Spalten, entsprechen diese durchaus Klartextbuchstaben und die Verteilung ist ebenfalls monoalphabetisch. Durch diese Information können Spaltenpaare gebildet werden, die die Lösung des Problems auf eine Spalten-Transpositions-Chiffre ohne fraktionierten Buchstaben zurückführt.

In diesem Fall ist die Schlüssellänge ungerade und es wird deutlich mehr Statistik und Mathematik benötigt, deshalb genügen an der Stelle auch die beiden bisherigen Nachrichten nicht aus. Die Länge und der Inhalt weitere Nachrichten spielt dabei keine Rolle, diese müssen lediglich mit demselben Schlüsselpaar verschlüsselt worden sein. Mit Hilfe der Häufigkeitsanalyse wird geprüft, ob die Reihenfolge der Spalten monoalphabetisch oder polyalphabetisch ist. Als erstes berechnet man Referenzwerte für einen mono- und polyalphabetischen Text und anschließend werden alle Bigramme gezählt und darüber ein Durchschnittswert berechnet. Wobei die Spaltenreihenfolge, deren Wert die kleinste Differenz zum monoalphabetischen Referenzwert hat, sehr wahrscheinlich die Richtige ist. So wird wiederholt fortgefahren, bis die Reihenfolge korrekt ist.

Zum Schluss erfolgt wie bei der vorhergehenden Methode, mittels Häufigkeitsanalyse die Rekonstruktion des Polybius-Quadrat und das Übersetzen der Nachricht in Klartext.[Fri39, 123-145]

#### 2.6.5.3 Special solution by the exact factor method

Bei der dritten Methode kommt es darauf an, dass die Spalten einer verschlüsselten Nachricht dieselbe Länge haben. Das bedeutet das die Länge der Nachricht ein Vielfaches des Transpositionsschlüssels ist. Dass genau dies eintritt ist theoretisch nur selten der Fall. Allerdings ist es für einen Nachrichtenschreiber einfacher, vollständig gefüllte Nachrichten zu verschicken, dies reduziert beim übertragen einfache Fehler. Deshalb kam es vor das Nachrichten bewusst aufgefüllt wurden.

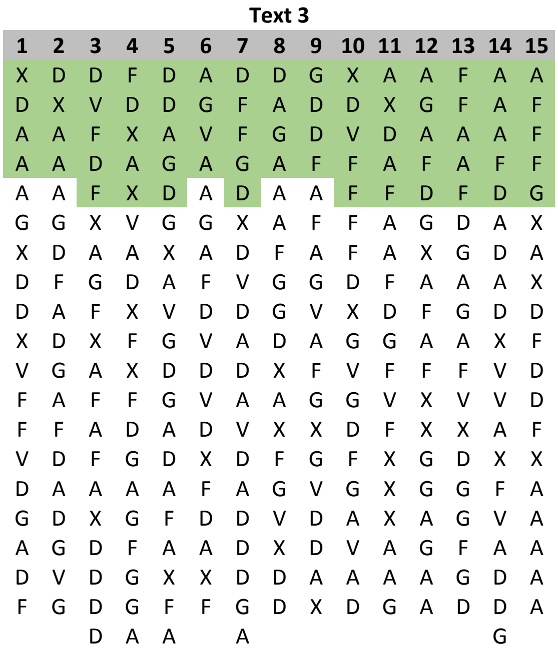
Nachdem die Nachricht in Spalten gleicher Länge angepasst wurden, wird die gleiche Methode angewendet, wie in Kapitel 2.6.5.2 mit gerader Schlüssellänge, um somit die beste monoalphabetische Verteilung festzustellen. Gelingt es bereits zwei oder drei Spaltenpaare richtig aufzustellen, erfolgt der Rest der Lösung fast automatisch.[Fri39, 145-148]

### 2.6.6 Angriffe mit generellen Lösungen

Später entwickelten Friedman, Konheim und Lasry Verfahren, die deutlich weniger Anforderungen an die Geheimtexte stellten.

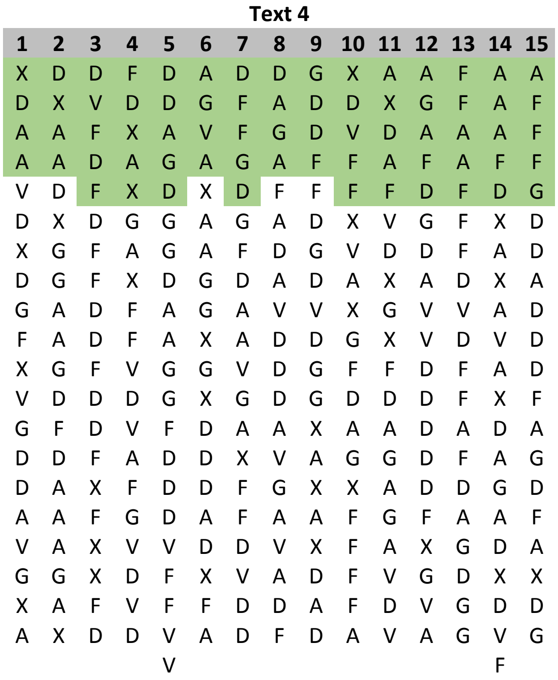
#### 2.6.6.1 W. F. Friedman

Die einzige Bedingung, die Friedman an seinen Lösungsansatz stellt ist, dass die Substitution einen monoalphabetischen Charakter haben muss. Im ersten Schritt nutzt er eine bekannte Eigenschaft, die auch in Abschnitt 2.6.5.2 verwendet wird, um zu erkennen, ob die Länge des Transpositionsschlüssels gerade oder ungerade ist. Bei gerader Länge sind die waagerechten und die senkrechten Zeichen des Polybius-Quadrats jeweils in den Spalten separiert. Bei ungerader Länge wechseln sie sich in den Spalten ab. Durch das Anlegen zweier Frequenztabellen, wobei in der ersten Tabelle immer das erste, dritte, fünfte, usw. und in der zweiten Tabelle das zweite, vierte, sechste, usw Zeichen notiert wird, kann unterschieden werden,



**Text 3**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X | D | D | F | D | A | D | D | G | X | A | A | F | A | A |
| D | X | V | D | D | G | F | A | D | D | X | G | F | A | F |
| A | A | F | X | A | V | F | G | D | V | D | A | A | A | F |
| A | A | D | A | G | A | G | A | F | F | A | F | A | F | F |
| A | A | F | X | D | A | D | A | A | F | F | D | F | D | G |
| G | G | X | V | G | G | X | A | F | F | A | G | D | A | X |
| X | D | A | A | X | A | D | F | A | F | A | X | G | D | A |
| D | F | G | D | A | F | V | G | G | D | F | A | A | A | X |
| D | A | F | X | V | D | D | G | V | X | D | F | G | D | D |
| X | D | X | F | G | V | A | D | A | G | G | A | A | X | F |
| V | G | A | X | D | D | D | X | F | V | F | F | F | V | D |
| F | A | F | F | G | V | A | A | G | G | V | X | V | V | D |
| F | F | A | D | A | D | V | X | X | D | F | X | X | A | F |
| V | D | F | G | D | X | D | F | G | F | X | G | D | X | X |
| D | A | A | A | A | F | A | G | V | G | X | G | G | F | A |
| G | D | X | G | F | D | D | V | D | A | X | A | G | V | A |
| A | G | D | F | A | A | D | X | D | V | A | G | F | A | A |
| D | V | D | G | X | X | D | D | A | A | A | A | G | D | A |
| F | G | D | G | F | F | G | D | X | D | G | A | D | D | A |
|  |  | D | A | A |  | A |  |  |  |  |  |  | G |  |



**Text 4**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X | D | D | F | D | A | D | D | G | X | A | A | F | A | A |
| D | X | V | D | D | G | F | A | D | D | X | G | F | A | F |
| A | A | F | X | A | V | F | G | D | V | D | A | A | A | F |
| A | A | D | A | G | A | G | A | F | F | A | F | A | F | F |
| V | D | F | X | D | X | D | F | F | F | F | D | F | D | G |
| D | X | D | G | G | A | G | A | D | X | V | G | F | X | D |
| X | G | F | A | G | A | F | D | G | V | D | D | F | A | D |
| D | G | F | X | D | G | D | A | D | A | X | A | D | X | A |
| G | A | D | F | A | G | A | V | V | X | G | V | V | A | D |
| F | A | D | F | A | X | A | D | D | G | X | V | D | V | D |
| X | G | F | V | G | G | V | D | G | F | F | D | F | A | D |
| V | D | D | D | G | X | G | D | G | D | D | D | F | X | F |
| G | F | D | V | F | D | A | A | X | A | A | D | A | D | A |
| D | D | F | A | D | D | X | V | A | G | G | D | F | A | G |
| D | A | X | F | D | D | F | G | X | X | A | D | D | G | D |
| A | A | F | G | D | A | F | A | A | F | G | F | A | A | F |
| V | A | X | V | V | D | D | V | X | F | A | X | G | D | A |
| G | G | X | D | F | X | V | A | D | F | V | G | D | X | X |
| X | A | F | V | F | F | D | D | A | F | D | V | G | D | D |
| A | X | D | D | V | A | D | F | D | A | V | A | G | V | G |
|  |  |  |  | V |  |  |  |  |  |  |  |  | F |  |

Abb. 2.5: Beide Nachrichten mit validen Spaltenlängen

ob es sich um eine gerade oder ungerade Länge handelt. Wird eine gerade Länge verwendet sind die beiden Tabellen in ihrer Aufteilung ähnlich, sind sie es nicht, wurde eine ungerade Länge verwendet. Friedman erwähnt, das dies bereits ab 20 oder 25 Zeichen pro Tabelle erkannt werden kann.

Anschließend nutzt man die erstellten Frequenztabellen, um herauszufinden, an welchen Stellen im Text ein Spaltenwechsel stattfindet. Zeichen, die in einer Frequenztabelle sehr häufig vermerkt wurden, tauchen abwechselnd in bestimmten Abständen an geraden und ungeraden Stellen im Text auf. Dies schränkt die Position eines Spaltenwechsels ein, genauer kann er erkannt werden, wenn auf ein häufig auftretendes Zeichen eines folgt, das in der zweiten Frequenztabelle eher unwahrscheinlich ist, aber in der gleichen Tabelle ebenfalls sehr häufig vorkommt. Sind die Anzahl Zeichen pro Spalte bekannt, kann anhand der Textlänge die Länge des Transpositionsschlüssels ermittelt werden. Identifiziert man weitere dieser Spaltenwechsel, können lange und kurze Spalten eingegrenzt werden. Je mehr Nachrichten zur Verfügung stehen, umso detaillierter die gewonnenen Informationen.

Im folgenden Schritt wird ein „mathematischeres“ Verfahren benötigt, um Spalten in zwei Kategorien, die die geraden und ungeraden Spalten repräsentieren sollen, zu trennen. Dafür werden die vorhandenen Frequenztabellen vergrößert und eine Gewichtung eingeführt, anhand derer eine Zuordnung der Spalten erstellt wird. Sind alle Spalten nach gerade und ungerade getrennt und alle möglichen Kombinationen von kurzen und langen Spalten berücksichtigt worden, ergeben sich bestimmte Bedingungen, die die Auswahl stark eingrenzen. Dieser Teil des Verfahrens ist allerdings sehr rechenaufwändig und hier stark vereinfacht dargestellt.

Nach der Lösung der Transposition kann mittels Häufigkeitsanalyse die Substitution ermittelt werden.[Fri34]

#### 2.6.6.2 A. G. Konheim

Auch Konheim nutze die statistische Analyse von potentiellen Bigrammen, um festzustellen, ob zwei Spalten aus dem ursprünglich nicht verschlüsselten Transpositionsrechteck nebeneinander liegen müssten. Bei den bisherigen Methoden wurde als nächstes die Transposition rückgängig gemacht; Konheim allerdings löst mithilfe von nicht ausführlich beschriebenen Standardtechniken die Substitution und erst danach die Transposition.[Kon85]

#### 2.6.6.3 G. Lasry

Das Grundprinzip, die *ADFGVX* zu entschlüsseln, beinhaltet, bis auf den Ansatz von Konheim, immer, dass zuerst die Transposition und dann die Substitution gelöst wird. Bei den bereits genannten Methoden wurden immer eine oder zwei Nachrichten eines bestimmtes Typs betrachtet. Dies ist bei George Lasry's Methode nicht der Fall. Der große Unterschied liegt dabei in der Entschlüsselung der

Transposition: Durch die Nutzung von Simulated Annealing, wie im Kapitel 2.3 beschrieben, benötigt er mehrere Nachrichten, die mit demselben Schlüssel verschlüsselt wurden. Ist die Schlüssellänge bekannt, versucht das Verfahren sich dem korrekten Schlüssel anzunähern. Ist diese nicht bekannt, wird das Verfahren für die unterschiedlichen Schlüssellängen jeweils erneut gestartet.

Da die Länge des Schlüssels nicht immer bekannt ist, wird eine Länge von 15 bis 25 Zeichen angenommen und dieser Raum entsprechend nacheinander durchsucht. Der Algorithmus beginnt damit, dass ein zufälliger Transpositionsschlüssel gewählt und die Nachricht mit dem Schlüssel entschlüsselt wird. Nun wird ein neuer Schlüssel aus dem bereits gewählten abgeleitet und die Nachricht ebenfalls wieder mit dem Schlüssel entschlüsselt. Von beiden entschlüsselten Nachrichten wird der Koinzidenzindex aller resultierenden Bigramme berechnet. Ist der Koinzidenzindex des neuen Schlüssels höher, wird dieser Schlüssel behalten. Ist er kleiner, wird er verworfen und ein neuer Schlüssel aus dem ersten abgeleitet. Das ganze wird solange wiederholt, bis der Koinzidenzindex über einem Schwellwert liegt, wodurch angenommen werden kann, dass der richtige Transpositionsschlüssel gefunden wurde. Ist es dennoch der falsche Schlüssel, beginnt die Prozedur von vorne und wählt einen anderen zufälligen Schlüssel aus und leitet wieder solange neue Schlüssel daraus ab, bis der Koinzidenzindex wieder über dem Schwellwert liegt.

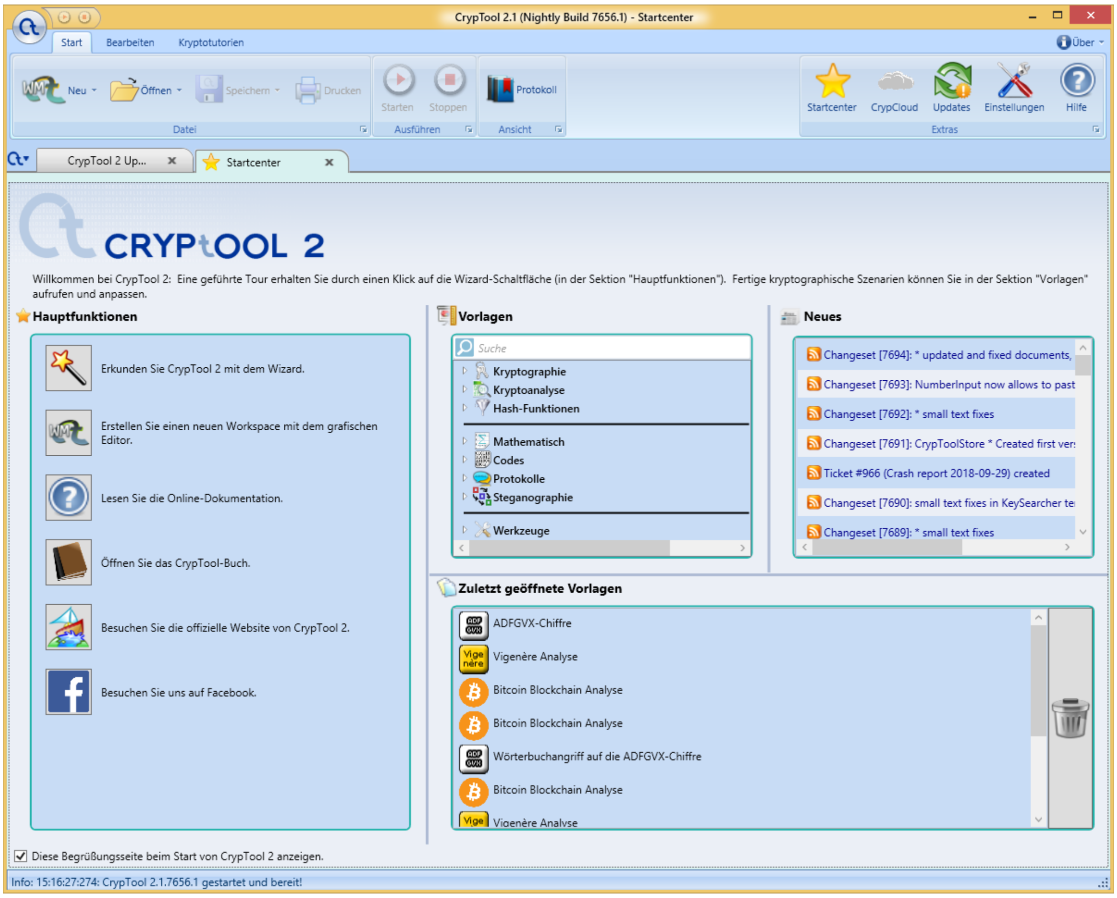
Sollten diese Schritte nach 1000 Versuchen nicht zum Erfolg führen, ist entweder die angenommene Schlüssellänge falsch, sind die Nachrichten zu kurz, nicht (wie angenommen) mit demselben Schlüssel verschlüsselt oder der Algorithmus hat den Schlüssel nicht gefunden. Bei 1000 Versuchen ist letzteres aber laut Lasry am unwahrscheinlichsten.

Diese Methode funktioniert bei Schlüsseln mit ungerader Länge sehr gut, bei einer geraden Länge gibt es allerdings ein Problem. Der Algorithmus erkennt zwar die richtigen Spaltenpaare, so dass eine richtige Zusammensetzung der Bigrammpaare möglich ist und das Polybius-Quadrat auch entschlüsselt werden kann, aber er erkennt nicht, ob die Spaltenpaare in der richtigen Reihenfolge angeordnet sind. Das liegt darin begründet, dass der Koinzidenzindex in den Fällen identisch ist. Ist zum Beispiel ein Transpositionsschlüssel aus vier Elementen gegeben (1,2,3,4) und die Spaltenpaare {(2,4)(1,3)}, so resultiert daraus, dass das Ergebnis des Algorithmus für (2,4,1,3) identisch zu (1,3,2,4) ist. In der entsprechenden Sprache ergibt jedoch nur eine Reihenfolge Sinn. Der Algorithmus berücksichtigt zwar auch Tetragramme, die in diesem speziellen Fall noch helfen, aber ab einer Schlüssellänge von 8 Elementen gibt es wieder Zusammenstellungen, die ein identisches Ergebnis hervorbringen.[Las16]

## 2.7 CrypTool 2

Unter der Leitung von Bernhard Esslinger wurde 1998 das Projekt „CrypTool" gegründet. Seit 2003 ist es ein Open-Source-Projekt. Die maßgeblichen Resultate des Projekts sind die E-Learning-Programme CrypTool 1 mit den Nachfolgern CrypTool 2 und JCrypTool (ab 2007/2008). Inzwischen zählen sie zu den weltweit am weitesten verbreiteten Programmen im Bereich Kryptografie und Kryptoanalyse und finden Anwendung sowohl in Lehre als auch in Aus- und Fortbildung.[Ver18]

CrypTool 2 (im weiteren Verlauf abgekürzt CT2) ist eine Windows-Anwendung und wird in der Programmiersprache C# [The17] entwickelt. Sie basiert auf dem derzeit aktuellen .NET-Framework 4.7.1 und der Windows Presentation Foundation WPF [Hub15], die eine Vektorgrafik-basierte Oberfläche erlaubt. Die Architektur der Software ist modularisiert, so dass sie ohne großen Aufwand um neue Funktionen und Komponenten ergänzt werden kann.[Ver18] Die Programme werden pro Monat rund 10.000 mal von der CT-Webseite herunter geladen.[4]



Willkommen bei CrypTool 2: Eine geführte Tour erhalten Sie durch einen Klick auf die Wizard-Schaltfläche (in der Sektion "Hauptfunktionen"). Fertige kryptographische Szenarien können Sie in der Sektion "Vorlagen" aufrufen und anpassen.

Erkunden Sie CrypTool 2 mit dem Wizard.

Erstellen Sie einen neuen Workspace mit dem grafischen Editor.

Lesen Sie die Online-Dokumentation.

Öffnen Sie das CrypTool-Buch.

Besuchen Sie die offizielle Website von CrypTool 2.

Besuchen Sie uns auf Facebook.

Changeset [7694]: * updated and fixed documents,
Changeset [7693]: NumberInput now allows to past
Changeset [7692]: * small text fixes
Changeset [7691]: CrypToolStore * Created first ver:
Ticket #966 (Crash report 2018-09-29) created
Changeset [7690]: small text fixes in KeySearcher te
Changeset [7689]: * small text fixes

☑ Diese Begrüßungsseite beim Start von CrypTool 2 anzeigen.

Info: 15:16:27:274: CrypTool 2.1.7656.1 gestartet und bereit!

Abb. 2.6: CT2-Startcenter

---

[4] Angaben aus der monatlichen Downloadstatistik des CT-Projekts

### 2.7.1 Das Startcenter – Bedienung

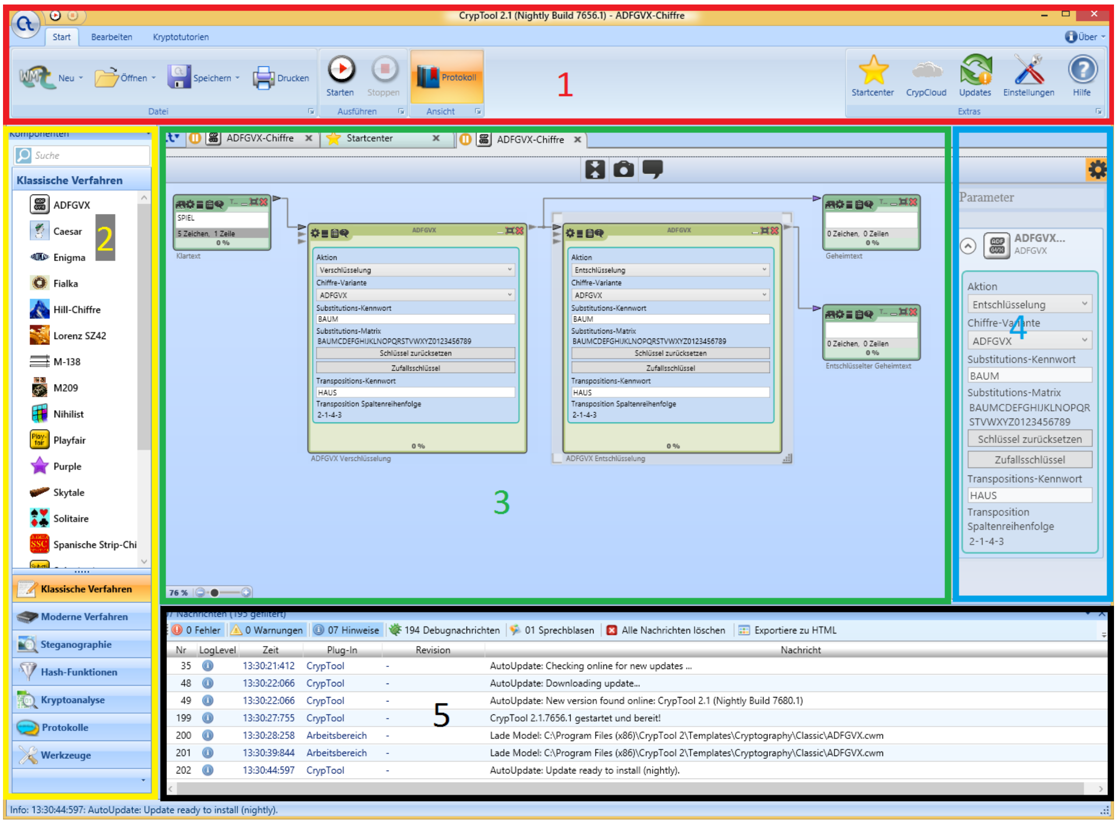
Beim Starten von CT2 öffnet sich das Startcenter (Abb. 2.6). In diesem gibt es **vier** Bereiche: *Hauptfunktionen*, *Vorlagen*, *Neues* und *Zuletzt geöffnete Vorlagen*:

- Über die *Hauptfunktionen* kann man bspw. mit dem Wizard das Programm erkunden, einen neuen Workspace mit dem eingebauten grafischen Editor erstellen, das CrypTool-Buch oder die Online-Dokumentation lesen sowie die offizielle Webseite oder die Facebook-Seite zu CrypTool 2 besuchen.
- Eine Vorlage ist ein grafisches Programm, das ein ganzes Szenario enthält. Dazu beinhaltet es eine oder mehrere Komponenten, die miteinander verbunden sind. Vorlagen werden als Ganzes in den Arbeitsbereich geladen. Alle verfügbaren Vorlagen findet man über die eingebaute Suchfunktion im Bereich *Vorlagen*.
- Im Bereich *Neues* sieht man die aktuellsten Änderungen am Projekt.
- Bereits vom Benutzer geöffnete Vorlagen werden in eine Liste unter *Zuletzt geöffnete Vorlagen* geschrieben.

### 2.7.2 Der Arbeitsbereich – Bedienung

Wenn man eine Vorlage im Arbeitsbereich öffnet, zeigt CT2 **sechs** Bereiche, wie in Abb. 2.7 dargestellt:

- Im Bereich mit der Nummer 1 (oben, rot umrahmt) ist die Titelleiste, das Hauptmenü und der Ribbonbar zu sehen. Hier kann man unter anderem die geöffneten Programme starten oder stoppen.
- Im zweiten Bereich (gelb umrahmt) sind alle verfügbaren Komponenten zu sehen, eingeteilt in Kategorien (Klassische Verfahren, Moderne Verfahren, Steganographie, hash-Funktionen, usw.). Zusätzlich können diese über die Suchfunktion oben schnell gefunden werden. Die in dieser Bachelorarbeit neu zu implementierende Analyse-Komponente wird in die Kategorie *Kryptoanalyse* einsortiert.
- Der dritte Bereich (grün umrahmt) ist die Arbeitsfläche, in der die Vorlagen geladen und ausgeführt, oder eigene Workflows und Vorlagen erstellt werden können. Die Komponenten auf der Arbeitsfläche können über ihre Ein- und Ausgänge miteinander verbunden werden. Hier sind exemplarisch zwei ADFGVX-Komponenten zu sehen, die eine Nachricht (nach Eingabe der beiden Schlüssel) verschlüsseln – und danach gleich wieder entschlüsseln.
- Rechts daneben (blau umrahmt) liegt der vierte Bereich, in dem die Parameter angezeigt werden, die sich bei der selektierten Komponente einstellen lassen. Parameter können nur verändert werden, wenn das Programm gestoppt ist.



Aktion
Entschlüsselung
Chiffre-Variante
ADFGVX
Substitutions-Kennwort
BAUM
Substitutions-Matrix
BAUMCDEFGHIJKLNOPQRSTVWXYZ0123456789
Schlüssel zurücksetzen
Zufallsschlüssel
Transpositions-Kennwort
HAUS
Transposition Spaltenreihenfolge
2-1-4-3

| Nr | LogLevel | Zeit | Plug-In | Revision | Nachricht |
|---|---|---|---|---|---|
| 35 | ⓘ | 13:30:21:412 | CrypTool | - | AutoUpdate: Checking online for new updates ... |
| 48 | ⓘ | 13:30:22:066 | CrypTool | - | AutoUpdate: Downloading update... |
| 49 | ⓘ | 13:30:22:066 | CrypTool | - | AutoUpdate: New version found online: CrypTool 2.1 (Nightly Build 7680.1) |
| 199 | ⓘ | 13:30:27:755 | CrypTool | - | CrypTool 2.1.7656.1 gestartet und bereit! |
| 200 | ⓘ | 13:30:28:258 | Arbeitsbereich | - | Lade Model: C:\Program Files (x86)\CrypTool 2\Templates\Cryptography\Classic\ADFGVX.cwm |
| 201 | ⓘ | 13:30:39:844 | Arbeitsbereich | - | Lade Model: C:\Program Files (x86)\CrypTool 2\Templates\Cryptography\Classic\ADFGVX.cwm |
| 202 | ⓘ | 13:30:44:597 | CrypTool | - | AutoUpdate: Update ready to install (nightly). |

Info: 13:30:44:597: AutoUpdate: Update ready to install (nightly).

Abb. 2.7: CT2-Arbeitsbereich mit geöffneter ADFGVX-Vorlage.

- Im fünften Bereich (unten, schwarz umrahmt) sieht man die Nachrichtenkonsole (Log): In dieser werden alle Fehler, Warnungen Hinweise oder sonstige Nachrichten angezeigt, die während der Nutzung von CrypTool 2 entstehen.
- Ganz unten ist die Statuszeile (braun umrahmt).

Mit F11 werden der gelbe und der schwarze Bereich (#2 und #5) ausgeblendet. Mit F12 werden der Ribbonbar mit seinen großen Ikonen (im roten Bereich, #1) und die Statuszeile ganz unten ausgeblendet. Beide Funktionstasten arbeiten alternierend. Damit kann man sich schnell mehr Platz für den Arbeitsbereich (Bereich #3) schaffen.

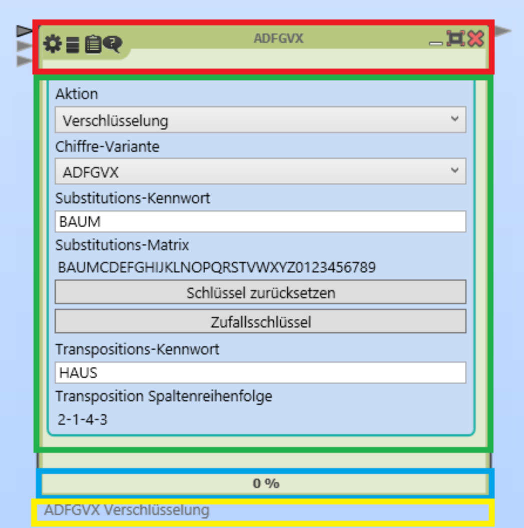
### 2.7.3 Komponenten – Bedienung

Entwickler, die Erweiterungen für CT2 schreiben, bauen ein Plugin. Ein Plugin ist eine .NET-Assembly, die eine oder mehrere Komponenten beinhaltet.[Ver18]

Eine Komponente ist eine Funktion, die auf dem Arbeitsbereich liegt und nach dem Starten (Drücken des Starten-Buttons; im Englischen Play-Button) berechnet und ausgeführt wird. Im minimierten Zustand wird eine Komponente als Icon dargestellt. Abb. 2.8 zeigt eine Komponente im maximierten Zustand, bei der im Präsentationsbereich die Einstellungen angezeigt werden.

Abb. 2.8 zeigt eine typische Komponente im maximierten (aufgeklappten) Zustand – diese wurde in der Abbildung in **vier** Bereiche unterteilt:

- Im oberen, roten Bereich sind links die drei Ikonen, mit denen man zwischen den Ansichten, die im Präsentationsbereich der Komponente dargestellt werden, wechseln kann und mit der 4. Ikone kann man die zugehörige Onlinehilfe aufrufen; mittig steht der Name der Komponente; und rechts hat man die Möglichkeit, das Fenster zu minimieren, maximieren oder zu schließen.
- Im grünen Bereich (Präsentationsbereich) wird die eingestellte Ansicht angezeigt, in diesem Fall die Einstellungen. Parameter der Einstellungen können auch über die Eingänge einer Komponente gesetzt werden. Normalerweise hat eine Komponente Default-Werte für ihre Parameter. Gibt es einen Eingang dazu, überschreibt er den Defaultwert. Während der Ausführung einer Komponente können die Parameterwerte innerhalb der Komponente nicht geändert werden; sind sie jedoch mit Eingängen verbunden, können Sie auch während der Ausführung geändert werden.
- Der blaue Bereich beinhaltet eine Fortschrittsanzeige.
- Unterhalb der Komponente, im gelben Bereich, steht eine kurze Beschreibung, die man frei wählen kann.



Aktion
Verschlüsselung
Chiffre-Variante
ADFGVX
Substitutions-Kennwort
BAUM
Substitutions-Matrix
BAUMCDEFGHIJKLNOPQRSTVWXYZ0123456789
Transpositions-Kennwort
HAUS
Transposition Spaltenreihenfolge
2-1-4-3

Abb. 2.8: ADFGVX-Komponente mit geöffneten Einstellungen

Mit den ein- und ausgehenden Konnektoren (Dreiecke), links und rechts des roten Bereichs, kann die Komponente mit anderen Komponenten verbunden werden. Wie viele Ein- und Ausgänge eine Komponente hat, wird vom Programmierer festgelegt. Konnektoren können mit dem Flag *mandatory* versehen werden, womit vom Programmierer festgelegt wird, dass ein Konnektor verbunden sein muss. Ohne dieses Flag ist der Default, dass ein Konnektor nicht verbunden sein muss.

### 2.7.4 Der Lebenszyklus einer Komponente

Der Lebenszyklus einer Komponente (siehe Abb. 2.9) beginnt damit, dass sie – beim Laden in den Workspace – initialisiert wird. Dabei wird die Methode *Initialize()* aufgerufen. In dieser Methode können initiale Einstellungen vorgenommen werden, zum Beispiel das Laden einer Ansicht für eine Präsentation.

Sobald man die Komponente mit einem Klick auf *Starten* (im Englischen *Play*) ausführt, startet CT2 alle Komponenten auf der Arbeitsfläche. Dabei wird als erstes bei allen Komponenten auf der Arbeitsfläche die Methode *PreExecution()* ausgeführt. In dieser Methode können gewisse Vorarbeiten geleistet werden, wie das Aufbauen einer Verbindung zu einem Server oder das prüfen bestimmter Einstellungen.

Anschließend wird die Methode *Execute()* ausgeführt. Hier finden die eigentlichen Berechnungen der Komponente statt. Diese wird immer wieder neu aufgerufen sobald sich einer der Eingänge verändert.

Mit Betätigung der Schaltfläche *Stoppen* wird die laufende Ausführung der Arbeitsfläche beendet. Dadurch wird in jeder Komponente auf der Arbeitsfläche die Methode *Stop()* ausgeführt, welche interne Berechnungen unterbricht. CT2 beendet alle laufenden Threads und ruft abschließend die Methode *PostExecution()* auf. Diese Methode ist das Äquivalent zu *PreExecution()*, hier kann die Komponente in ihren Ursprungszustand zurück versetzt werden.

Entfernt man eine Komponente oder schließt die Arbeitsfläche, wird die Methode *Dispose()* ausgeführt, in der noch verwendete Ressourcen freigegeben werden können. Damit ist der Lebenszyklus einer Komponente beendet. Der Prozess startet von Neuem, sobald eine Komponente neu auf die Arbeitsfläche gezogen wird.

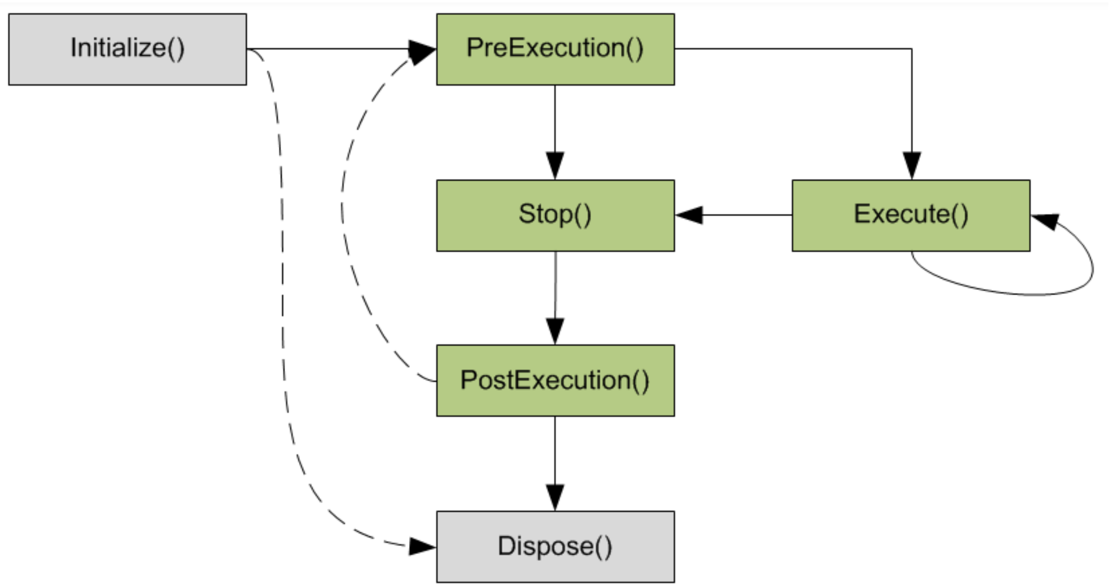


Abb. 2.9: Lebenszyklus einer Komponente [Cry]

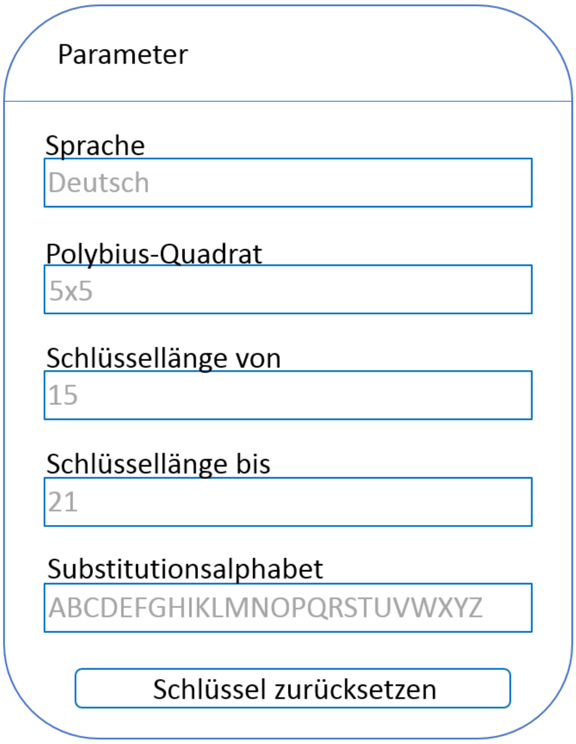
# 3 Konzept und Design der neuen Analyse-Komponente

In diesem Kapitel werden Konzept und Design der neu implementierten Analyse-Komponente vorgestellt. Darüber hinaus werden die an sie gestellten Anforderungen diskutiert. Dem CT2-Anwender wird eine Komponente zur Verfügung gestellt, mit der er verschlüsselte ADFGX- oder ADFGVX-Nachrichten brechen kann. Außerdem bekommt er in einer grafischen Präsentation dargestellt, welche und wie viele Schlüssel die Komponente bereits analysiert hat.

## 3.1 Aufbau der Komponente

Da die ADFGVX-Chiffre mit dem Polybius-Quadrat und der anschließenden Transposition eine sehr flexible Chiffre ist und in mehreren Varianten ausgeführt werden kann, soll die zu implementierende Komponente dies über die Einstellungen ebenfalls abbilden können. Dazu gehört in erster Linie die Größe des Polybius-Quadrats, sowie das Alphabet, das für die Substitution eingesetzt wird. Zur Auswahl für die Substitution sollen ein 5x5-, ein 6x6- und ein 7x7-Quadrat stehen, wovon jeweils eines in den Einstellungen, siehe Abb. 3.1, der Komponente ausgewählt werden kann. Die ersten beiden Größen sind die ursprünglich verwendeten der ADFGVX. Die Belegung des Alphabets soll frei wählbar sein. In Kapitel 5 Evaluation der ADFGVX-Analyse-Komponente wird diskutiert, wie sich die Wahl eines größeren Polybius-Quadrats auf die Sicherheit der Chiffre auswirkt. In Kapitel 2.6.3 wurde bereits erläutert, welchen Einfluss die Substitution in dem Verschlüsselungsverfahren der ADFGVX theoretisch hat. Zusätzlich zum Alphabet muss auch die Sprache eingestellt werden, in der die Nachrichten verschlüsselt wurden. Diese Einstellung ist wichtig, da die Häufigkeitsanalyse, um den richtigen Substitutionsschlüssel zu ermitteln, auf dem Koinzidenzindex basiert.

Laut Lasry [Las16] werden für das Entschlüsseln mehrere Nachrichten benötigt, die mit demselben Schlüssel verschlüsselt wurden. Um die Eingabe aller Nachrichten für den Anwender zu vereinfachen und flexibel zu halten, soll es nur ein Eingabefeld geben. In den Einstellungen kann ein Trennzeichen definiert werden, damit die Komponente die einzelnen Nachrichten auseinander halten kann. Als Standardwert wird das Semikolon verwendet. Ebenfalls in Kapitel 5 wird analysiert, wie sich die Anzahl und Länge der Nachrichten auf das Ziel, den richtigen Transpositionsschlüssel zu finden, auswirkt.



Sprache: Deutsch
Polybius-Quadrat: 5x5
Schlüssellänge von: 15
Schlüssellänge bis: 21
Substitutionsalphabet: ABCDEFGHIKLMNOPQRSTUVWXYZ

Abb. 3.1: Mockup der Einstellungen

Ist die Komponente mit der Berechnung des Schlüssels fertig, werden die eingegebenen Nachrichten mit diesem entschlüsselt ausgegeben. Jetzt sollte der entschlüsselte Text lesbar sein, wenn der gefundene Schlüssel und die zuvor eingestellte Sprache korrekt war. In Abbildung 3.2 ist eine schematische Darstellung der zu entwickelnden Komponente zu sehen.

## 3.2 Ursachen für falsche Analyseergebnisse

Einige Faktoren können das Ergebnis negativ beeinflussen:

- `Schlüssellänge`
- `Nachrichten sind mit unterschiedlichen Schlüsseln verschlüsselt`
- `Länge der Nachrichten (Gesamtlänge) zu kurz`
- `Einstellung der erwarteten Textsprache falsch`

Die Schlüssellänge wird dann zu einem Problem, wenn sie gerade ist. Das führt dazu das es Anordnungen der Spalten geben kann, wo die Spaltenpaare korrekt sind, aber nicht deren Reihenfolge. Dies wurde bereits ausführlicher in Kapitel 2.6.6.3 beschrieben. Sind die eingegebenen Nachrichten mit unterschiedlichen Schlüsseln verschlüsselt, funktioniert das Verfahren von Lasry eventuell nicht mehr. Sind neun

Abb. 3.2: Schematische Darstellung der Komponente

von zehn Nachrichten mit demselben Schlüssel verschlüsselt, kann das Verfahren trotzdem für diese neun Nachrichten den korrekten Transpositionsschlüssel finden.

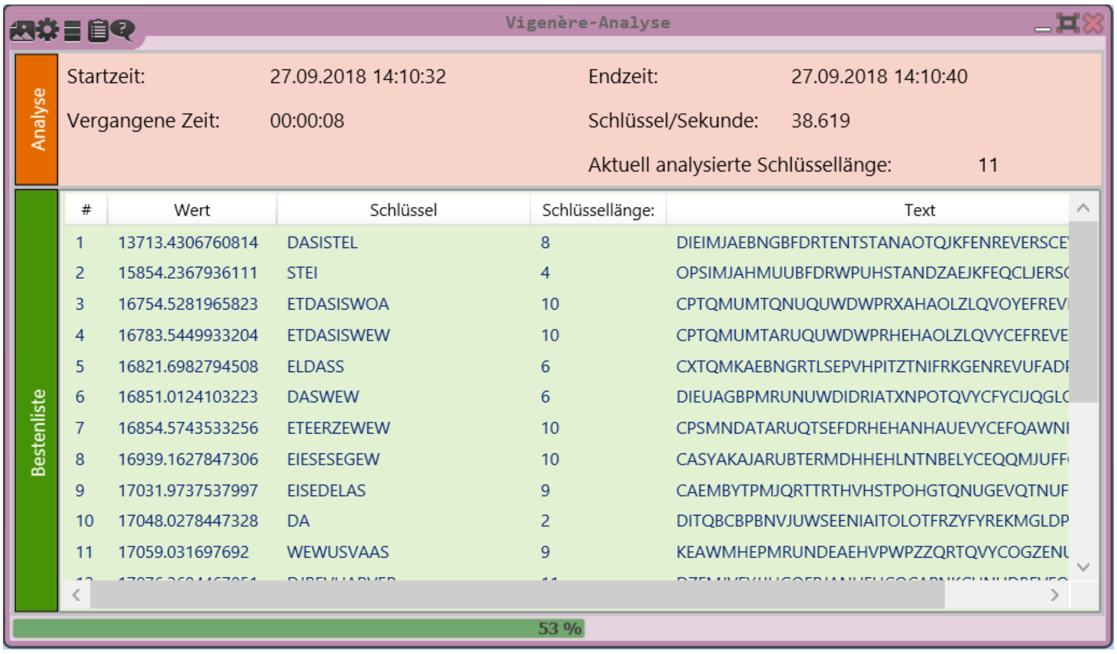
Die Länge der einzelnen Nachrichten bzw. die gesamte Länge aller eingegebenen Nachrichten spielt ebenfalls eine Rolle für die Transposition. Weniger Text bedeutet weniger Statistik. Eine falsche Einstellung der zu erwartenden Textsprache kann ebenfalls zu einem falschen Analyseergebnis führen. Europäische Sprachen ähneln sich aber so sehr in ihrer Häufigkeitsverteilung, dass der entschlüsselte Text trotzdem lesbar ist. Das liegt auch an der bereits in Kapitel 2.4 angesprochenen Tatsache, dass die zehn häufigsten Buchstaben ausreichen, um den Inhalt des Textes lesen zu können.

## 3.3 Präsentation der Komponente

Die Präsentationsansicht innerhalb der Komponente soll sich an bereits vorhandenen Komponenten orientieren. In diesem Fall dient der Vigenère-Analyzer als Vorlage (siehe Abb. 3.3).

Die Präsentationsansicht ist in zwei Bereiche aufgeteilt. Im oberen Bereich finden sich ein paar Kennzahlen zur durchgeführten Analyse, die für den Anwender relevant sind. Das ist zum Einen die Zeit die zum Entschlüsseln benötigt wird. Zum Anderen aber auch, wie viele Schlüssel probiert wurden. Im Gegensatz zum Vigenère-Analyzer werden beim ADFGVX-Analyzer die Anzahl der insgesamt versuchten und die Schlüssel pro Sekunde angezeigt. Die Angabe der aktuell analysierten Schlüssellänge wird wieder übernommen.

Im unteren Bereich, der Bestenliste, werden die aktuell besten Schlüssel angezeigt. Außerdem der entschlüsselte Beginn der Nachrichten, sowie der errechnete Wert der Kostenfunktion. Es kann passieren, dass das Verfahren einen besseren Schlüssel findet als den tatsächlich korrekten. Dieser könnte aber an einer Position darunter gelistet sein. Das Verfahren selbst arbeitet rein mathematisch und ist im Vergleich



Startzeit: 27.09.2018 14:10:32

Endzeit: 27.09.2018 14:10:40

Vergangene Zeit: 00:00:08

Schlüssel/Sekunde: 38.619

Aktuell analysierte Schlüssellänge: 11

| # | Wert | Schlüssel | Schlüssellänge: | Text |
|---|---|---|---|---|
| 1 | 13713.4306760814 | DASISTEL | 8 | DIEIMJAEBNGBFDRTENTSTANAOTQJKFENREVERSCE |
| 2 | 15854.2367936111 | STEI | 4 | OPSIMJAHMUUBFDRWPUHSTANDZAEJKFEQCLJERS |
| 3 | 16754.5281965823 | ETDASISWOA | 10 | CPTQMUMTQNUQUWDWPRXAHAOLZLQVOYEFREV |
| 4 | 16783.5449933204 | ETDASISWEW | 10 | CPTQMUMTARUQUWDWPRHEHAOLZLQVYCEFREVE |
| 5 | 16821.6982794508 | ELDASS | 6 | CXTQMKAEBNGRTLSEPVHPITZTNIFRKGENREVUFAD |
| 6 | 16851.0124103223 | DASWEW | 6 | DIEUAGBPMRUNUWDIDRIATXNPOTQVYCFYCIJQGL |
| 7 | 16854.5743533256 | ETEERZEWEW | 10 | CPSMNDATARUQTSEFDRHEHANHAUEVYCEFQAWN |
| 8 | 16939.1627847306 | EIESESEGEW | 10 | CASYAKAJARUBTERMDHHEHLNTNBELYCEQQMJUFF |
| 9 | 17031.9737537997 | EISEDELAS | 9 | CAEMBYTPMJQRTTRTHVHSTPOHGTQNUGEVQTNUF |
| 10 | 17048.0278447328 | DA | 2 | DITQBCBPBNVJUWSEENIAITOLOTFRZYFYREKMGLDP |
| 11 | 17059.031697692 | WEWUSVAAS | 9 | KEAWMHEPMRUNDEAEHVPWPZZQRTQVYCOGZEN |

Abb. 3.3: Vigenère-Analyzer aus CT2

zum Anwender nicht in der Lage, eine kognitive Leistung zu erbringen. Durch die Anzeige des entschlüsselten Beginns der Nachrichten ist es dem Anwender in so einem Fall leicht möglich, den korrekten Schlüssel zu finden. Zum Entschlüsseln wird automatisch der beste Schlüssel gewählt. Durch einen Doppelklick auf eine andere Zeile bzw. einen anderen Schlüssel kann man das aber ändern und der gewählte Schlüssel wird verwendet, um die Nachrichten ganz zu entschlüsseln und in die Ausgabe zu schreiben.

Ein Zyklus des Verfahrens funktioniert wie folgt:

1. Zufälligen Transpositionsschlüssel wählen
2. Nachrichten entschlüsseln
3. Kostenfunktion berechnen
4. Für jede valide Transformation Folgendes ausführen:
    a) Aktuellen Transpositionsschlüssel transformieren
    b) Nachrichten entschlüsseln
    c) Kostenfunktion berechnen
    d) Ist der Wert der Kostenfunktion besser, wird der Schlüssel behalten, die Substitution gelöst und mit 4.a die nächste Transformation durchgeführt.

    e) Ist der Wert schlechter, wird mit dem vorherigen Schlüssel mit 4.a die nächste Transformation durchgeführt (bei minimal schlechterem Wert kann anhand der Wahrscheinlichkeitsfunktion auch entschieden werden, mit diesem weiter zu arbeiten).

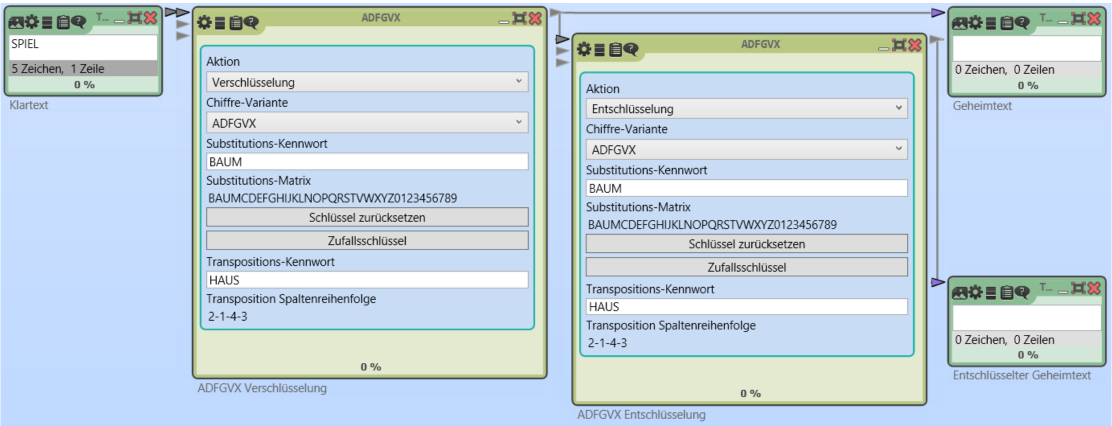
5. Solange der Wert der Kostenfunktion verbessert werden kann, wird Schritt 4 wiederholt.

Ein neuer Zyklus beginnt wieder bei Schritt 1.

## 3.4 Vorlagen und Hilfen

Für alle implementierten CT2-Komponenten werden Vorlagen und Hilfen bereit gestellt. In einer Vorlage wird die entwickelte Komponente mit weiteren Komponenten verknüpft, sodass sie direkt eingesetzt werden kann. So wie die Vorlage für die ADFGVX-Komponente in Abb. 3.4. Die Vorlagen sind über das Startcenter auszuwählen. In der dazugehörigen XML-Datei können Schlagwörter definiert werden, womit diese über die Suche leichter gefunden werden kann.

Zusätzlich werden Hilfen bereit gestellt, diese sind über die Ikone mit dem Fragezeichen (oben rechts im CT2-Fenster) zu öffnen. In der Onlinehilfe werden weitere Informationen zu den Komponenten, zu den Verfahren selbst, sowie weiterführende, externe Quellen für die detailliertere Informationsbeschaffung bereit gestellt. Darüber hinaus sind dort Angaben zum Entwickler der Komponente einzusehen.



SPIEL
5 Zeichen, 1 Zeile

Aktion: Verschlüsselung
Chiffre-Variante: ADFGVX
Substitutions-Kennwort: BAUM
Substitutions-Matrix: BAUMCDEFGHIJKLNOPQRSTVWXYZ0123456789
Transpositions-Kennwort: HAUS
Transposition Spaltenreihenfolge: 2-1-4-3

Aktion: Entschlüsselung
Chiffre-Variante: ADFGVX
Substitutions-Kennwort: BAUM
Substitutions-Matrix: BAUMCDEFGHIJKLNOPQRSTVWXYZ0123456789
Transpositions-Kennwort: HAUS
Transposition Spaltenreihenfolge: 2-1-4-3

0 Zeichen, 0 Zeilen
0 Zeichen, 0 Zeilen

Abb. 3.4: Vorlage zum Ver- und Entschlüsseln von ADFGVX-Nachrichten

# 4 Implementierung

Dieses Kapitel beschäftigt sich mit dem bereit gestellten Quellcode von Lasry und der Implementierung seines Verfahrens in die ADFGVX-Analyse-Komponente.

In Abschnitt 4.1 wird detailliert beschrieben, wie das Verfahren aufgebaut ist, um die Transposition zu brechen. Zusätzlich beinhaltet es die Übertragung des Quellcodes von Java nach C#, um es erstmal in einer Konsolenanwendung ausführbar zu machen.

In Abschnitt 4.2 wird beschrieben, welche Programmabschnitte angepasst wurden, um die Implementierung in CT2 zu ermöglichen.

Abschnitt 4.3 umfasst die geforderten Erweiterungen wie das Bearbeiten der bereits vorhandenen ADFGVX-Komponente, um zusätzlich 7x7-Matrizen mit der ADFGVX-Chiffre ver- und entschlüsseln zu können.

Im letzten Abschnitt 4.4 werden die Vorlagen und Hilfen vorgestellt, die in CT2 zu implementieren sind.

## 4.1 Der Quellcode von Lasry

Die zentrale Aufgabe dieser Bachelorarbeit ist es, das von Lasry entwickelte Verfahren in CT2 zu implementieren. Als Basis dazu diente der von ihm geschriebene Quellcode in Java. Um die Transposition zu brechen, nutzte er das heuristische Verfahren Simulated Annealing (siehe Kapitel 2.3) und den Koinzidenzindex (siehe Kapitel 2.5) als Kostenfunktion.

### 4.1.1 Programmanalyse

Lasry kann mit seinem Code beide Teile der ADFGVX-Chiffre, also sowohl die Transposition als auch die Substitution, brechen. Da aber in CT2 bereits eine sehr gute Lösung für die Substitution implementiert ist und der Schwerpunkt auf der Transposition liegt, wird auch nur dieser Teil beschrieben.

Der Programmablauf wird in Abb. 4.1 vereinfacht dargestellt. Die Anzahl der Zyklen, die das Programm durchläuft, sind frei wählbar. Die Methode **eval(key)** führt die Entschlüsselung der Transposition durch und berechnet die Kostenfunktion. Die Temperaturwerte sind Standardwerte, die aber angepasst werden können. Die Methode **randomize()** ändert die Reihenfolge der mögliche Transformationen zufällig und **transform(key)** transformiert den übergebenen Schlüssel. **Transforms** ist die Anzahl an durchführbaren Transformationen. Wie das Programm im Detail funktioniert, wird anschließend ausführlich erläutert.

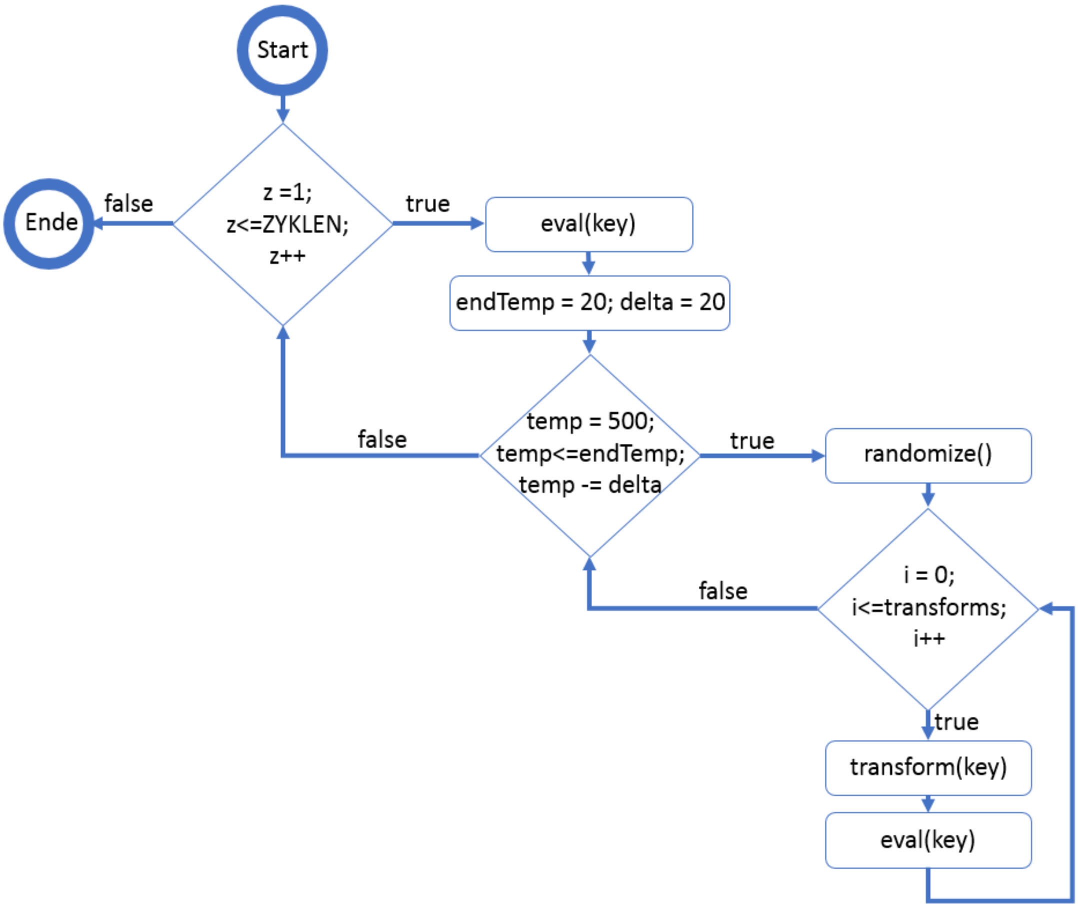


Abb. 4.1: Diese Grafik zeigt eine vereinfachte Darstellung von Lasrys Analyseverfahren

Folgende Klassen werden für die Analyse der Transposition benötigt:

1. `Algorithm`
2. `ADFGVX`
3. `Vector`
4. `ADFGVXVector`
5. `AlphabetVector`
6. `Alphabet36Vector`
7. `TranspositionTransformations`
8. `SimulatedAnnealing`

Die Klasse **Algorithm** ist der zentrale Einstiegspunkt. Einem Objekt dieser Klasse werden die Schlüssellänge und die verschlüsselten Nachrichten übergeben. Aus der Methode **SANgramsIC()** wird die Analyse gestartet.

Die Klasse **ADFGVX** repräsentiert einen ADFGVX-Schlüssel. Dieser beinhaltet den Transpositions- und den Substitutionsschlüssel. Diese Schlüssel werden auf unterster Ebene als Integer-Arrays dargestellt (bspw. DAECB = [3,0,4,2,1]). Dies wird während der Transformation eines Schlüssels deutlich. Einem Objekt dieser Klasse stehen, unter anderem, die Methoden zum Entschlüsseln einer Nachricht zur Verfügung.

Die Klassen **ADFGVXVector**, **AlphabetVector** und **Alphabet36Vector** erben von der Klasse **Vector**. Einem Objekt dieser Klasse bekommt 4 Werte übergeben. Die Textlänge und drei weitere Boolesche Werte, mit denen die Statistik berechnen wird. Je nach verwendeter Klasse beinhaltet das Objekt ein anderes Alphabet.

- `ADFGVXVector: ADFGVX`
- `AlphabetVector: ABCDEFGHIJKLMNOPQRSTUVWXYZ`
- `Alphabet36Vector: ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789`

Die Klasse **TranspositionTransformations** übernimmt die Transformation des Transpositionsschlüssels. Ihr werden, neben der Schlüssellänge, noch drei weitere Boolesche Werte übergeben:

- `SLIDES`
- `SWAPS`
- `INVERSIONS`

Ein Objekt dieser Klasse erzeugt ein `HashSet` aus Permutationen des Transpositionsschlüssels. Die Anzahl der Permutationen in diesem `HashSet` wird über die drei Booleschen Werte gesteuert.

Die Klasse **SimulatedAnnealing** beinhaltet eine Methode, um zu entscheiden, ob der neue Schlüssel angenommen oder verworfen wird. Ist der Wert des neuen besser, wird er angenommen. Ist er schlechter, wird mittels einer Wahrscheinlichkeitsfunktion entschieden, ob er angenommen oder verworfen wird.

Das Verfahren beginnt damit, dass ein Objekt der Klasse **Algorithm** erzeugt wird. Die gewählte Schlüssellänge wird als Integer und die verschlüsselten Nachrichten als String-Array übergeben. Für jede verschlüsselte Nachricht wird ein Objekt der Klasse **ADFGVXVector** erzeugt und alle verschlüsselten Nachrichten werden in einem **ADFGVXVector**-Array gespeichert. Als einfache Integer Werte werden die Gesamtlänge aller Nachrichten und die Länge der längsten Nachricht gespeichert. Abschließend wird ein weiteres Objekt der Klasse ADFGVXVector und zwei Objekte der Klasse Alphabet36Vector erzeugt. Nachfolgend alle Objekte aufgelistet:

- `ciphers: ADFGVXVector[] (alle verschlüsselten Nachrichten)`
- `totalPlainLength: Integer (Gesamtlänge aller Nachrichten)`
- `maxPlainLength: Integer (Länge der längsten Nachricht)`
- `allPlain: Alphabet36Vector (Text der Länge` *totalPlainLength*`)`
- `plain: Alphabet36Vector (Text der Länge` *maxPlainLength*`)`
- `interim: ADFGVXVector (ADFGVX-Text der Länge` *maxPlainLength·2*`)`

Mit dem erzeugten **Algorithm**-Objekt kann die Methode **SANgramsIC()** gestartet werden. Zu Beginn werden vier Objekte erzeugt, die für die Berechnung benötigt werden:

- `keepTranspositionKey: AlphabetVector (Text der Länge des Schlüssels)`
- `newTranspositionKey: AlphabetVector (Text der Länge des Schlüssels)`
- `key: ADFGVX (beinhaltet die Schlüsselinformationen)`
- `t: TranspositionTransformations`

Wie oft innerhalb der Methode versucht wird, die Transposition zu brechen, ist abhängig von drei Faktoren:

1. `Anzahl der Simulated Annealing Wiederholungen (Zyklen)`
2. `Starttemperatur und Abkühlung des Simulated Annealing`
3. `Anzahl an Wiederholungen, berechnet anhand der Schlüssellänge`

Die Anzahl der Zyklen, die Starttemperatur und der Abkühlungsverlauf sind frei wählbar. Die dritte Variable berechnet sich auf Basis der Schlüssellänge und ist abhängig von den gesetzten Booleschen Werten **slides**, **swaps** und **inversions**. Diese berechnen sich einzeln wie folgt(k = Schlüssellänge):

`inversions:`

$$k \cdot k + 2$$

`slides:`

$$k \cdot k \cdot (k-1) + k$$

`swaps:`

$$\sum_{i=1}^{\lfloor \frac{k}{4} \rfloor} \sum_{j=1}^{k-2i} j \;+\; \sum_{i=1}^{3} \sum_{j=0}^{k-(3i+1)} \sum_{l=1}^{k-(3i+j)} 2l \;+\; (k \cdot (k-1) \cdot (k-2))$$

`!swaps und !slides:`

$$\sum_{j=1}^{k-2i} j \;+\; \sum_{i=1}^{3} \sum_{j=0}^{k-(3i+1)} \sum_{l=1)}^{k-(3i+j} 2l \;+\; (k \cdot (k-1) \cdot (k-2))$$

Die erzeugten Elemente werden anschließend durch einen Comparator verglichen und Duplikate entfernt. Wie viel Durchläufe sich dadurch ergeben ist in Tabelle 4.1 für die Schlüssellängen 5 - 21 aufgelistet. Nachfolgend repräsentiert `t.size()` diese Anzahl an Wiederholungen.

Ein Zyklus der Analyse beginnt damit, dass mit **key.randomTranspositionKey()** ein zufälliger Transpositionsschlüssel erzeugt wird. Mit dem Objekt **key**, in dem dieser Transpositionsschlüssel hinterlegt ist, wird die Methode **eval()** aufgerufen.

In **eval()** werden alle Nachrichten nacheinander entschlüsselt. Dafür wird folgende Methode aufgerufen:

```
key.decode(cipher, interim, plain)
```

**key.decode(cipher, interim, plain)** bekommt eine der Nachrichten (**cipher**) und die vorher erstellten **interim** und **plain** Objekte übergeben. Die Nachricht **cipher** wird mit dem Transpositionsschlüssel aus **key** transponiert und die Ausgabe in **interim** abgelegt. Danach wird **interim** substituiert und in **plain** ausgegeben. Abschließend wird **plain** an das Objekt **allPlain** angehangen. Dies wird für alle Nachrichten durchgeführt.

Über **allPlain** wird jetzt mit der Methode **stats()** der Koinzidenzindex von Mono-, Bi- und Trigrammen berechnet. Diese Werte werden in den Variablen **ic1**, **ic2** und **ic3** abgelegt. Der Rückgabewert der Methode **eval()** berechnet aber nur aus **ic1** und **ic2** wie folgt:
`6.000 · allPlain.ic1 + 180.000 · allPlain.ic2`

| Schlüssellänge | Zyklen | Abkühlungen | t.size() | Formel | Durchläufe |
|---|---|---|---|---|---|
| 5 | 100 | 25 | 57 | $100 \cdot 25 \cdot 57 + 100$ | 142.600 |
| 6 | 100 | 25 | 116 | $100 \cdot 25 \cdot 116 + 100$ | 290.100 |
| 7 | 100 | 25 | 203 | $100 \cdot 25 \cdot 203 + 100$ | 507.600 |
| 8 | 100 | 25 | 333 | $100 \cdot 25 \cdot 333 + 100$ | 832.600 |
| 9 | 100 | 25 | 505 | $100 \cdot 25 \cdot 505 + 100$ | 1.262.600 |
| 10 | 100 | 25 | 729 | $100 \cdot 25 \cdot 729 + 100$ | 1.822.600 |
| 11 | 100 | 25 | 1.016 | $100 \cdot 25 \cdot 1.016 + 100$ | 2.540.100 |
| 12 | 100 | 25 | 1.390 | $100 \cdot 25 \cdot 1.390 + 100$ | 3.475.100 |
| 13 | 100 | 25 | 1.836 | $100 \cdot 25 \cdot 1.836 + 100$ | 4.590.100 |
| 14 | 100 | 25 | 2.373 | $100 \cdot 25 \cdot 2.373 + 100$ | 5.932.600 |
| 15 | 100 | 25 | 3.010 | $100 \cdot 25 \cdot 3.010 + 100$ | 7.525.100 |
| 16 | 100 | 25 | 3.784 | $100 \cdot 25 \cdot 3.784 + 100$ | 9.460.100 |
| 17 | 100 | 25 | 4.656 | $100 \cdot 25 \cdot 4.656 + 100$ | 11.640.100 |
| 18 | 100 | 25 | 5.656 | $100 \cdot 25 \cdot 5.656 + 100$ | 14.140.100 |
| 19 | 100 | 25 | 6.793 | $100 \cdot 25 \cdot 6.793 + 100$ | 16.982.600 |
| 20 | 100 | 25 | 8.121 | $100 \cdot 25 \cdot 8.121 + 100$ | 20.302.600 |
| 21 | 100 | 25 | 9.569 | $100 \cdot 25 \cdot 9.569 + 100$ | 23.922.600 |

Tab. 4.1: Anzahl an Durchläufen, exemplarisch für 100 Zyklen und 25 Abkühlungsvorgänge

Dieser Rückgabewert bewertet den Transpositionsschlüssel und dient als erster Kostenwert, bis ein anderer Transpositionsschlüssel einen höheren Wert erreicht.

Im nächsten Schritt wird die Starttemperatur und die Abkühlungsgeschwindigkeit festgelegt. Die Standardwerte hierfür sind:

- `Starttemperatur = 500`
- `Abkühlungsgeschwindigkeit = 20`
- `Endtemperatur = 20`

Basierend auf diesen Standardwerten wird die Temperatur in 25 Iterationen von 500 auf 20 herunter gekühlt.

Zu Beginn einer solchen Iteration werden die Permutationen aus dem Objekt **t** zufällig in ihrer Reihenfolge vertauscht und in einem zweidimensionales Array **list[,]** gespeichert. Dies wird benötigt, damit in jeder Iteration unterschiedliche Schlüssel

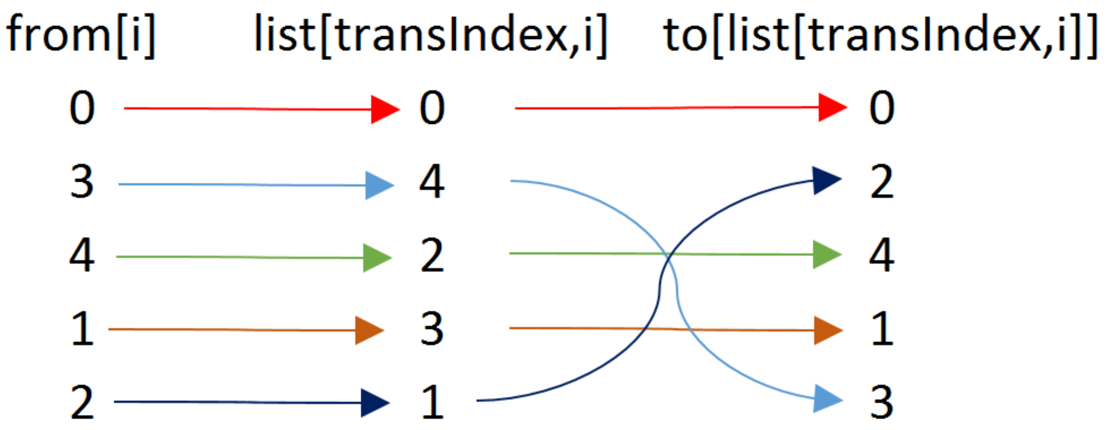


Abb. 4.2: Transformation eines Schlüssels der Schlüssellänge 5

in anderer Reihenfolge analysiert werden. In jeder dieser Abkühlungsiterationen wird genau **t.size()**-mal folgender Programmabschnitt ausgeführt (`i` = Iteration der for-Schleife über **t.size()**):

```
keepTranspositionKey.copy(key.transpositionKey)
t.transform(keepTranspositionKey.v,newTranspositionKey.v
    ,keyLength,i)
key.setTranspositionKey(newTranspositionKey)
double newScore = eval(key)
if (SimulatedAnnealing.accept(newScore, score, temp)
score = newScore
else
key.setTranspositionKey(keepTranspositionKey)
```

Der zu Beginn des Zyklus gesetzte Transpositionsschlüssel wird nach **keepTranspositionKey** kopiert, über **t.transform()** transformiert und in **newTranspositionKey** gespeichert. Diese Transformation geschieht wie folgt:

```
public void transform(int[] from, int[] to, int length,
    int transIndex)
for (int i = 0; i < length; i++)
to[list[transIndex, i]] = from[i]
```

Wie in Abb. 4.2 grafisch dargestellt, bestimmt die Permutation aus dem Objekt **list**, an der Stelle **transIndex**, von wo nach wo der Inhalt transformiert wird.

Der transformierte Schlüssel wird im Objekt **key** gesetzt und anschließend evaluiert. Das Simulated Annealing prüft im nächsten Schritt, ob der neue Schlüssel akzeptiert oder verworfen wird. Ist der **newScore** größer als **score** gibt die Methode **accept(newScore, score, temp)** `true` zurück.

Ist **newScore** kleiner als **score**, wird

$$e^{\frac{newScore-score}{temp}}$$

berechnet und geprüft, ob dieser Wert kleiner 0,0085 ist. Ist das der Fall, gibt die Methode `false` zurück. Ist der Wert größer als 0,0085, wird eine zufällige Zahl zwischen 0 und 1 erzeugt. Wenn die zufällige Zahl kleiner ist, wird `true` zurück gegeben, sonst `false`. Dies führt dazu, dass auch schlechtere Schlüssel gewählt werden können.

Die Rückgabe `true` führt dazu, dass der aktuelle **score** mit dem **newScore** überschrieben wird. Bei der Rückgabe `false`, wird der Transpositionsschlüssel in **key** auf den Schlüssel **keepTranspositionKey** zurück gesetzt.

Insgesamt werden in einem Zyklus `1 + Abkühlungen · t.size()` Evaluierungen durchgeführt, wie vereinfacht in Abb. 4.1 dargestellt ist. Der Anwender selbst kann entscheiden, wie viele Zyklen das Programm ausführen soll.

### 4.1.2 Übersetzung von Java nach C#

Die Sprachen Java und C# sind sich sehr ähnlich. Deshalb konnte der Quellcode zum größten Teil übernommen werden. Meist waren es nur kleine Änderungen, weil die Syntax in C# eine andere ist wie unter Java. In Tabelle 4.2 sind einige dieser einfachen Beispiele aufgelistet.

Ein Teil musste komplett neu implementiert werden: Lasry benutzte in Java die Schnittstelle **Runnable**, um mehrere Threads gleichzeitig starten zu können. Dazu schrieb er sich seine eigene Klasse ***Runnables***. In C# wurde dafür die Schnittstelle **Thread** benutzt.

Unter Java verwendete Lasry für die Menge an Transpositionen ein **Set**, welches dem **HashSet** in C# gleicht. Allerdings funktionierte der von ihm benutze Comparator nicht mehr. Um den Comparator selbst zu definieren, muss beim HashSet in C# von der Klasse **IEqualityComparator** geerbt werden. Dies führte aber trotz mehrmaligem Debuggen nicht zum gewünschten Ergebnis. Daher wurde ein Workaround geschaffen und bereits beim Hinzufügen der Elemente ein Vergleich durchgeführt. Existiert das Objekt bereits im HashSet, wird es gar nicht erst hinzugefügt.

Zum Schluss wurde eine neue Klasse geschrieben, die eine Textdatei mit den verschlüsselten Nachrichten laden kann und anschließend das Verfahren startet. Darüber hinaus wurde eine **Logger**-Klasse hinzugefügt, um die Ausgaben in Dateien umzuleiten.

Mit der so entstandenen Konsolenanwendung (adfgvxCT2) wurde auch die Evaluation durchgeführt. Wie viele Evaluierungen durchgeführt wurden, ist in Kapitel 5 dokumentiert.

| **Beschreibung** | |
|---|---|
| **Java** | **C#** |
| **Zugriff auf Char i in String (String s)** | |
| s.charAt(i) | s[i] |
| **Erzeugen eines zweidimensionalen Arrays** | |
| int [][] list | int [,] list |
| **Aufruf der Dimensionen in einem zweidimensionalen Array** | |
| list.length \ list[].length | list.GetLength(0) \ list.GetLength(1) |
| **Kopieren eines eindimensionalen Arrays** | |
| System.arraycopy(. . . ) | Array.Copy(. . . ) |
| **foreach-Schleifenkopf** | |
| for(char c : chars) | foreach(char c in chars) |
| **Ein Array a mit Nullen füllen** | |
| Array.fill(a, 0) | for (i=0;i<a.length;i++){a[i]=-1;} |
| **Eigenschaften/Methoden werden groß geschrieben, z.B. Länge eines Objekts (String)** | |
| s.length | s.Length |
| **Regex replace in einem String** | |
| toString().replaceAll(“ “, ““) | Regex.Replace(ToString(), “ “, ““) |
| **Aufruf eines erbenden Konstruktors** | |
| super(. . . ); | : base(. . . ) { } |

Tab. 4.2: Unterschiede zwischen Java und C#

## 4.2 Z1: Implementierung in CrypTool 2

Mit der Implementierung in CT2 wurde der Code für das Verfahren an einigen Stellen überarbeitet. Dazu gehörten überwiegend das Umbenennen vieler Variablen oder Methoden, sowie das Entfernen von nicht benötigtem Quellcode. Zusätzlich wurde eine Hilfsklasse **ThreadingHelper** geschrieben, die Race-Conditions verhindern soll. Abschließend wurde die Präsentation mittels Windows Presentation Foundation (WPF) hinzugefügt.

### 4.2.1 Refactoring

Unter dem Begriff Refactoring versteht man eine manuelle oder automatisierte Optimierung von Quellcode. Damit können die Lesbarkeit, Verständlichkeit, Wartbarkeit und/oder Erweiterbarkeit verbessert werden.[Wika]

#### 4.2.1.1 Lesbarkeit und Verständlichkeit

Umbenannt wurden Objekte und Methoden in folgenden Klassen:

- `Algorithm`
- `Vector`

In der Klasse **Algorithm** gibt es ein Objekt Namens **interim** der Klasse **ADFGVX-Vector**. Dieses wird (siehe Kapitel 4.1.1) als Zwischenspeicher, zum Entschlüsseln der Nachrichten, benötigt. Da dieses Objekt immer noch einer ADFGVX-Nachricht entspricht, nur mit rückgängig gemachter Transposition, wird es in **interimCiphertext** umbenannt.

Das String-Array **cipherStr** beinhaltet alle verschlüsselten Nachrichten. Dies wurde vereinheitlicht in **messages**.

Die Variablen **ic1** und **ic2** stehen für den Koinzidenzindex der Mono- und Bigramme. Die formal korrekte Abkürzung für Koinzidenzindex lautet IoC (Index of Coincidence). Daher wurden sie umbenannt in **IoC1** und **IoC2**.

Das Objekt **t** der Klasse TranspositionTransformations wurde in **transformations** umbenannt. Da ein einzelner Buchstabe als Variablen- oder Objektname nicht aussagekräftig genug ist.

In der Klasse **Vector** wurden ebenfalls die Namen der Variablen **ic1** und **ic2** zu **IoC1** und **IoC2** geändert. Außerdem wurde das Integer-Array **v** in **TextInInt** umbenannt, weil durch dieses Integer-Array der Text in Integer-Werten interpretiert wird.

### 4.2.1.2 Entfernen von nicht erreichbarem Code

Aus folgenden Klassen wurde nicht erreichbarer Code entfernt:

- `SimulatedAnnealing`
- `Vector`
- `adfgvx`

In der Klasse **SimlulatedAnnealing** wurde eine zweite **accept()** Methode entfernt, die für die Entschlüsselung der Transposition nicht benötigt wird.

In der Klasse **Vector** gab es viele Methoden, die zum Entschlüsseln der Substitution verwendet wurden. Da die Substitutionsanalyse in CT2 von einer schon bestehenden Komponente durchgeführt wird, werden diese Methoden nicht mehr benötigt. Folgende Methoden wurden entfernt, ohne genauer auf deren Funktionsumfang einzugehen:

```
eval(int n)
eval(Vector compare)
fromSubstitutionKey(String keyS)
random()
randomValues()
randomSkewed()
randomDifferentValues()
score1(Vector reference)
score2(Vector reference)
score3(Vector reference)
printStats(bool printStream)
printMonograms()
printBigrams()
printTrigrams()
finalizeLength()
ReadTextFile(String fileName, int alloc, bool clean)
```

Lasry berechnete zusätzlich zu den Koinzidenzindizes für Mono- und Bigramme auch einen für Trigramme. In der Kostenfunktion der Transpositionsanalyse spielt dieser aber keine Rolle. Deshalb wurde die Berechnung ebenfalls komplett entfernt.

In der Klasse **adfgvx** sind ebenfalls einige Methoden, die Lasry für die Substitutionsanalyse verwendete. Auch diese werden in CT2 nicht benötigt. Folgende Methoden wurden entfernt, ohne genauer auf deren Funktionsumfang einzugehen:

```
decode(ADFGVXVector cipher, Alphabet36Vector plain)
encode(Alphabet36Vector plain, ADFGVXVector interim,
    ADFGVXVector cipher)
encode(Alphabet36Vector plain, ADFGVXVector cipher)
```

```
encodeSubstitution(Alphabet36Vector interim,
    ADFGVXVector cipher)
```

Folgende Hilfsklassen wurden durch das Entfernen des unerreichbaren Codes nicht mehr benötigt:

- `stats`
- `digitsVector`
- `Languages`

Darüber hinaus wurden weitere Variablen, sowie *Getter* und *Setter* entfernt die aufgrund des Refactoring nicht mehr benötigt werden.

#### 4.2.1.3 Änderung der Berechnung des Koinzidenzindex

Lasry setzte für die Berechnung seines Koinzidenzindex nicht die in Kapitel 2.5 beschriebene Formel ein. Er berechnete den Koinzidenzindex wie folgt:

$$IoC = \sum_{i=1}^{N} \left(\frac{Anzahl_i}{Textlaenge}\right)^2$$

wobei $N$ die Anzahl an Zeichen des Alphabets darstellt, $Anzahl_i$ die Anzahl des Zeichens i und $Textlaenge$ die Gesamtlänge des Textes.

Der Koinzidenzindex ist eine Approximation dieser Formel und die Ergebnisse weichen leicht voneinander ab. Um in der CT2-Komponente den exakten Koinzidenzindex angeben zu können, wurde die Berechnung angepasst.

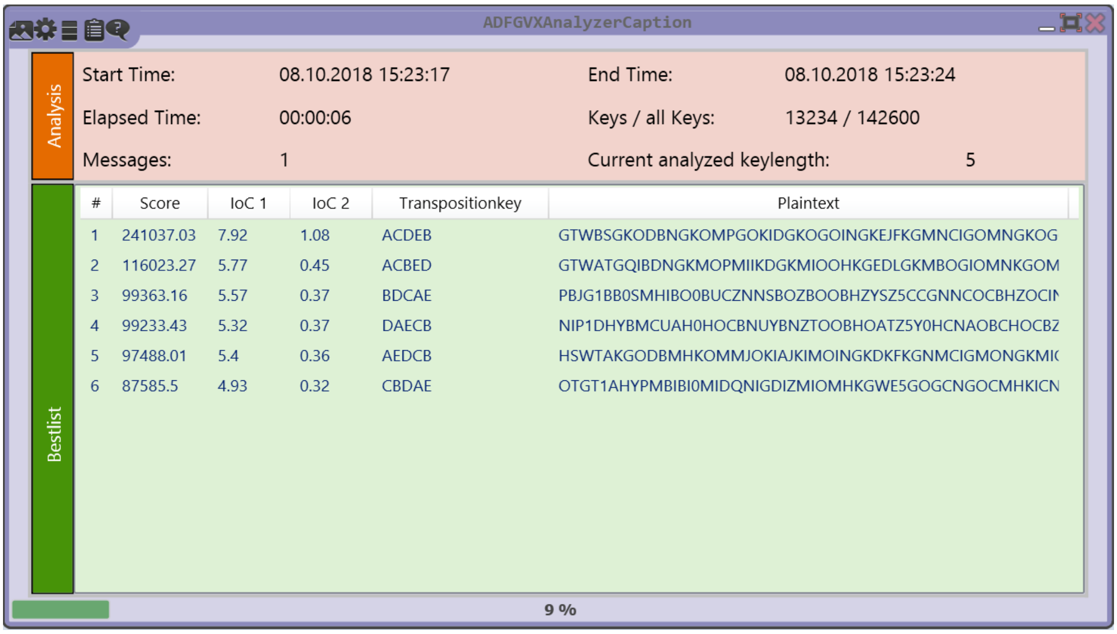
#### 4.2.1.4 Vermeidung von Race Conditions

Das Verfahren ist so aufgebaut, dass mehrere Threads gestartet werden können, um das Finden eines Schlüssels zu parallelisieren. Die gleichzeitig laufenden Threads haben dabei Zugriff auf eine Zählvariable (Anzahl der Entschlüsselungen) und die Variable, die den besten Kostenwert beinhaltet.

Dabei kann es dazu kommen, dass zwei Threads gleichzeitig eine dieser Variablen verändern wollen. Solch eine unbeaufsichtigte Wettlaufsituation nennt man Race Condition. Diese gilt es zu vermeiden.[Mic]

Um das zu verhindern gibt es in C# das Schlüsselwort **lock**. Dadurch wird ein Abschnitt als kritisch betrachtet und gesperrt, sobald ein Thread diesen betritt.[Mic]

Der Nachteil dabei ist, dass ein anderer Thread, der ebenfalls diesen Abschnitt ausführen will, warten muss, bis dieser Abschnitt wieder freigegeben ist. So behindern bzw. verlangsamen sich die Threads gegenseitig. Daher ist es ratsam, sparsam mit kritischen Abschnitten umzugehen und sie so klein wie möglich zu halten.



Start Time: 08.10.2018 15:23:17
End Time: 08.10.2018 15:23:24
Elapsed Time: 00:00:06
Keys / all Keys: 13234 / 142600
Messages: 1
Current analyzed keylength: 5

| # | Score | IoC 1 | IoC 2 | Transpositionkey | Plaintext |
|---|---|---|---|---|---|
| 1 | 241037.03 | 7.92 | 1.08 | ACDEB | GTWBSGKODBNGKOMPGOKIDGKOGOINGKEJFKGMNCIGOMNGKOG |
| 2 | 116023.27 | 5.77 | 0.45 | ACBED | GTWATGQIBDNGKMOPMIIKDGKMIOOHKGEDLGKMBOGIOMNKGOM |
| 3 | 99363.16 | 5.57 | 0.37 | BDCAE | PBJG1BB0SMHIBO0BUCZNNSBOZBOOBHZYSZ5CCGNNCOCBHZOCI |
| 4 | 99233.43 | 5.32 | 0.37 | DAECB | NIP1DHYBMCUAH0HOCBNUYBNZTOOBHOATZ5Y0HCNAOBCHOCBZ |
| 5 | 97488.01 | 5.4 | 0.36 | AEDCB | HSWTAKGODBMHKOMMJOKIAJKIMOINGKDKFKGNMCIGMONGKMI |
| 6 | 87585.5 | 4.93 | 0.32 | CBDAE | OTGT1AHYPMBIBI0MIDQNIGDIZMIOMHKGWE5GOGCNGOCMHKICN |

Abb. 4.3: ADFGVX-Analyse-Komponente (berechnete hier gerade einen Transpositionsschlüssel der Länge fünf: ACDEB)

Mit der Variable **decryptions** (Long-Array) der Klasse ThreadingHelper wollen wir die Anzahl der gesamten Entschlüsselungsversuche über alle Threads zusammen zählen, um sie in der grafischen Oberfläche anzeigen zu können. Beim Starten des Verfahrens ist bekannt, wie viele Threads gestartet werden sollen. In dem ThreadingHelper wird dieses Long-Array erzeugt mit der Anzahl an Threads. Jeder Thread schreibt in sein eigenes Element dieses Arrays. Zu Beginn jedes Abkühlungsdurchlaufs werden die Elemente aus diesem Array addiert und an die grafische Oberfläche weiter gereicht:

```
lock (threadingHelper.decryptionsLock)
foreach (long d in threadingHelper.decryptions)
alldecryptions += d;
```

### 4.2.2 Interaktive Präsentations-Ansicht

Um den Nutzern von CT2 die Bedienung zu erleichtern, sind die Präsentations-Ansichten von fast allen Analyse-Komponenten gleich aufgebaut. Deshalb orientiert sich auch die neue ADFGVX-Analyse-Komponente an dieser Ansicht (siehe Abb. 4.3)

Der Bereich **Analysis** umfasst 6 Basisinformationen zum gestarteten Verfahren. Dem Benutzer werden die Startzeit, Dauer und Endzeit angezeigt. Ergänzt wird dies durch die Anzahl an Nachrichten, die aktuell analysierte Schlüssellänge und wie viele Schlüssel bereits berechnet wurden bzw. wie viele insgesamt berechnet werden. Parallel zum Verhältnis `berechnete Schlüssel / Schlüssel insgesamt` läuft am unteren Bildrand ein Fortschrittsbalken mit.

Der Bereich **Bestlist** listet in 6 Spalten Informationen zu den gefundenen Schlüsseln auf:

- `Score: Kostenwert aus` $6000 \cdot IoC1 + 180000 \cdot IoC2$
- `IoC1: Koinzidenzindex von Monogrammen`
- `IoC2: Koinzidenzindex von Bigrammen`
- `Transpositionkey: ein möglicher Transpositionsschlüssel`
- `Plaintext: Substitution mit Standardalphabet`[1]

Die Ansicht aktualisiert sich automatisch und gibt, sobald es einen neuen besten Eintrag gibt, ihre Informationen über die drei Ausgabe-Konnektoren weiter an andere Komponenten.

Abb 4.4 zeigt, welche Textausgaben ausgegeben werden. Mit einem Doppelklick auf einen Eintrag der **Bestlist** kann der Benutzer interaktiv auch Informationen von Schlüsseln an die Ausgaben weitergeben, die nicht an erster Stelle stehen.

## 4.3 Z2: Implementierung der geforderten Erweiterungen

Die ADFGVX-Analyse-Komponente soll so erweitert werden, dass auch 7x7-Matrizen verarbeitet werden können. In CT2 gab es bisher keine Komponente, die das ADFGVX-Verfahren mit einer 7x7-Polybius-Matrix durchführen kann. Daher musste die bereits vorhandene ADFGVX-Chiffrier-Komponente ebenfalls um diese Funktion erweitert werden.

Die notwendigen Anpassungen dieser Komponente sind vergleichsweise gering, da sie bereits sehr modular aufgebaut ist. Es mussten lediglich ein paar Konstanten hinzugefügt werden, um in den Einstellungen die entsprechenden Optionen auswählen zu können (siehe Abb. 4.5).

Eine sehr einfache Umsetzung der Anforderungen an die Bachelorarbeit wäre, die Klassen **ADFGVXVector** und **Alphabet36Vector** statisch zu verändern. Dies könnte erfolgen durch die Angabe eines 49-Zeichen Alphabets in **Alphabet36Vector** und eines 7-Zeichen Alphabets in **ADFGVXVector**.

Diese Umsetzung wäre aber eine sehr einfache Form der gewünschten Erweiterung. Eleganter ist es, über die Einstellungen der Komponente dynamisch ein Substitutionsalphabet eingeben zu können, das vor der Ausführung auf seine Länge geprüft

---

[1] Das Standardalphabet eines Polybius-Quadrat der Größe 6x6 ist `ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789` – ohne ein Schlüsselwort (siehe Abb. 2.1).

```
GTWBSGKODBNGKOMPGOKIDGKOGOINGKEJFKGMNCIGOMNGKOGONGTGSBAGSFKGTDBNMK
OFGSJBAUKJSWGKUFBOGAGOIGSBUGOGTWBSGKODBNGKOTUCGEMJPNZLBGKOIBOZIGWPGJO
NKEJGTJPNZTEJGKUFSBCTTGONBIGTKDWBNFDKUVKGNGOBOFGSOTUCGEMGOBCHFGSAGKIGG
KOHCJSDBOOMBDNCFTKGBNNGBCHFGOWBIGOCOFHCJSFBDKUZCSTUBFUFGDTEJSGKOGSUPO
KVPSFBTJBCTFBTJPNZWBSFIGTBGIUCOFIGTQBNUGUFGOOKDMBNUGOWKOUGSTPNNUGGTKDM
OKTUGSOFGOPHGOFKGTUCAGWBGSDGOGKOINCGEMFBTTUPOKFBTGKOGTEJGKUAGDGSMUGG
TWBSTPJCGATEJIGSBFGCOFJBUUGMGKOGOBTUFSCDTUGNNUGGTFGSTEJSGKOGSKOGKOGGEM
GTGKOGSWGSMTUBUUCOFFBEJUGGKOICUGTINBUUGTTUCGEMTWBGSTEJBFGGTZCVGSASGOO
GO
```

553 Zeichen, 1 Zeile

0 %

```
ACDEB
```

5 Zeichen, 1 Zeile

0 %

```
Starting Thread: 1
Task: 1
cycle: 1
temp: 500
trans key: ACDEB
bestOverall: 241037,031213146
IoC1 and IoC2: 7,91794428283146 107,516314175643
Task id: 1 starting with cycle: 10
Task id: 1 starting with cycle: 20
Starting Thread: 1
Task: 1
cycle: 1
temp: 500
trans key: ACDEB
bestOverall: 241037,031213146
IoC1 and IoC2: 7,91794428283146 107,516314175643
Task id: 1 starting with cycle: 10
Task id: 1 starting with cycle: 20
```

442 Zeichen, 19 Zeilen

0 %

Abb. 4.4: Beispielausgaben der ADFGVX-Analyse-Komponente (Text der noch zu substituieren ist, Transpositionsschlüssel und Logdatei)

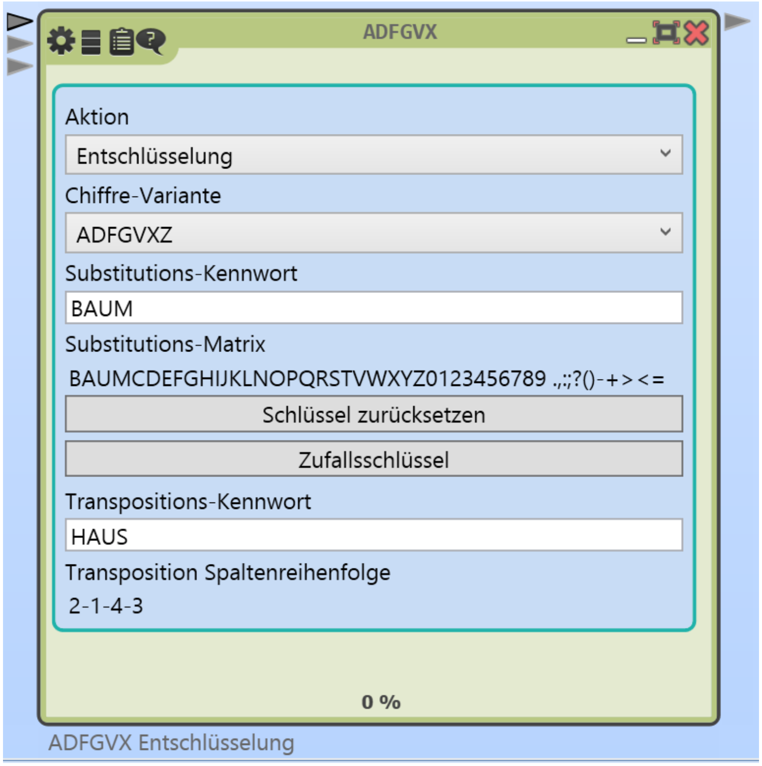
wird. Erlaubt wären nur die Längen $N^2$. Wobei $N$ die Anzahl der verfügbaren Zeichen im Polybius-Quadrat darstellen. Zusätzlich kann der Benutzer die Zeichen für das Polybius-Quadrat bestimmen.

Die Klasse **Alphabet36Vector** wurde umbenannt in **AlphabetNVector** und die Klasse **ADFGVXVector** in **CiphertextAlphabetVector**. Beide erhalten ihr Alphabet aus den Einstellungen der Analyse-Komponente, die vom Benutzer bearbeitet werden können.

So könnte man die ADFGVX-Chiffre auch mit anderen Zeichen ausführen. Allerdings gibt es in CT2 aktuell keine Komponente, die die ADFGVX-Chiffre mit anderen Zeichen umsetzt.

## 4.4 Z3: Vorlagen und mehrsprachige Hilfen

Vorlagen und Hilfen dienen dem Benutzer dazu, die Bedienung zu erleichtern und den Funktionsumfang der Komponenten einfacher und schneller kennen zu lernen.



Aktion
Entschlüsselung
Chiffre-Variante
ADFGVXZ
Substitutions-Kennwort
BAUM
Substitutions-Matrix
BAUMCDEFGHIJKLNOPQRSTVWXYZ0123456789 .,:;?()-+><=
Transpositions-Kennwort
HAUS
Transposition Spaltenreihenfolge
2-1-4-3

Abb. 4.5: Zu sehen sind die Einstellungen der ADFGVX-Chiffrier-Komponente mit der Variante ADFGVXZ (7x7)

### 4.4.1 Vorlagen

Eine Vorlage besteht aus drei Dateien:

1. `CWM-Projekt-Datei`
2. `Bild (png)`
3. `XML-Datei`

Die CWM-Datei besteht aus einer Anordnung mehrerer Komponenten. Diese Datei kann von CT2 in die Arbeitsfläche geladen werden. Textfelder in einer solchen Vorlage können nicht nur direkt mit Texten gefüllt werden, sondern auch als Variablen definiert sein.

Diese Variablen können dann über die XML-Datei angesteuert werden. So kann das Szenario einer einzigen Vorlage gleich für mehrere Sprachen erzeugt werden. Außerdem kann eine Zusammenfassung geschrieben werden, die angezeigt wird, sobald man im Startcenter die Maus über den Namen der Vorlage bewegt. Zusätzlich können Schlüsselwörter definiert werden, über die man die Vorlage suchen kann.

Das Icon wird ebenfalls in der XML-Datei angegeben, so wird es später in CT2, neben dem Namen der Vorlage, angezeigt.

### 4.4.2 Mehrsprachige Hilfen

Über das Icon mit dem Fragezeichen lässt sich die Onlinehilfe aufrufen. Zu jeder Komponente muss eine Onlinehilfe in den Sprachen Deutsch und Englisch existieren.

Eine solche Onlinehilfe ist in verschiedene Abschnitte gegliedert. Der Benutzer erfährt etwas zum Hintergrund der Komponente (bspw. dass die ADFGVX-Chiffre im ersten Weltkrieg eingesetzt wurde und eine Kombination aus Substitution und Transposition ist), aber auch wie die Komponente zu benutzen ist. Die Konnektoren und Einstellungsmöglichkeiten werden explizit aufgelistet, mit einer kurzen Beschreibung und welcher Datentyp verlangt wird.

Zum Schluss folgt eine Auflistung der Vorlagen, in der die Komponente eingebunden ist, sowie eine Angabe zu Referenzen, die weitere Informationen liefern.

# 5 Evaluation der ADFGVX-Analyse-Komponente

In diesem Kapitel werden die Ergebnisse der Evaluation der neu implementierten ADFGVX-Analyse-Komponente präsentiert. Dazu wurden 5.377 Durchläufe mit den Schlüssellängen $5 - 21$ durchgeführt und ausgewertet. Diese ungerade Zahl kommt daher, dass aus zeitlichen Gründen nur 127, anstatt den geplanten 250, Analysen für die Schlüssellänge 21 durchgeführt werden konnten. Alle grafischen Darstellungen zur Länge 21 beziehen sich auf die 127 Analysen, es fand keine Skalierung auf 250 statt.

Lasry schreibt, dass er mehrere Nachrichten braucht, die mit demselben Schlüssel verschlüsselt wurden.[Las16] Bei seinem Verfahren werden diese Einzel-Nachrichten (Geheimtexte) transponiert, mit dem Standard-Alphabet monoalphabetisch substituiert und anschließend in einem String-Objekt zusammen gefasst (siehe Kapitel 4.1.1). Erst dann wird der Koinzidenzindex für dieses String-Objekt berechnet, die Anzahl der Nachrichten spielt demnach keine Rolle, die Länge ist entscheidend.

Da aber in der Realität eher viele kurze Nachrichten mit der ADFGVX-Chiffre verschlüsselt wurden, wurde die Evaluation auch mit vielen kurzen Nachrichten durchgeführt.

## 5.1 Datenbasis der Evaluation

Zur Erzeugung der Datenbasis der Evaluation wurde das Kinderbuch Pinocchio ausgewählt.[Col] Aus diesem wurden die ersten zehn Kapitel (33.000 Zeichen) kopiert, um Sonderzeichen bereinigt, Zahlen ausgeschrieben und Umlaute ersetzt.

Die Zahlen wurden ausgeschrieben, weil diese ein Problem in der Substitutionsanalyse darstellen. Lasry [Las16] schreibt dazu selbst in seinem Kapitel 5.4:

*Eight of the 14 transposition keys we recovered are identical to Childs's keys, with some minor differences in the substitution keys. Those differences are mainly due to the fact our algorithm is unable to determine the exact position of the digits 0 to 9 in the substitution Polybius square.*

Aus diesem Klartext wurden 330 Dateien jeweils der Länge 100 erzeugt und gespeichert. Anschließend wurden für die Schlüssellängen 5 bis 21 jeweils 5 Schlüssel generiert ($17 \cdot 5 = 85$). Für jeden dieser Schlüssel wurden zufällig 50 dieser 330

Dateien mit der ADFGVX-Chiffre verschlüsselt ($50 \cdot 85 = 4.250$). Außerdem wird für jeden dieser 85 Schlüssel eine Datei (combined.txt) erzeugt, in der pro Zeile jeweils einer der 50 erzeugten Geheimtexte steht.

Aus einer Datei combined.txt werden 50 Nachrichten erzeugt, jeweils fünf mit 1 bis 10 Geheimtexten. Somit enthalten jeweils fünf dieser Dateien 200 bis 2.000 Zeichen in der Geheimschrift ADFGVX (100 bis 1.000 Zeichen im Klartext). Pro Schlüssellänge wurden also 250 Dateien mit Geheimtexten erzeugt.

Daraus ergeben sich insgesamt 4.250 verschlüsselte Dateien in den Längen 200 bis 2.000, die mit 85 verschiedenen Schlüsseln verschlüsselt wurden. Diese wurden zur Berechnung der Evaluation der Analyse-Komponente benutzt.

Um die Menge an Nachrichten zu verschlüsseln, wurden zwei C#-Programme geschrieben: Das Programm **adfgvxKonsole** beinhaltet Teile des Quellcodes der ADFGVX-Chiffrier-Komponente aus CT2, um mit dieser Konsolenanwendung die Texte automatisch zu verschlüsseln. Das zweite Programm **FileGenerator** erzeugte anschließend die Nachrichten, die für die Evaluation relevant sind.

Durchgeführt wurde die Evaluation auf einem vServer[1]. Dieser ist mit acht virtuellen CPUs und dem Betriebssystem Windows Server 2016 konfiguriert (Abb. 5.1). Erreichbar ist er über die Remotedesktop-Verbindung, die jedes aktuelle Windows-Betriebssystem mit bringt.

An Rechenzeit wurden insgesamt 81,6 Stunden verbraucht. Eine Single-CPU würde für die selben Berechnungen 27,2 Tage benötigen.

## 5.2 Durchführung der Evaluation

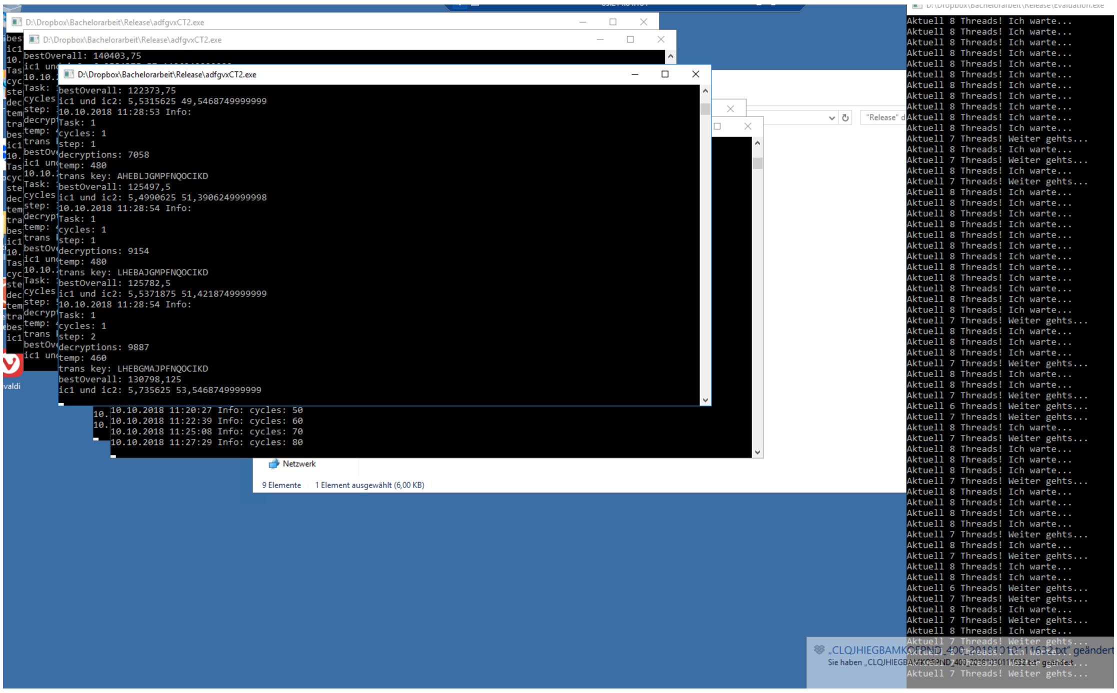
Um die Evaluation durchzuführen wurde die bereits in Kapitel 4.1.2 erwähnte Konsolenanwendung (adfgvxCT2) benutzt. Ergänzt wurde die Konsolenanwendung durch das Programm **EvaluationsScheduler**, welches alle Dateien eines Ordners ausliest, für jede dieser Dateien einen Prozess der Konsolenanwendung startet und maximal acht dieser Prozesse gleichzeitig abarbeitet. Nur acht Prozesse gleichzeitig, weil der vServer nur 8 virtuelle CPUs besitzt.

Die Konsolenanwendung (adfgvxCT2) kann über eine Konfigurationsdatei konfiguriert werden. Für die Evaluation war die Eingabe der abzuarbeitenden Schlüssellänge relevant. So konnten für jede Schlüssellänge alle erstellten Dateien automatisiert abgearbeitet werden, ohne dass ein manuelles Eingreifen notwendig wurde. Die Ausgaben wurden durch einen Logger zusätzlich in Dateien umgeleitet, die eindeutig benannt wurden:
$[KEY]\_[TEXTLAENGE]\_[ZEITSTEMPEL]$

Für die Evaluation wurde die Konsolenanwendung so programmiert, dass sie abbricht, sobald der richtige Schlüssel gefunden oder 100 Zyklen durchlaufen wurden.

---

[1] Virtueller Server eines Hosting-Anbieters



```
bestOverall: 140403,75
```

```
bestOverall: 122373,75
ic1 und ic2: 5,5315625 49,5468749999999
10.10.2018 11:28:53 Info:
Task: 1
cycles: 1
step: 1
decryptions: 7058
temp: 480
trans key: AHEBLJGMPFNQOCIKD
bestOverall: 125497,5
ic1 und ic2: 5,4990625 51,3906249999998
10.10.2018 11:28:54 Info:
Task: 1
cycles: 1
step: 1
decryptions: 9154
temp: 480
trans key: LHEBAJGMPFNQOCIKD
bestOverall: 125782,5
ic1 und ic2: 5,5371875 51,4218749999999
10.10.2018 11:28:54 Info:
Task: 1
cycles: 1
step: 2
decryptions: 9887
temp: 460
trans key: LHEBGMAJPFNQOCIKD
bestOverall: 130798,125
ic1 und ic2: 5,735625 53,5468749999999
```

```
10.10.2018 11:20:27 Info: cycles: 50
10.10.2018 11:22:39 Info: cycles: 60
10.10.2018 11:25:08 Info: cycles: 70
10.10.2018 11:27:29 Info: cycles: 80
```

```
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 6 Threads! Weiter gehts...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 8 Threads! Ich warte...
Aktuell 6 Threads! Weiter gehts...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 8 Threads! Ich warte...
Aktuell 7 Threads! Weiter gehts...
Aktuell 7 Threads! Weiter gehts...
```

„CLQJHIEGBAMKOFPND_400_20181010111632.txt" geändert
Sie haben „CLQJHIEGBAMKOFPND_400_20181010111632.txt" geändert

Abb. 5.1: Analysen der Schlüssellänge 17 auf dem vServer

In Abb. 5.1 ist zu sehen, wie Analysen für die Schlüssellänge 17 durchgeführt werden.

## 5.3 Evaluation der Optimierungen am Quellcode

Durch das Refactoring ergaben sich auch zwangsläufig Optimierungen am Code. Zum Beispiel beschleunigt der Wegfall des Koinzidenzindex für Trigramme die Durchführung der Methode **stats()** erheblich. Um das mit Zahlen bestätigen zu können, wurden die Schlüssellängen 5 bis 9 einmal ohne und einmal mit den Optimierungen durchgeführt.

In Tabelle 5.1 ist zu sehen, dass die optimierte Berechnungszeit bei geraden Schlüssellängen höher liegt als bei ungeraden (Pro Schlüssellänge werden 250 Analysen durchgeführt und die Konsolenanwendung bricht ab, sobald der Schlüssel gefunden oder 100 Zyklen durchlaufen wurden). Die Optimierung brachte eine Beschleunigung bis zu Faktor 30. Da der eigens programmierte Scheduler nicht auf Performance programmiert wurde, reduzierte sich der Faktor, bei 250 Analysen, auf bis zu 18.

Tabelle 5.2 zeigt, wie erfolgreich die Analyse der jeweils 250 Dateien pro Schlüssellänge waren. Aus diesen beiden Tabellen ist abzuleiten, dass sich die Qualität der Ergebnisse durch die Optimierung weder wirklich verbessert noch verschlechtert,

| Schlüssellänge | Optimiert | Nicht optimiert |
|---|---|---|
| 5 | 29 | 30 |
| 6 | 39 | 305 |
| 7 | 37 | 58 |
| 8 | 40 | 737 |
| 9 | 37 | 127 |

Tab. 5.1: Berechnungszeit in Minuten mit und ohne Optimierung für die Schlüssellängen 5 bis 9

| Schlüssellänge | Optimiert | Nicht optimiert |
|---|---|---|
| 5 | 100 | 100 |
| 6 | 98,4 | 98,4 |
| 7 | 99,2 | 98,8 |
| 8 | 98 | 97,6 |
| 9 | 98,8 | 98,8 |

Tab. 5.2: Erfolgreiche Ergebnisse in Prozent mit und ohne Optimierung für die Schlüssellängen 5 bis 9

sich aber die Geschwindigkeit deutlich erhöht hat.

## 5.4 Evaluation der optimierten Konsolenanwendung

Die optimierte Konsolenanwendung ist vom Quellcode her identisch mit der Implementierung in CT2, nur dass in CT2 eine grafische Oberfläche die Daten aufbereitet darstellt.

In Abb. 5.2 wird die Berechnungszeit der Schlüssellängen 5 bis 21 grafisch dargestellt. Dabei ist zu erkennen, dass die Berechnung von geraden Schlüssellängen immer länger gedauert hat als bei den beiden benachbarten ungeraden Schlüssellängen. In Kapitel 2.6.6.3 wurde bereits beschrieben, dass Lasrys Verfahren bei ungeraden erfolgreicher ist als bei geraden Schlüssellängen.

Abb. 5.3 zeigt, dass die Ergebnisse bei geraden Schlüssellängen nicht immer schlechter sind als bei benachbarten ungeraden Schlüssellängen. Da die Schlüssel zu den

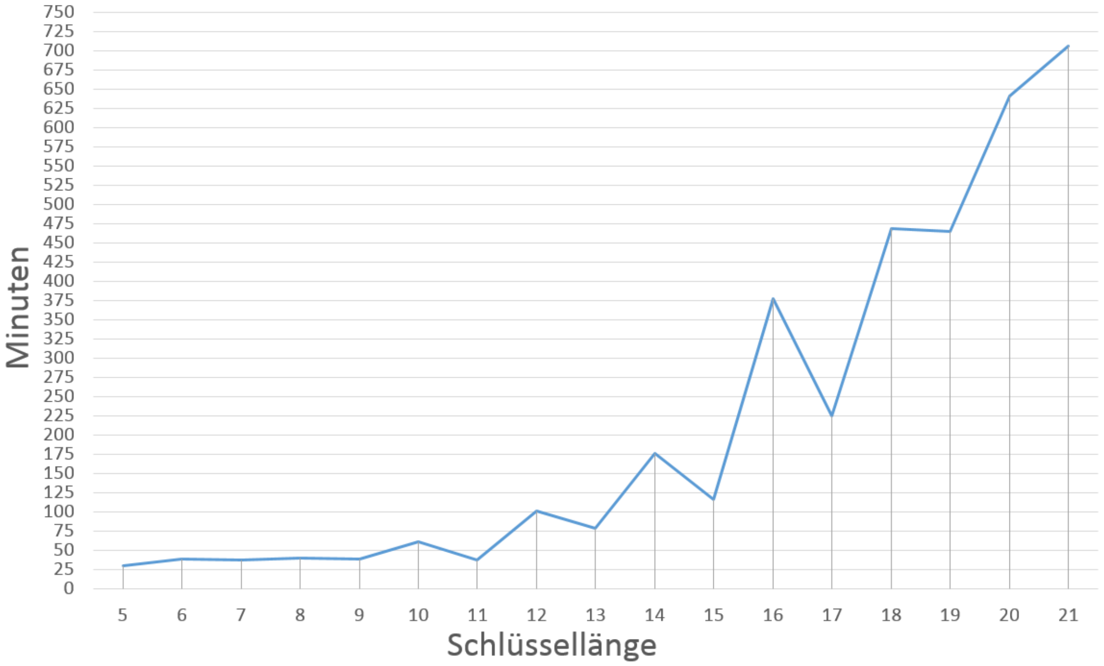


Abb. 5.2: Die Abbildung zeigt, wie lange es dauert, 250 Analysen pro Schlüssellänge durchzuführen. Bei Schlüssellänge 21 nur 127 Analysen

Nachrichten bekannt sind, kann die Konsolenanwendung die gefundenen Schlüssel mit dem richtigen vergleichen. Dass die Berechnungszeit höher ist, liegt daran, dass die Konsolenanwendung nur vorzeitig abbricht, wenn der 100% richtige Transpositionsschlüssel gefunden wurde. Bei geraden Schlüssellängen gibt es aber mindestens eine Permutation des Schlüssels, die exakt das gleiche Ergebnis liefern wird. Bei diesem Schlüssel bricht die Konsolenanwendung allerdings nicht ab.

Eine Erklärung, warum das so ist, lässt sich aus Friedmans Methode in Kapitel 2.6.5.2 herleiten, die sich eine Eigenschaft zunutze machte, die bei geraden Schlüssellängen auftritt. Alle Zeichen in einer Spalte sind entweder die Zeichen der Spalte oder der Zeile des Polybius-Quadrats.

Angenommen, es werden die Zeichen `ANGEWANDTE INFORMATIONSSICHERHEIT` mit einem Standard-Polybius-Quadrat substituiert und anschließend mit einem Transpositionsschlüssel der Länge acht (2-8-5-6-7-1-4-3) verschlüsselt. Daraus ergibt sich folgender ADFGVX-Geheimtext:
`AFVFDAADADDAADAGFAVVDVDXFAADGAGDDAGDFDGGAFADXGXAGADGGADDGAGGVXAVFVFF`

Dieser Geheimtext kann auch mit verdrehten Spaltenpaaren (8-2-6-5-1-7-3-4) und einem gespiegeltem Substitutionsalphabet wieder in den richtigen Klartext entschlüsselt werden (Abb. 5.4).

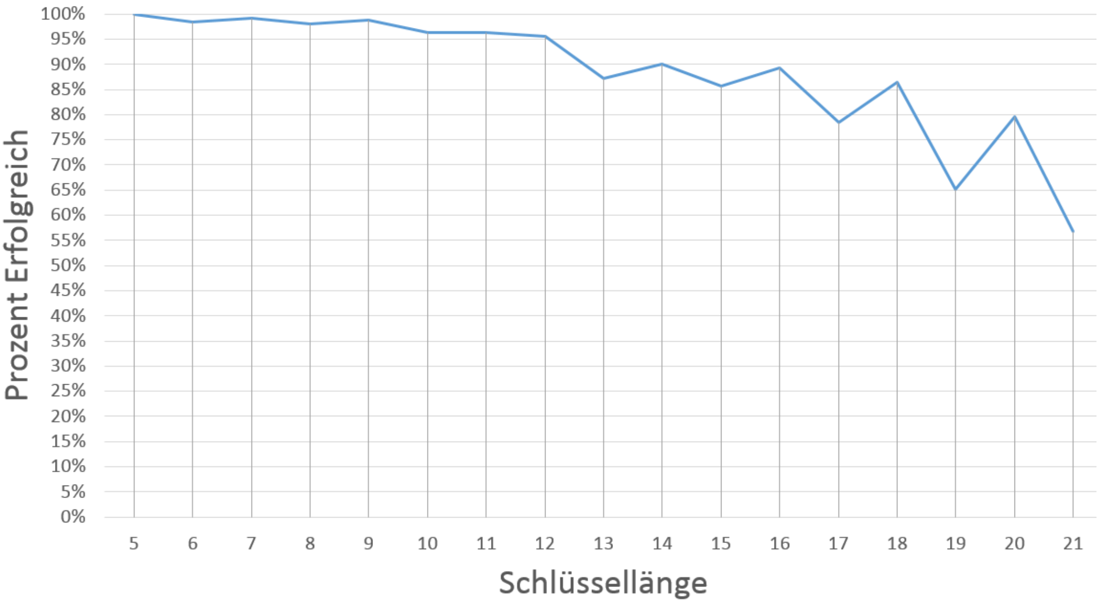
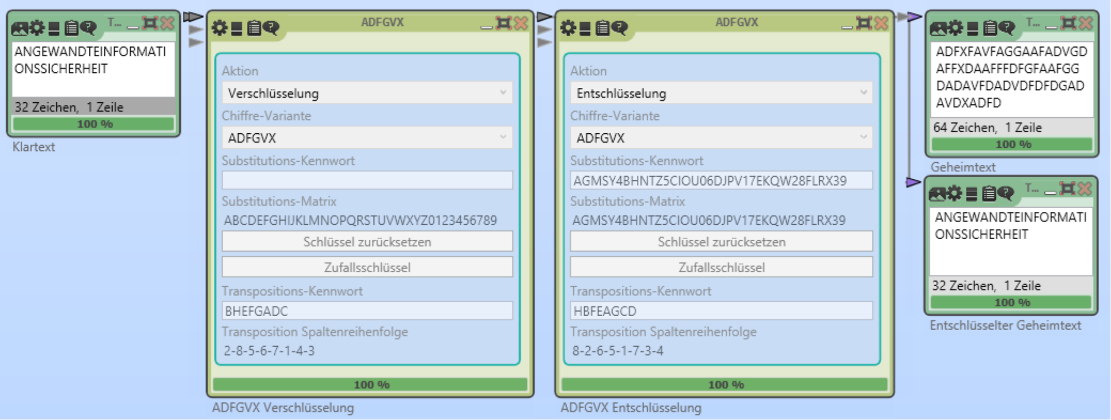


Abb. 5.3: Die Abbildung zeigt, wie erfolgreich die 250 Analysen pro Schlüssellänge waren. Bei Schlüssellänge 21 nur 127 Analysen



Klartext

ANGEWANDTEINFORMATI
ONSSICHERHEIT

32 Zeichen, 1 Zeile

ADFGVX Verschlüsselung

Aktion: Verschlüsselung
Chiffre-Variante: ADFGVX
Substitutions-Kennwort:
Substitutions-Matrix: ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
Transpositions-Kennwort: BHEFGADC
Transposition Spaltenreihenfolge: 2-8-5-6-7-1-4-3

ADFGVX Entschlüsselung

Aktion: Entschlüsselung
Chiffre-Variante: ADFGVX
Substitutions-Kennwort: AGMSY4BHNTZ5CIOU06DJPV17EKQW28FLRX39
Substitutions-Matrix: AGMSY4BHNTZ5CIOU06DJPV17EKQW28FLRX39
Transpositions-Kennwort: HBFEAGCD
Transposition Spaltenreihenfolge: 8-2-6-5-1-7-3-4

Geheimtext

ADFXFAVFAGGAAFADVGD
AFFXDAAFFFDFGFAAFGG
DADAVFDADVDFDFDGAD
AVDXADFD

64 Zeichen, 1 Zeile

Entschlüsselter Geheimtext

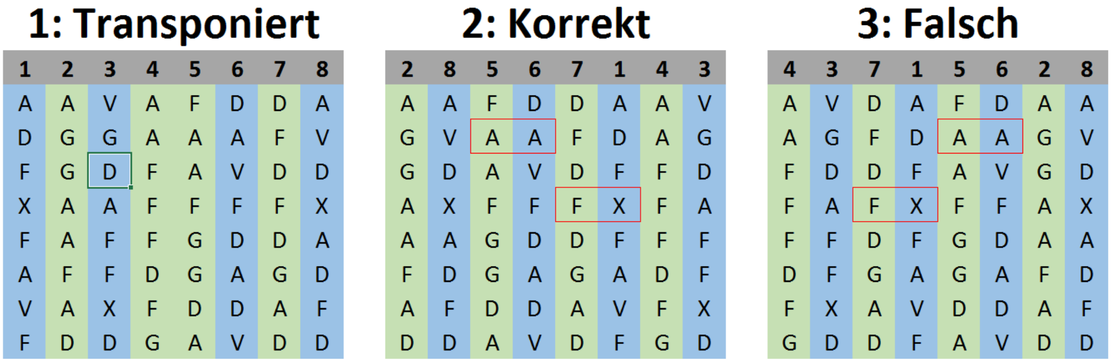
ANGEWANDTEINFORMATI
ONSSICHERHEIT

32 Zeichen, 1 Zeile

Abb. 5.4: Die Abbildung zeigt, dass es bei geraden Schlüssellängen eine andere Permutation mit dem gespiegeltem Substitutionsalphabet gibt, die denselben Klartext ausgibt.

Es ist also die "richtige" Permutation des Schlüssels. Das wird von der Konsolenanwendung nur nicht erkannt, so dass sie nicht vorzeitig abbricht. Dieselbe Eigenschaft führt aber noch zu einem weiteren Problem, welches die Kostenfunktion im Gegensatz zum ersten Problem nicht abfangen kann.

In Abb. 5.5 sieht man den ADFGVX-Text im linken (1: Transponiert) Rechteck noch unverändert, von oben nach unten und links nach rechts, geschrieben. Spalten 1, 3, 6 und 8 (blau) sind ADFGVX-Zeichen aus der Spalte und die Spalten 2, 4, 5 und 7 (grün) aus der Zeile des Polybius-Quadrats. Das mittlere Rechteck (2: Korrekt) ist die korrekte Anordnung der Spalten, um mittels Substitution auf den Ausgangstext zu kommen. Das rechte Rechteck (3: Falsch) würde folgenden Klartext ergeben: `EGNADNAWNIETMROFOITAISSNREHCTIEH`



**1: Transponiert**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| A | A | V | A | F | D | D | A |
| D | G | G | A | A | A | F | V |
| F | G | D | F | A | V | D | D |
| X | A | A | F | F | F | F | X |
| F | A | F | F | G | D | D | A |
| A | F | F | D | G | A | G | D |
| V | A | X | F | D | D | A | F |
| F | D | D | G | A | V | D | D |

**2: Korrekt**

| 2 | 8 | 5 | 6 | 7 | 1 | 4 | 3 |
|---|---|---|---|---|---|---|---|
| A | A | F | D | D | A | A | V |
| G | V | A | A | F | D | A | G |
| G | D | A | V | D | F | F | D |
| A | X | F | F | F | X | F | A |
| A | A | G | D | D | F | F | F |
| F | D | G | A | G | A | D | F |
| A | F | D | D | A | V | F | X |
| D | D | A | V | D | F | G | D |

**3: Falsch**

| 4 | 3 | 7 | 1 | 5 | 6 | 2 | 8 |
|---|---|---|---|---|---|---|---|
| A | V | D | A | F | D | A | A |
| A | G | F | D | A | A | G | V |
| F | D | D | F | A | V | G | D |
| F | A | F | X | F | F | A | X |
| F | F | D | F | G | D | A | A |
| D | F | G | A | G | A | F | D |
| F | X | A | V | D | D | A | F |
| G | D | D | F | A | V | D | D |

Abb. 5.5: Transpositionsproblem gerader Schlüssellängen

Trotzdem hat diese Anordnung denselben Koinzidenzindex über Monogramme wie der richtige Klartext. Über Bigramme kann man in diesem Fall noch die korrekte Reihenfolge feststellen, aber auch da gibt es Konstellationen, die ähnlich gute Kostenwerte aufweisen wie der eigentliche Klartext. Dieses Problem mittels Koinzidenzindex zu lösen ist nicht trivial, erklärt aber zusätzlich, warum gerade Schlüssellängen problematischer sind als ungerade.

Bisher wurden in der Evaluation die Faktoren Zeit und Schlüssellänge betrachtet, aber noch nicht die Länge der zu entschlüsselnden Nachrichten.

In Abb. 5.6 sieht man für die Schlüssellängen 5 bis 21, wie hoch die Erfolgsquote abhängig von der Nachrichtenlänge ist. Der neue Analyzer ist für niedrige Schlüssellängen bereits bei einer Nachrichtenlänge von 100 Zeichen sehr erfolgreich. Ab Schlüssellänge elf werden kontinuierlich längere Nachrichten benötigt. Die Schlüssellänge 17 liefert erst ab ca. 400 Zeichen eine Erfolgsrate von über 90%. Die Länge der analysierten Nachrichten trägt somit erheblich zur Erfolgsrate bei.

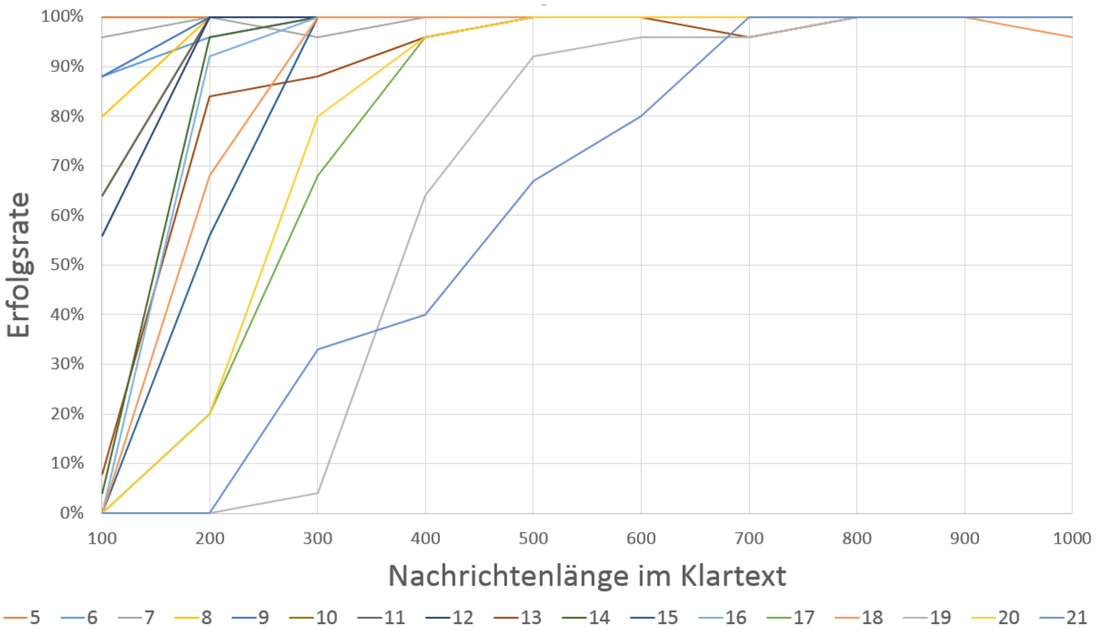


Abb. 5.6: Die Abbildung zeigt, wie bei den Graphen verschiedener Schlüssellängen Erfolgsquote und Nachrichtenlänge zusammenhängen.

In Abb. 5.7 wird gezeigt, wie viele Entschlüsselungen im Median pro Schlüssellänge durchgeführt wurden, um den richtigen Schlüssel zu finden. Dabei wurden alle 250 Analysen pro Schlüssellänge berücksichtigt. Wurde die Berechnung nach 100 Zyklen abgebrochen, sind die Werte aus der Spalte Durchläufe der Tabelle 4.1 in die Auswertung mit eingeflossen.

Der Median wurde gewählt, weil es immer wieder große Ausreißer gibt, welche dazu führen, dass das arithmetische Mittel nicht aussagekräftig ist. Außerdem konnten so die Analysen mit berücksichtigt werden, die nicht den richtigen Transpositionsschlüssel gefunden haben.

In Abb. 5.7 ist zu erkennen, dass die logarithmierte Anzahl an Entschlüsselungen im Median mit der Schlüssellänge ansteigt. Bis zur Schlüssellänge elf steigt der

Graph sukzessiv an. Ab der Schlüssellänge zwölf steigt die Anzahl an Entschlüsselungen im Median für gerade und ungerade Schlüssellängen stark unterschiedlich an. Der Grund dafür ist, dass es bei geraden Schlüssellängen zwei richtige Transpositions-Schlüssel gibt. Dass der Median insgesamt so stark ansteigt, liegt an dem exponentiell wachsenden Schlüsselraum.

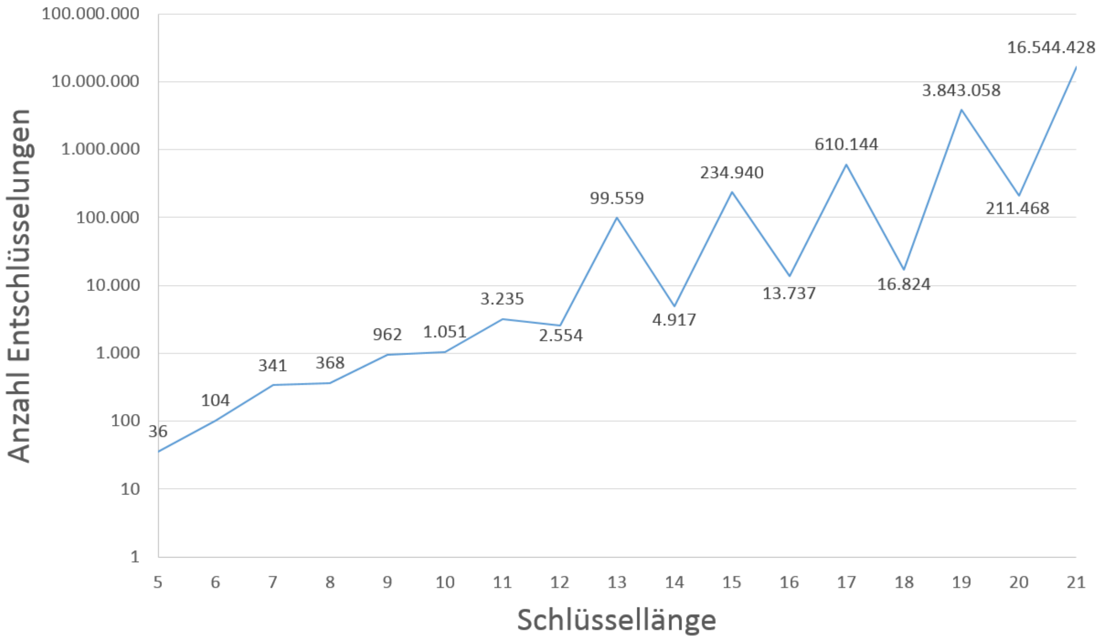


Abb. 5.7: Die Abbildung zeigt, wie viele Entschlüsselungen im Median pro Schlüssellänge durchgeführt wurden, um den richtigen Schlüssel zu finden (logarithmiert).

## 5.5 Challenges bereitgestellt von den Betreuern

Um die ADFGVX-Analyse-Komponente zusammen mit der Substitutionsanalyse-Komponente in CT2 zu testen, erstellten meine Betreuer jeweils zehn unterschiedlich lange Nachrichten für die Schlüssellängen 13, 15, 19, 20, 23 und 25. Diese wurden alle erfolgreich entschlüsselt.

Nachfolgend werden die Nachrichten, die mit einer Schlüssellänge von 25 transponiert wurden, und das Ergebnis präsentiert.

**Geheimtexte:**

1. `XADAV AAVVG AFFAV AFDFA AFXAF VAAAV DAAXG AVFVD VDAAA FFAAG FFFAG FFDAA AFGAA DVAAF GVDAD XDFFF DGAVA XDVGA DGAVA DAXDG GFAAA AXAVA VAAAF AAAAG AXFAA DDDDD GAFFD AFVDV FAAXG X`

2. `VFAAA DGFXD VAFFD VFXGD FAGAV DXDFF XGVAF GAAAF FXDAA FAXAD VAVAD XAGVG XAGAV DGADA ADXDX FXAFD GGXFV AAVFX XAAFF AVFAA GAXGF DFVFD ADGXD XGXFF FFVAA AAFDX DX`

3. `GDVFG FFGDF GAGVG DFAAF AFAVA GVFXA DAAAX AVVDA VAXDG AFAVA DVDFD FAFFA XADAD AXAAG DFGAF GAAVG GAXAG VAADD DVADA ADFAF XAFAV FAFFF GXAVF AFFAA VAGF`

4. `XAADV AGAFX AAFXF GVAVF XFXAF FXAVF VGFVF FFGFV AVGXA GAVGF AAVAD FFAAX GGAVA AAFXF VAAXA GDXAV FXGVG DDGAV ADGXA GFFVA VFVFG XAGAA AXAVA VFFDA VAGDF FGAAD AFFGX AXDXA GFXFD FADAX DAFAV DXGFA XGVDA AXAAF AGFGF XVAFA VAXFG FAXAX GVDAA VFVFA`

5. `ADFDV AXAGA FDVGG AAAFD VAXDV FGFDF AGDAA XAAAV AFFDA AVFXF VAAGF FFAVF XFGAV FDAFG AAVDA GVGGA GAGVF FXAVA AFAAF XAADG FAGFV FVAGA XAFAF FADXF FAXAA AGXGV AVGFA FAVAX DXFDX AVAFD VFVAG DAGFA DVFXA DAAAX GAFAA GAXDX VAGGG FGAAA ADAFV GVAAA DAAVA FFXFF AGFXD FVAGF XDGA`

6. `VFFAX AXAXF XDFFG AGADF GDXAV FAAAG FAGVG XAGAX DVAFF FAAAA VGGGF AXFVD VDDFV AAAAA FVFAF VAGAX DVFVA VGAAF GADGV FFDAX FVFVG VXAXF VAGAD AAXFX AVAAF FVFAA VDFDF FAFAF AFGFF AAVAF FXGXF XAVFA DVFGD AFXGV ADAFA VXAFA AAXFX FVFGF GFGXA DFVFV AFFAA AAVDD AVDDX FAADG`

7. `XFXDV AAFGD GVAAF AFFDG AVAAG AVFFA GAFGX VFDAA FAVFX AVDGG XDAVA VAFGD GAXDV AXFAF GDGDA AGADA DDXDV VFXAA VDXDD AXAGF AFFGX AVAXA XAXDX AVFAF AVAAV AFADG AFDGD VADFX`

8. `DAVFG AFAAD VADAA GAXDX AXAFG DVGXD VAAAF XAAFV AAADV AGAAF FADAA VDAAA XDXFV AXDXA FFDAF AAFXD VGAAG XAAAF VDGAA AFDGA XAAAF ADAXF AAGAV ADAGA AFAFV DXDDX AVAAA GAFFA`

9. `XFVAA DGFGA VGGDV FXAVF VVFDD ADGFA XFDAG AADFD XAVFG DAXFA AAFFA ADDFV AGDGA XGAFD GAVDD AAAAF AAXAV AGAGG ADAAX DGAFF VAFDX FVFGA FDXAV AVXAG AVAFG FAGVA GFAFX AADXD VAVDV XGXAG ADFFG AAAGV FFDGD XAAFA AAVAA GAAAX DV`

10. `XDFAV AXGAA AADXD XDGFF FFGFG FXDVF VGVDA DVDVA FAFFG ADFDF VAFDG VAXAX AGADD FFGAG AFFDF FFFAF FAGVD AADDG DXAGF DFDAG FDAGD XAAAV GADVD AAVAV FFAAX AXAGD XAGAF GDXDA AFAGG DGGGA AAVGF DAVGG AXGVF VDVDX AFAAA GFXDD DVAVA AVFXA GAVAV FDVFX AVDGA XAFDG DXAVG AAGFF VGGAF GDFFD AXAGG VGGAF AGAGF XDXAA AVFXA X`

Die Analyse ergab:
Transpositionsschlüssel: NDEIFWYCBQXTOGKUJRSALPHMV
Permutation: (14-4-5-9-6-23-25-3-2-17-24-20-15-7-11-21-10-18-19-1-12-16-8-13-22)
Substitutionsalphabet: FGHIEJKLMNOPQRBX3AUDCVW7SZ21Y

1. DICKE BERTA FEUERBEREIT Y ERBITTE ZIELKOORDINATEN FUER SCHUSSABGABE IN FEINDGEBIET

2. ERHEBLICHE VERLUSTE AN MENSCH UND MATERIAL Y WANN IST MIT NACHSCHUB ZU RECHNEN

3. OESTLICHE FLANKE WURDE UEBERRANT Y KP DREI ZIEHT ZURUECK ZU STELLUNG OTTO

4. SIEG IST NAHE KAMERADEN Y DER KAISER ERWARTET UNEINGEFOCHTENEN KAMPFESWILLEN Y SO AN ALLE MANNSCHAFTEN WEITERLEITEN

5. DER KAISER IST ABGEDANKT Y DER KRIEG IST ZU ENDE Y FEUER SOFORT EINSTELLEN UND WAFFEN NIEDER LEGEN Y TRANSPORT IN DIE HEIMAT STEHT BEVOR

6. BENZIN UND MUNITION IST EINGETROFFEN UND KANN IN DIE JEWEILIGEN KOMPANIEN WEITERGELEITET WERDEN Y HPTM STOLZE DIREKT ANSPRECHEN

7. DER GEGENANGRIFF ERFOLG UM MITTERNACHT Y ALLE SOLDATEN HABEN SICH BEREIT ZU HALTEN

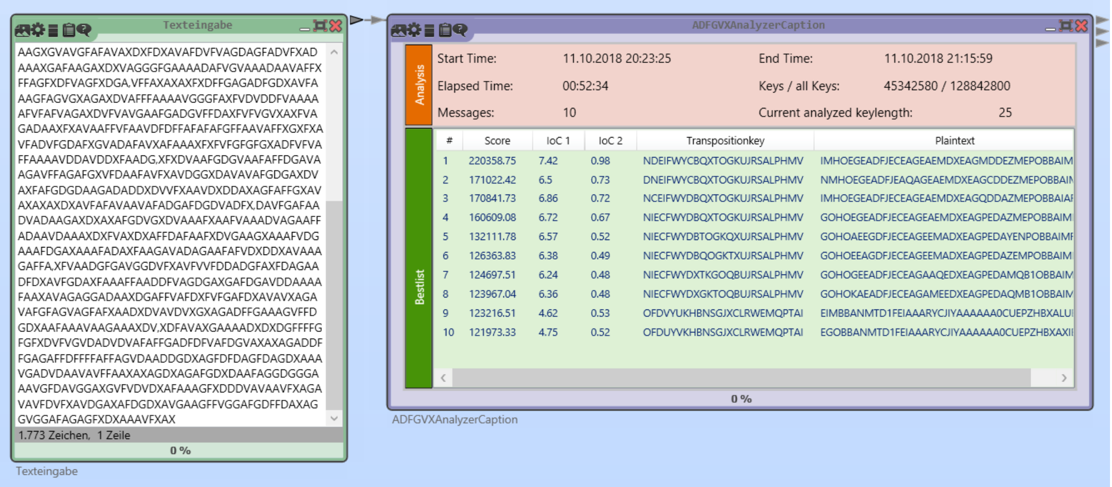
8. ALBRECHT DONAU FELDHERR BERTA BERTA PETER DREI DREI NEUN EINS ALBRECHT DONAU BERTA

9. GEGEN ACHT NULL NULL UHR SIND SOLDATEN ABMARSCHBEREIT Y ERBITTE WEITERE BEFEHLE FUER FORTANG DER OFFENSIVE

10. MELDE SCHLACHTSCHIFF HOMBURG IST GESUNKEN Y FEINDLICHES UBOOT HAT TOETLICHEN TREFFER GELANDET Y MANNSCHAFT GROESSTENTEILS GERETTET Y KAUM VERLUSTE

Bei der Substitutionsanalyse wurden in diesem Fall P <-> Y miteinander vertauscht (ist oben bereits korrigiert). Trotzdem gelten die Nachrichten als gebrochen, da der Inhalt klar verständlich ist.

In den folgenden Abb. 5.8 und 5.9 sieht man, dass die Analyse, um die Transposition zu brechen, in 52 Minuten 45.342.580 Schlüssel berechnet hat. Die anschließende Substitutionsanalyse hat 9 Sekunden gedauert.



AAGXGVAVGFAFAVAXDXFDXAVAFDVFVAGDAGFADVFXAD
AAAXGAFAAGAXDXVAGGGFGAAAADAFVGVAAADAAVAFFX
FFAGFXDFVAGFXDGA,VFFAXAXAXFXDFFGAGADFGDXAVFA
AAGFAGVGXAGAXDVAFFFAAAAVGGGFAXFVDVDDFVAAAA
AFVFAFVAGAXDVFVAVGAAFGADGVFFDAXFVFVGVXAXFVA
GADAAXFXAVAAFFVFAAVDFDFFAFAFAFGFFAAVAFFXGXFXA
VFADVFGDAFXGVADAFAVXAFAAAXFXFVFGFGFGXADFVFVA
FFAAAAVDDAVDDXFAADG,XFXDVAAFGDGVAAFAFFDGAVA
AGAVFFAGAFGXVFDAAFAVFXAVDGGXDAVAVAFGDGAXDV
AXFAFGDGDAAGADADDXDVVFXAAVDXDDAXAGFAFFGXAV
AXAXAXDXAVFAFAVAAVAFADGAFDGDVADFX,DAVFGAFAA
DVADAAGAXDXAXAFGDVGXDVAAAFXAAFVAAADVAGAAFF
ADAAVDAAAXDXFVAXDXAFFDAFAAFXDVGAAGXAAAFVDG
AAAFDGAXAAAFADAXFAAGAVADAGAAFAFVDXDDXAVAAA
GAFFA,XFVAADGFGAVGGDVFXAVFVVFDDADGFAXFDAGAA
DFDXAVFGDAXFAAAFFAADDFVAGDGAXGAFDGAVDDAAAA
FAAXAVAGAGGADAAXDGAFFVAFDXFVFGAFDXAVAVXAGA
VAFGFAGVAGFAFXAADXDVAVDVXGXAGADFFGAAAGVFFD
GDXAAFAAAVAAGAAAXDV,XDFAVAXGAAAADXDXDGFFFFG
FGFXDVFVGVDADVDVAFAFFGADFDFVAFDGVAXAXAGADDF
FGAGAFFDFFFFAFFAGVDAADDGDXAGFDFDAGFDAGDXAAA
VGADVDAAVAVFFAAXAXAGDXAGAFGDXDAAFAGGDGGGA
AAVGFDAVGGAXGVFVDVDXAFAAAGFXDDDVAVAAVFXAGA
VAVFDVFXAVDGAXAFDGDXAVGAAGFFVGGAFGDFFDAXAG
GVGGAFAGAGFXDXAAAVFXAX

1.773 Zeichen, 1 Zeile

0 %

Start Time: 11.10.2018 20:23:25
End Time: 11.10.2018 21:15:59
Elapsed Time: 00:52:34
Keys / all Keys: 45342580 / 128842800
Messages: 10
Current analyzed keylength: 25

| # | Score | IoC 1 | IoC 2 | Transpositionkey | Plaintext |
|---|---|---|---|---|---|
| 1 | 220358.75 | 7.42 | 0.98 | NDEIFWYCBQXTOGKUJRSALPHMV | IMHOEGEADFJECEAGEAEMDXEAGMDDEZMEPOBBAIM |
| 2 | 171022.42 | 6.5 | 0.73 | DNEIFWYCBQXTOGKUJRSALPHMV | NMHOEGEADFJEAQAGEAEMDXEAGCDDEZMEPOBBAI |
| 3 | 170841.73 | 6.86 | 0.72 | NCEIFWYDBQXTOGKUJRSALPHMV | IMHOEGEADFJECEAGEAEMDXEAGQDDAZMEPOBBAIA |
| 4 | 160609.08 | 6.72 | 0.67 | NIECFWYDBQXTOGKUJRSALPHMV | GOHOEGEADFJECEAGEAEMDXEAGPEDAZMEPOBBAIM |
| 5 | 132111.78 | 6.57 | 0.52 | NIECFWYDBTOGKQXUJRSALPHMV | GOHOAEEGDFJECEAGEEMADXEAGPEDAYENPOBBAIM |
| 6 | 126363.83 | 6.38 | 0.49 | NIECFWYDBQOGKTXUJRSALPHMV | GOHOEEAGDFJECEAGEEMADXEAGPEDAZEMPOBBAIM |
| 7 | 124697.51 | 6.24 | 0.48 | NIECFWYDXTKGOQBUJRSALPHMV | GOHOGEEADFJECEAGAAQEDXEAGPEDAMQB1OBBAIM |
| 8 | 123967.04 | 6.36 | 0.48 | NIECFWYDXGKTOQBUJRSALPHMV | GOHOKAEADFJECEAGAMEEDXEAGPEDAQMB1OBBAIM |
| 9 | 123216.51 | 4.62 | 0.53 | OFDVYUKHBNSGJXCLRWEMQPTAI | EIMBBANMTD1FEIAAARYCJIYAAAAAA0CUEPZHBXALU |
| 10 | 121973.33 | 4.75 | 0.52 | OFDUYVKHBNSGJXCLRWEMQPTAI | EGOBBANMTD1FEIAAARYCJIYAAAAAA0CUEPZHBXAXI |

0 %

Abb. 5.8: Challenge – Transpositionsanalyse der Schlüssellänge 25

Start: 11.10.2018 20:38:16
Ende: 11.10.2018 20:38:25
Vergangen: 00:00:09
Getestete Schlüssel: 362.790
Schlüssel / sec: 37.727

| # | Wert | Angriff | Schlüssel | |
|---|---|---|---|---|
| 1 | -4,47230 | H | fghiejklmnopqrbx3audcvw7sz2 1y | DICKEBERTAFEUERBEREITPERBITTEZIELKOORDINATENFUERSCHUSSABGAB |
| 2 | -4,47801 | H | fghiejklmnopqrbs3audcxw1vz 7y2 | DICKEBERTAFEUERBEREITVERBITTEZIELKOORDINATENFUERSCHUSSABGAB |

0 %

fghiejklmnopqrbx3audcvw7sz2 1y

30 Zeichen, 1 Zeile

0 %

ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜß

30 Zeichen, 1 Zeile

0 %

DICKEBERTAFEUERBEREITPERBITTEZIELKOORDINATENFUERSCHUSSABGABEINF
EINDGEBIETERHEBLICHEVERLUSTEANMENSCHUNDMATERIALPWANNISTMITN
ACHSCHUBZURECHNENOESTLICHEFLANKEWURDEUEBERRANTPKYDREIZIEHTZ
URUECKZUSTELLUNGOTTOSIEGISTNAHEKAMERADENPDERKAISERERWARTETU
NEINGEFOCHTENENKAMYFESWILLENPSOANALLEMANNSCHAFTENWEITERLEIT
ENDERKAISERISTABGEDANKTPDERKRIEGISTZUENDEPFEUERSOFORTEINSTELLE
NUNDWAFFENNIEDERLEGENPTRANSYORTINDIEHEIMATSTEHTBEVORBENZINU
NDMUNITIONISTEINGETROFFENUNDKANNINDIEJEWEILIGENKOMYANIENWEIT
ERGELEITETWERDENPHYTMSTOLZEDIREKTANSYRECHENDERGEGENANGRIFFER
FOLGUMMITTERNACHTPALLESOLDATENHABENSICHBEREITZUHALTENALBREC
HTDONAUFELDHERRBERTABERTAYETERDREIDREINEUNEINSALBRECHTDONAU
BERTAGEGENACHTNULLNULLUHRSINDSOLDATENABMARSCHBEREITPERBITTE
WEITEREBEFEHLEFUERFORTANGDEROFFENSIVEMELDESCHLACHTSCHIFFHOMB
URGISTGESUNKENPFEINDLICHESUBOOTHATTOETLICHENTREFFERGELANDETP
MANNSCHAFTGROESSTENTEILSGERETTETPKAUMVERLUSTE

IMHOEGEADFJECEAGEAEMDXEAGMDDEZMEPOBBAIMRFDERJCEAUHLCUUFGKFGEMRJEMRIKEGMEDEALEGPMHLEVEAPCUDEFR
QERUHLCRIQFDEAMFPXWFRRMUDQMDRFHLUHLCGZCAEHLRERBEUDPMHLEJPFROEWCAIECEGEAAFRDXOSIAEMZMELDZCACE
HOZCUDEPPCRKBDDBUMEKMUDRFLEOFQEAFIERXIEAOFMUEAEAWFADEDCREMRKEJBHLDEREROFQSJEUWMPPERXUBFRFPPEQ
FRRUHLFJDERWEMDEAPEMDERIEAOFMUEAMUDFGKEIFRODXIEAOAMEKMUDZCERIEXJECEAUBJBADEMRUDEPPERCRIWFJJERRM
EIEAPEKERXDAFRUSBADMRIMELEMQFDUDELDGEVBAGERZMRCRIQCRMDMBRMUDEMRKEDABJJERCRIOFRRMRIMENEWEMPMK
EROBQSFRMERWEMDEAKEPEMDEDWEAIERXLSDQUDBPZEIMAEODFRUSAEHLERIEAKEKERFRKAMJJEAJBPKCQQMDDEARFHLDXF
PPEUBPIFDERLFGERUMHLGEAEMDZCLFPDERFPGAEHLDIBRFCJEPILEAAGEADFGEADFSEDEAIAEMIAEMRECREMRUFPGAEHLDIBRF
CGEADFKEKERFHLDRCPPRCPPCLAUMRIUBPIFDERFGQFAUHLGEAEMDXEAGMDDEWEMDEAEGEJELPEJCEAJBADFRKIEABJJERUMV
EQEPIEUHLPFHLDUHLMJJLBQGCAKMUDKEUCROERXJEMRIPMHLEUCGBBDLFDDBEDPMHLERDAEJJEAKEPFRIEDXQFRRUHLFJDKA

Abb. 5.9: Challenge – Substitutionsanalyse

## 5.6 Fazit

Die Evaluation hat gezeigt, dass das Analyseverfahren von Lasry gut funktioniert. Wie zu erwarten steigt die Berechnungszeit mit länger werdenden Schlüsseln, und die Qualität des Ergebnisses hängt von der Länge der zu analysierenden Nachrichten ab. Trotzdem sind auch Schlüssellängen von 17 bis 21 in wenigen Minuten (bei einer Geheimtextlänge von 500) zu brechen.

Zu der Problematik der geraden Schlüssellängen hatte Lasry [Las16] auf Seite 113 geschrieben:

*„This algorithm performs very well for transposition keys with odd length. In the case of transposition keys with an even length, it will often fail.“*

Die Evaluation hat gezeigt, dass bei niedrigeren Schlüssellängen (5-12) die Erfolgsrate von ungeraden Schlüssellängen minimal besser ist. Bei höheren Schlüssellängen (13-21) war die Erfolgsrate von geraden Schlüssellängen besser. Eine Erklärung könnte dabei sein, dass bei geraden Schlüssellängen immer zwei richtige Schlüssel existieren. Es macht einen großen Unterschied, ob sich in einem sehr großen Schlüsselraum nur ein oder zwei richtige Schlüssel befinden.

Die Tatsache der zwei richtigen Schlüssel, erklärt auch, warum bei höheren geraden Schlüssellängen im Median weniger Entschlüsselungen notwendig waren, bis der richtige Schlüssel gefunden werden konnte.

Diese Erkenntnis ist überraschend, da Lasry ja beschrieben hat, dass das Verfahren bei geraden Schlüssellängen häufiger fehlschlägt.

# 6 Verwandte Arbeiten

Zusätzlich zur Eigen-Evaluation der neuen Analyse-Komponente in Kapitel 5 wurde die Komponente auch mit anderen Implementierungen verglichen. Unter GitHub gibt es zwei Implementierungen dazu mit fast gleichem Namen.

## 6.1 adfgvx-solver

Der **adfgvx-solver** ist in Python programmiert. Der Autor Jon Winsley beschreibt seinen Algorithmus aus 2016 wie folgt: [Win]

1. Wählen Sie eine Länge für den Transpositionsschlüssel
2. Berechnen Sie für jeden möglichen Transpositionsschlüssel der gegebenen Länge den Koinzidenzindex
3. Speichern Sie die 40 Transpositionsschlüssel mit den höchsten Koinzidenzindizes
4. Transponieren Sie den Textes mit diesen 40 Schlüsseln und führen dabei eine Substitution mit einem beliebigen Schlüssel durch
5. Brechen Sie die konvertierten Geheimtexte mit einer Substitutions-Analyse

In Schritt 2 wird deutlich, dass es ein Brute-Force-Angriff ist. In seinem Quellcode ist ein Beispiel mit einer Transpositions-Schlüssellänge von fünf umgesetzt.

Brute-Force-Angriffe auf größere Schlüssellängen werden schnell inpraktikabel: Kapitel 2.2.2 beschrieb, dass schon im Ersten Weltkrieg die üblichen Schlüssellängen größer 17 waren. Ein Brute-Force-Angriff würde dabei 17! = 355.687.428.096.000 Transpositionsschlüssel berechnen müssen.

In Kapitel 2.6 wurde angegeben, das 1.000.000 ($\approx 2^{20}$) Schlüssel pro Sekunde berechnet werden können. Bei $17! \approx 2^{48}$ würde man 268.435.456 ($2^{28}$) Sekunden benötigen ($\approx 8,5 Jahre$).

„`There are other ways to achieve the same goal, and this might not be the best - but it works.`“ [Win]

Dieses Zitat beschreibt den Algorithmus ganz gut. Er ist somit auch nicht zu vergleichen mit der Mächtigkeit des Verfahrens von Lasry.

## 6.2 adfgvxSolver

Ben Ruijl beschreibt seinen in Java geschriebenen Algorithmus aus dem Jahr 2010 in [Ruia, Ruib] wie folgt:

1. Identifizieren der Spalten und Zeilen
2. Suchen der richtigen Reihenfolge der Spalten und Zeilen
3. Transposition rückgängig machen und eine Substitutionsanalyse durchführen

In seiner Einleitung sagt er, dass sein Algorithmus nur für Texte funktioniert, die das „Transpositionsgitter“ zu einem perfekten Rechteck machen und dass die Schlüssellänge gerade sein muss. Diese beiden Eigenschaften erinnern bereits sehr stark an Friedmans Ansatz in Kapitel 2.6.5.2, wenn die Schlüssellängen gerade sind.

Weiter führt er aus, dass die Spalten durch Ausnutzen von Häufigkeitsinformationen aus dem Polybius-Quadrat identifiziert werden können. Dazu wird der Text in ein Gitter geschrieben und aufgrund der Einschränkungen ist jede Spalte in diesem Gitter entweder eine Zeile oder eine Spalte des Polybius-Quadrats. Dies beschreibt tatsächlich den Ansatz von Friedman.

Im weitern Verlauf schildert er, dass sich das Problem damit von $n!$ auf $((n/2)!)^2$ reduzieren lässt. Bei einer Schlüssellänge von 18 bedeutet das, dass immer noch $131.681.894.400 \approx 2^{37}$ Möglichkeiten durchlaufen werden. Zum Vergleich: Laut Tabelle 4.1 in Kapitel 4.1.1 führt die neue Analyse-Komponente bei 100 Zyklen und derselben Schlüssellänge nur 14.140.100 Durchläufe durch (also knapp $\frac{1}{10.000}$ davon).

Schon allein durch die Einschränkungen ist diese Implementierung nicht mit der Flexibilität des Verfahrens von Lasry zu vergleichen.

## 6.3 Weitere ADFGVX-Implementierungen

Rsheldiii auf GitHub beschreibt seinen Algorithmus aus 2012 ebenfalls als einen Brute-Force-Angriff.[Rsh] In diesem benutzt er allerdings ein Wörterbuch, um die Transposition zu brechen. Es werden alle Wörter aus dem Wörterbuch eingelesen und nacheinander durchprobiert.

Chatfield veröffentlichte in 2013 einen Algorithmus, vermeintlich in Python und CUDA.[Cha] Das Repository ist aber unvollständig, es fehlt die Implementierung, um die Transposition zu brechen.

Hexpresso und ein Artikel von wiremask.eu beziehen sich auf ein Beispiel aus der Hackingweek 2015.[hex, wir] Beide gehen davon aus, dass das Ende des Textes im

Klartext bekannt ist. Beide nutzen einen Brute-Force-Angriff, um die Transposition zu lösen.

Alkhalid und Alkhfagi veröffentlichen im Jahr 2016 den Artikel „Cryptanalysis of ADFGX using Genetic Algorithm“.[AA] Sie beschreiben ein Verfahren, das bereits ab einer Schlüssellänge von acht Zeichen, aufgrund der benötigten Rechenzeit, nicht mehr praktikabel ist.

## 6.4 Fazit

Die gefundenen Implementierungen sind alle deutlich langsamer als die Analyse von Lasry. Sie gehören eher zu den Brute-Force-Ansätzen statt zu den heuristischen Verfahren.

Die in der Tabelle 6.1 angegebene maximal Länge bei Lasrys Verfahren ist vermutlich höher. Allerdings wurde in dieser Arbeit kein längerer Transpositionsschlüssel evaluiert.

Trotz Recherche wurden keine weiteren, öffentlich zugänglichen, funktionierenden Implementierungen gefunden, die die ADFGVX-Chiffre wirklich brechen können.

| Autor | Bezeichnung | Transposition | max Länge |
|---|---|---|---|
| Winsley | adfgvx-solver | Brute-Force | - |
| Ruijl | adfgvxSolver | Brute-Force | - |
| Rsheldiii | adfgvx decryptor | Wörterbuch | - |
| Chatfield | adfgvx-solver | Brute-Force | - |
| Hexpresso | ADFGVX | Brute-Force | 11 |
| Alkhalid und Alkhfagi | ADFGX GA | Brute-Force | 8 |
| Lasry | ADFGVX Analyzer | SA | 25 |

Tab. 6.1: Die Tabelle zeigt verwandte Arbeiten zu Lasry und wie dort die Transposition gebrochen wird

# 7 Auswirkung möglicher Erweiterungen der ADFGVX-Chiffre auf die Analysekomplexität

Dieses Kapitel befasst sich kurz theoretisch damit, wie sich verschiedene Größen des Polybius-Quadrats sowie eine doppelte statt einer einfachen Spaltentransposition auf die Schwierigkeit der ADFGVX-Chiffre auswirken.

## 7.1 Unterschiedliche Größen der Substitutionsmatrix in ADFGVX

Der eigentlich triviale Teil der ADFGVX-Chiffre ist die Substitution. Eine der Anforderungen an die Bachelorarbeit war das Implementieren einer 7x7-Matrix. Zu diskutieren ist, ob das einen Sicherheitsgewinn der Chiffre mit sich bringt.

Wird ein Polybius-Quadrat über die Häufigkeitsanalyse rekonstruiert, werden alle im Text vorkommenden Zeichen gezählt und im Verhältnis zur Textlänge gestellt. Vergrößert man das Alphabet, benutzt aber weiterhin nur die Zeichen A bis Z, ist die Textstatistik gleich. Die fehlenden Felder im Polybius-Quadrat werden zufällig aufgefüllt, da sie für den Text irrelevant sind. Eine reine Vergrößerung des Alphabets stellt somit keinen Sicherheitsgewinn dar. Entscheidend ist, wie das Alphabet belegt wird und wie es vom Benutzer zum Verschlüsseln genutzt wird.

Ein Alphabet, wie es in der Einleitung beschrieben wurde[1], hat nur dann einen Effekt, wenn der Benutzer auch alle Zeichen verwendet. Werden im Klartext trotzdem nur die Zeichen A bis Z verwendet, ist der Sicherheitsgewinn minimal. Weder die Substitution noch die Transposition werden schwieriger zu brechen, weil sich sowohl die Häufigkeitsanalyse als auch der Koinzidenzindex nicht ändern, wenn das Alphabet zusätzliche Zeichen enthält, die nicht genutzt werden.

Bei der Verwendung aller Zeichen können sowohl die Werte der Häufigkeitsanalyse (zum Entschlüsseln der Substitution) als auch der Koinzidenzindex (zum Entschlüsseln der Transposition) des Textes von den erwarteten abweichen.

In dem folgenden Beispiel wurde ein 168 Zeichen langer Text mit der ADFGVXZ-Chiffre (Abb. 4.3) verschlüsselt und anschließend versucht, mit der ADFGVX-Analyse-Komponente den Transpositionsschlüssel zu finden.

---

[1] `ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 -.,;:+?=`

```
WIR TESTEN JETZT EINEN TEXT MIT SONDERZEICHEN ();:;,.--+ UND WERDEN
IN DEM SUBSTITUTIONSSCHLUESSEL EBENFALLS SONDERZEICHEN VERWENDEN.
WIRD DIESER TEST FUNKTIONIEREN?
```

Der Text ist in deutscher Sprache. Daher würde man einen Koinzidenzindex nahe 7,6% erwarten. Obwohl der Text mit 134 von 168 ($\approx$ 80%) Zeichen aus den üblichen Zeichen A bis Z besteht, beträgt der Koinzidenzindex in diesem Fall bereits nur 6,96%. Ein Text mit einem niedrigeren Anteil der Zeichen A bis Z könnte dazu führen, dass der Koinzidenzindex soweit sinkt, dass er sich von dem eines polyalphabetisch verschlüsselten Textes nicht unterscheidet.

Ein weiterer Ansatz könnte sein, ein 8x8 Polybius-Quadrat zu verwenden, um Groß- und Kleinbuchstaben des lateinischen Alphabets und zusätzliche Sonderzeichen zu benutzen. Dadurch unterscheidet sich ein `E` von einem `e`.[SS]

Wie sich ein solche Nutzung im Detail auswirkt, müsste genauer evaluiert werden. Da es hierfür Häufigkeits-Statistiken [SS] gibt, bietet ein größeres Substitutions-Alphabet keinen ausreichenden Sicherheitsgewinn.

## 7.2 Doppelte Spaltentransposition

In Kapitel 2.1 wurde bereits erwähnt, dass Fritz Nebel eigentlich eine doppelte Spaltentransposition für die ADFGVX vorgeschlagen hatte, sich seine Vorgesetzten jedoch dagegen entschieden hatten.

Der Grund dieser Entscheidung lag in der Benutzbarkeit von Hand, weil durch eine doppelte Spaltentransposition ein zweiter Transpositionsschlüssel benötigt wird. Darüber hinaus wäre der Aufwand, eine Nachricht zu verschlüsseln und anschließend wieder zu entschlüsseln deutlich gestiegen, da die beiden Spaltentranspositionen nacheinander durchgeführt werden müssen. Die Annahme war, dass die Verschlüsselung bereits mit einer Spaltentransposition stark genug sei, so dass auf die zweite verzichtet werden könnte.

Dass sich Fritz Nebel's Offiziere in diesem Punkt geirrt haben, ist mittlerweile bekannt. Die doppelte Spaltentransposition hätte laut Painvin [Bau, p. 206 und 207] vermutlich dazu geführt, dass die ADFGVX-Chiffre in ihrer Zeit nicht hätte gebrochen werden können.

Die Kombination einer doppelten Spaltentransposition und der darunter liegenden Substitution ist selbst bei korrekter Verwendung mit heutigen Mitteln nur mit sehr viel Text zu lösen. Eine doppelte Spaltentransposition kann zwar, mit den Schlüsseln der Länge `A` und `B`, auf eine einfache Spaltentransposition zurück geführt werden, allerdings ergibt sich daraus eine Schlüssellänge von $A \cdot B$. Wird ein 13 und ein 15 Zeichen langer Schlüssel verwendet, so ergibt sich ein 195 Zeichen langer Schlüssel für die einfache Spaltentransposition.

Bei den in den Kapiteln 2.6.5.1 und 2.6.5.2 verwendeten Beispielen von Nachrichten, deren Längen zwischen 152 und 302 Zeichen betragen, stehen unter diesen Umständen maximal zwei Zeichen pro Spalte. Dies hätte damals ein Brechen unmöglich gemacht, da in den Nachrichten zu wenig Informationen enthalten sind, um wie in Abschnitt 2.6.3 erwähnt, zwischen die Transposition und Substitution zu kommen.

Zum Vergleich: Lasrys Verfahren benötigt bei einer Schlüssellänge von 17 bereits 400 Zeichen im Klartext, um eine Erfolgsrate von über 90% zu erreichen. Diese 400 Zeichen entsprechen 800 Zeichen im Geheimtext, damit stehen ca. 47 Zeichen in jeder Spalte.

Die Auswahl der beiden Schlüssellängen ist bei der doppelten Spaltentransposition sehr wichtig. Wählt man die Längen 15 und 20, ergibt sich daraus nicht die Schlüssellänge 300, da der `ggT` beider Zahlen gleich fünf ist. Deswegen ergibt sich dabei eine Schlüssellänge für die einfache Spaltentransposition von 60. Es ist also wichtig darauf zu achten, dass der `ggT` der beiden Schlüssellängen gleich eins ist.

# 8 Ausblick und Zusammenfassung

Nach intensiver Beschäftigung mit der Chiffre ADFGVX und deren Analyse mit dem Verfahren von Lasry wird diese Bachelorarbeit mit einer Zusammenfassung und einem Ausblick abgeschlossen.

## 8.1 Zusammenfassung

Die in der Einleitung (Kapitel 1) formulierten Ziele dieser Bachelorarbeit konnten erfolgreich umgesetzt werden. Diese Ziele waren:

Z1 Die Implementierung von Lasrys Quellcode in eine neue Analyse-Komponente für CrypTool 2

Z2 Die Implementierung der geforderten Erweiterungen in der bisherigen Chiffrier- und der neuen ADFGVX-Analyse-Komponente

Z3 Das Erstellen von Vorlagen und Hilfen in CrypTool 2

Z4 Die Evaluation der implementierten ADFGVX-Analyse-Komponente

Kapitel 2 stellt die Grundlagen der ADFGVX-Chiffre und zugehörige Themen wie Häufigkeitsanalyse, Koinzidenzindex und heuristische Verfahren vor. Ebenfalls wird die Open-Source-Software CT2 erläutert.

Beginnend mit Kapitel 3 wurde die neu zu implementierende Komponente beschrieben. Kapitel 4.1 und 4.2 führen aus, wie Lasrys bereitgestellter Quellcode aufgebaut ist und wie er in CT2 implementiert wurde. Während der Implementierung wurde der Code optimiert.

Durch die Optimierung konnte die Transpositionsanalyse um den Faktor 30 beschleunigt werden.

In Kapitel 4.3 wird beschrieben, wie die bereits vorhandene ADFGVX-Chiffrier-Komponente erweitert wurde, um 7x7-Matrizen ver- und entschlüsseln zu können. Ergänzend wurde die neue ADFGVX-Analyse-Komponente ebenfalls so erweitert, dass 7x7-Matrizen analysiert werden können.

Die Umsetzung der geforderten Vorlage und mehrsprachigen Hilfen wird in Kapitel 4.4 beschrieben.

Die Evaluation (Kapitel 5) der neuen ADFGVX-Analyse-Komponente zeigt, von welchen Faktoren die Erfolgsrate des Verfahrens abhängig ist. Dabei spielt neben der Schlüssellänge vor allem die Länge der Nachrichten eine entscheidende Rolle. Außerdem wird bestätigt, dass das Verfahren für ungerade Schlüssellängen besser funktioniert als für gerade Schlüssellängen. Der Unterschied ist nicht so gravierend, wie nach Lasrys Aussagen zu befürchten war.

Während der Evaluation wurden 4.250 verschlüsselte Nachrichtensets versucht zu entschlüsseln, die mit Schlüsseln der Längen 5 bis 21 verschlüsselt worden waren. Von den 4.250 Nachrichtensets konnten $\approx 90\%$ entschlüsselt werden. Aus zeitlichen Gründen konnten die Schlüssellängen größer 21 nicht mehr evaluiert werden.

In einer abschließenden Betreuer-Challenge wurden Nachrichtensets bestehend aus 10 Nachrichten, mit einem 23 und 25 Zeichen langen Transpositionsschlüssel, erfolgreich entschlüsselt.

In Kapitel 6 wurden andere Implementierungen der ADFGVX-Analyse besprochen und mit dem Verfahren von Lasry verglichen. Die Vergleiche zeigten auf, dass viele ADFGVX-Lösungen nur auf Brute-Force-Ansätzen basieren und somit nicht für längere Schlüssellängen geeignet sind.

## 8.2 Ausblick

Lasry ermöglichte mit seinem Verfahren aus 2016, dass weitere historische Informationen über den Ablauf des Ersten Weltkrieges bekannt und analysiert werden konnten. Mit der Umsetzung seines Verfahrens in eine CT2-Komponente ist ein aktueller und neuer Ansatz, eine ältere Chiffre zu brechen, in die Open-Source-Software hinzugefügt wurden. Da CT2 inzwischen zu den weltweit am weitesten verbreiteten Programmen im Bereich Kryptografie und Kryptoanalyse gehört, steht auch diese Komponente zukünftig für Lehr- und Ausbildungszwecke zur Verfügung.

Um das Verfahren noch zu verbessern, könnte ein Wörterbuchangriff ergänzt werden. Durch diesen könnten Schlüssel, auch solche mit noch größeren Längen, schneller gefunden werden, wenn ihre Permutation einem der Wörter entspricht.

Auch könnte evaluiert werden, wie sich eine Veränderung der Kostenfunktion auf die Erfolgsrate auswirken würde. Aktuell wird die Kostenfunktion aus dem Koinzidenzindex für Mono- und Bigramme wie folgt berechnet:

Aktuell: $Score_{aktuell} = 6.000 \cdot IoC1 + 180.000 \cdot IoC2$

Die Kostenfunktion würde dann so geändert, das sich die Werte dem ïdealen"Kostenwert des Koinzidenzindex der natürliche Sprache annähern. Je geringer die Abweichung, desto besser der Schlüssel:

Ausblick: $Score_{neu} = |Score_{ideal} - Score_{aktuell}|$

Die Statistiken über Mono- und Bigramme können mit ausreichend Text selbst erstellt werden.

Die Art und Weise, den Score zu berechnen, bleibt erhalten, er wird aber neu interpretiert. Dabei kann man erwarten, dass die Analyse für gerade Schlüssellängen optimiert wird, weil Permutationen, die einen deutlich höheren Koinzidenzindex erreichen als erwartet, weniger in Frage kommen.

# Literaturverzeichnis

[AA] ALKHALID, A. S. und A. O. ALKHFAGI. `http://iieng.org/images/proceedings_pdf/U1215011.pdf` (besucht: 09.10.2018).

[Bau] BAUER, C. P.: *Secret History: The Story of Cryptology.* `https://books.google.de/books?id=f4PNBQAAQBAJ&printsec=frontcover&hl=de#v=onepage&q&f=false` (besucht: 12.10.2018).

[Ben15] BENESCH, A.: *Ver- und Entschlüsselungsmethoden im Ersten Weltkrieg. Kryptografieeinsatz an der Westfront und in der Nordsee*, 2015. `https://www.grin.com/document/310524` (besucht: 08.08.2018).

[Cha] CHATFIELD, D. `https://github.com/danielchatfield/adfgvx_solver/tree/master` (besucht: 09.10.2018).

[Col] COLLODI, C.: *Pinocchio – Die Geschichte vom hölzernen Bengele.* `https://www.ebook.de/de/product/19739556/carlo_collodi_pinocchio_die_geschichte_vom_hoelzernen_bengele.html` (besucht: 02.10.2018).

[Cry] CRYPTOOL: *IPlugin call flow.* `https://www.cryptool.org/trac/CrypTool2/wiki/IPluginHints` (besucht: 02.10.2018).

[Eil] EILERS, C.: *Grundlagen der Kryptographie, Teil 5: One-Time-Pad.* `https://www.ceilers-news.de/serendipity/758-Grundlagen-der-Kryptographie,-Teil-5-One-Time-Pad.html` (besucht: 04.10.2018).

[Fri34] FRIEDMAN, W. F.: *General solution of the ADFGVX cipher system. Technical paper of the Signal Intelligence Section*, 1934. `https://www.nsa.gov/news-features/declassified-documents/friedman-documents/assets/files/publications/FOLDER_269/41784769082379.pdf` (besucht: 08.08.2018).

[Fri39] FRIEDMAN, W. F.: *Military Cryptanalysis, Part IV transposition and fractionating systems*, 1939. `https://www.nsa.gov/news-features/declassified-documents/friedman-documents/assets/files/publications/FOLDER_452/41749819078904.pdf` (besucht: 08.08.2018).

[hex] HEXPRESSO. `https://github.com/hexpresso/WU-2015/tree/master/hackingweek-2015/crypto/crypto1` (besucht: 09.10.2018).

[Hub15] HUBER, T. C.: *Windows Presentation Foundation: Das umfassende Handbuch. Rheinwerk Computing*, 2015.

[Kon85] KONHEIM, A. G.: *Cryptanalysis of ADFGVX encipherment systems, Advances in Cryptology. Springer, 339 bis 341*, 1985.

[Krya] KRYPTOGRAFIE.DE: *Häufigkeitsverteilung.* `http://kryptografie.de/kryptografie/kryptoanalyse/haeufigkeitsverteilung.htm` (besucht: 02.10.2018).

[Kryb] KRYPTOGRAFIE.DE: *Koinzidenzindex.* `http://kryptografie.de/kryptografie/kryptoanalyse/koinzidenzindex.htm` (besucht: 02.10.2018).

[Las16] LASRY, G.: *Deciphering ADFGVX messages from the Eastern Front of World War I*, 2016. `http://dx.doi.org/10.1080/01611194.2016.1169461` (besucht: 08.08.2018).

[Mic] MICROSOFT: *lock-Anweisung (C#-Referenz).* `https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/lock-statement`(besucht: 07.10.2018).

[MvOV96] MENEZES, A., P. VAN OORSCHOT und S. VANSTONE: *Handbook of Applied Cryptography, CRC Press, Kapitel 7, S. 245ff*, 1996. `http://cacr.uwaterloo.ca/hac/about/chap7.pdf` (besucht: 02.10.2018).

[Pat] PATEL, A.: *Amit's A* Pages.* `http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html` (besucht: 02.10.2018).

[Pom] POMMERENING, K.: *Kryptologie.* `https://www.staff.uni-mainz.de/pommeren/Kryptologie/Klassisch/7_Perfekt/redund.pdf` (besucht: 02.10.2018).

[Pra] PRACTICALCRYPTOGRAPHY.COM: *Unicity Distance.* `http://practicalcryptography.com/cryptanalysis/text-characterisation/statistics/` (besucht: 08.08.2018).

[Rsh] RSHELDIII, B. `https://gist.github.com/rsheldiii/3000444` (besucht: 09.10.2018).

[Ruia] RUIJL, B. `https://www.tapatalk.com/groups/crypto/analytically-solving-the-adfgvx-cipher-t428.html` (besucht: 07.10.2018).

[Ruib] RUIJL, B. `https://github.com/benruijl/adfgvxSolver` (besucht: 07.10.2018).

[Sch] SCHRADER, R.: *Forschungsschwerpunkt Verkehr.* `http://www.zaik.de/AFS/publications/annualreports/99-00/html/node24.html` (besucht: 02.10.2018).

[Sha49] Shannon, C. E.: *Communication Theory of Secrecy Systems.*, 1949. `http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf` (besucht: 08.08.2018).

[SS] Sommer-Stumpenhorst, N. `http://graf-ortho.de/html/buchst.html` (besucht: 09.10.2018).

[The17] Theis, T.: *Einstieg in C# mit Visual Studio 2017. Rheinwerk Computing*, 2017.

[Ver18] Versteeg, O.: *Visualisierung und Implementierung von Betriebsmodi von Blockchiffren für CrypTool 2*, 2018.

[Wika] Wikipedia. `https://de.wikipedia.org/wiki/Refactoring` (besucht: 06.10.2018).

[Wikb] Wikipedia: *ADFGVX.* `https://de.wikipedia.org/wiki/ADFGX#Anekdote` (besucht: 02.10.2018).

[Wikc] Wikipedia: *Simulated Annealing.* `https://de.wikipedia.org/wiki/Simulated_Annealing` (besucht: 24.09.2018).

[Win] Winsley, J. `https://github.com/glitchassassin/adfgvx-solver` (besucht: 07.10.2018).

[wir] wiremask.eu. `https://wiremask.eu/writeups/hackingweek-2015-crypto-1/` (besucht: 09.10.2018).

# Versicherung an Eides Statt

Ich, Dominik Vogt, Matrikelnummer 30210970, wohnhaft in 34292 Ahnatal, versichere an Eides Statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen übernommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe.

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

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

Kassel, 14. Oktober 2018

Dominik Vogt