Mathematische Methoden der Analyse einfacher Kryptosysteme

Facharbeit
im Fach
Mathematik (Leistungskurs)

vorgelegt von
Jan Olligs

Gymnasium am Moltkeplatz, Krefeld

Schuljahr 2002/2003

Krefeld, den 26. März 2003

Inhaltsverzeichnis

1 Über diese Facharbeit

Seit ungefähr 50 Jahren, nach der Axiomatisierung der Informationstheorie durch SHANNON[1],[2] hat sich die Kryptographie von einer Geheimwissenschaft zu einem Teilgebiet der Mathematik, das sehr exakt untersucht werden kann, entwickelt.

In diesen 50 Jahren wurden viele systematische Fehler in Kryptosystemen entdeckt und dementprechend verfeinerte Chiffrierverfahren entwickelt. Als Reaktion auf diese verfeinerten Kryptosysteme wurden selbstverständlich weitere Analyseverfahren entwickelt, die zu beschriben jedoch den Rahmen dieser Facharbeit sprengen würde.

In dieser Facharbeiten sollen einfache mathematische Methoden der Kryptoanalytik, d. h. des Entschlüsselns verschlüsselter Texte ohne Zuhilfenahme des Schlüssels, beschrieben werden. Dabei wird deutlich werden, wie man unter Verwendung einfacher schulmathematischer Methoden Chiffren brechen kann. Die verwendeten Methoden stammen hauptsächlich aus den Bereichen Statistik, Zahlentheorie und Kombinatorik, die auch in der Schule behandelt werden. In der vorliegenden Facharbeit wird weiterhin deutlich werden, wie unsicher einige bekannte und häufig verwendete Verschlüsselungsverfahren sind; der Leser sollte nach der Lektüre dieser Arbeit befähigt sein, mit den beschriebenen Verfahren verschlüsselte Texte selbst zu entschlüsseln.

Um die Eigenständigkeit und kreative Leistung in dieser Facharbeit nachzuweisen, sind einige der im Folgenden beschriebenen Verfahren als teilweise automatisierte Analyseprogramme bzw. -methoden in `C++` umgesetzt. Die zugehörigen Quellcodes befinden sich im Anhang dieser Facharbeit.

Leider konnte ich im Rahmen dieser Facharbeit nicht alle Themen, über die ich mir vorgenommen hatte zu schreiben, behandeln. Es stellte sich heraus, dass schon die Beschreibung und Erläuterung der behandelten Verfahren den Rahmen der Facharbeit ausfüllt; ich musste einige Sektionen aus Platzgründen streichen.

An dieser Stelle kann ich daher nur auf das Literaturverzeichnis verweisen. Von den aufgeführten Werken werden drei für den Leser dieser Facharbeit besonders interessant sein: Das Buch von SINGH ([Sin.Code]) ist sehr einfach geschrieben, und verzichtet, so weit wie möglich, auf Mathematik. BAUERs Buch ([Bau.Entz]) ist sehr umfangreich – die Tabellen aus dem Anhang sind ihm entnommen – das Buch ist allerdings sehr formellastig und manchmal leicht unsystematisch. Für Informatiker dürfte WOBSTs Buch ([Wob.Aben]) von Interesse sein; es beschäftigt sich auch mit komplizierteren Algorithmen.

[1] SHANNON, CLAUDE – Amerikanischer Mathematiker (1916–), entwickelte 1936–1940 die mathematische Theorie der Informationssysteme (veröffentlicht 1949). s. [New.Encyc], S. 257

[2] s. [Wel.Codes] als Einführung in die axiomatische Informationstheorie

2 Frühe Kryptosysteme

2.1 Die Skytala – eine Transpositionschiffre

Eine der ersten bekannten Methoden, das Lesen von Nachrichten durch Unberechtigte zu verhindern, wird den Spartanern zugeschrieben. Wie der griechische Historiker PLUTARCH berichtet, besaßen Sender und Empfänger einer geheimen Botschaft je eine sog. *Skytala*, einen zylindrischen Holzstab mit festgelegtem Umfang.[3] Der Sender wickelte nun einen schmalen Streifen Pergament helixförmig um den Zylinder und schrieb die Botschaft zeilenweise darauf. Wurde der Streifen abgewickelt, so war die Botschaft unlesbar, da Zeichen, die im verschlüsselten Text aufeinander folgten, in der ursprünglichen Botschaft lediglich in der selben Spalte, jedoch in unterschiedlichen Zeilen standen. Die Botschaft konnte nun nur mit einem Holzstab des gleichen Durchmessers, wie ihn der Empfänger besaß, entschlüsseln.

Mathematisch ist diese Methode mit einem im zweiten Weltkrieg verwendeten Verfahren äquivalent. Die sogenannte Kastenmethode besteht darin, dass eine Geheimbotschaft zeilenweise in einen Kasten eingetragen und spaltenweise ausgelesen wird. Als Schlüssel müssen sich Sender und Empfänger auf eine natürliche Zahl s, die Spaltenanzahl, einigen.

Eine Eigenschaft der Skytala ist, dass die Buchstaben des Klartextes alle im verschlüsselten Text auftreten; sie befinden sich lediglich an anderen Positionen. Chiffren mit dieser Eigenschaft bezeichnet man als *Transpositionschiffren*; wie in der Überschrift dieses Abschnittes bemerkt, ist die Verschlüsselung mit der Skytala also eine Transpositionschiffre.

2.2 Die Caesar-Verschlüsselung

Laut SUETON[4] verschlüsselte CAESAR[5] geheime Nachrichten, indem er jeden Buchstaben durch denjenigen ersetzte, der im Alphabet drei Stellen nach ihm folgte.[6] Dabei wird nach dem „`Z`“ beim „`A`“ weitergezählt. Aus dem Text „`caiusiuliuscaesar`“ wird so zum Beispiel „`FDLXVLXOLXVFDHVDU`“.

Bei der Caesar-Verschlüsselung lassen sich Klartext- und Geheimtextalphabet zu einem einfachen Chiffrierschema gegenüberstellen:

Klartext:	`abcdefghijklmnopqrstuvwxyz`
Geheimtext:	`DEFGHIJKLMNOPQRSTUVWXYZABC`

Bei diesem Chiffrierschema stehen, wie in der Kryptologie allgemein üblich, Kleinbuchstaben

[3] s. [Beu.Krypt], S. 3–4
[4] SUETON, röm. Geschichtsschreiber
[5] CAESAR, CAIUS IULIUS (100–44 a. Chr. n.), röm. Feldherr und Staatsmann
[6] s. [Beu.Krypt], S. 4

für Klartext und Großbuchstaben für Geheimtext. Diese Gegenüberstellung von Klartext- und Geheimtextbuchstaben ist bei allen monoalphabetischen Substitutionen[7] über dem standardlateinischen Alphabet möglich.

Heutzutage wird unter einer *Caesar-Chiffrierung* im Allgemeinen[8] eine Verschiebung um n Buchstaben verstanden. Mathematisch ist dies bei einem modernen lateinischen Standardalphabet eine Addition plus n *modulo* 26.

2.3 Monoalphabetische Substitution

Eine Erweiterung der allgemeinen Caesar-Chiffre (d. h. einer Caesar-Chiffre mit variablen Abstand) ist die *monoalphabetische Substitution*[9]. Bei der monoalphabetischen Substitution wird jeder Buchstabe im Alphabet durch einen zweiten Buchstaben ersetzt. Es ist auch möglich, dass ein Buchstabe für sich selbst steht.

Als Schlüssel eignet sich gut eine Gegenüberstellung, wie wir sie oben[10] kennengelernt haben. Zum Beispiel wäre

Klartext:	`abcdefghijklmnopqrstuvwxyz`
Geheimtext:	`DMGOSLRXWEVCJTPIAHZYUKBQFN`

ein solcher Schlüssel. Der Text „`unknackbar`“ würde mit diesem Schlüssel als „`UTVTDGVMH`“ verschlüsselt werden. Hierbei ist das „`u`“ im Klartext ein „`U`“ im Geheimtext; eine Ersetzung eines Buchstabens durch ihn selber ist also, wie oben bemerkt, möglich.

2.4 Angriffe auf frühe Kryptosysteme

2.4.1 Die Entschlüsselung durch Brute-Force Angriff

Bei einem sog. *Brute-Force Angriff*, auch *Exhaustive Search*[11] genannt, werden alle möglichen Schlüssel der Reihe nach darauf getestet, ob sie den gegebenen verschlüsselten Text zu einem sinnvollen Klartext entschlüsseln. Auch wenn die Erkennung eines „sinvollen Klartextes“ als eine Aufgabe erscheint, zu der ein Computer nicht befähigt ist, so kann Computern mit Hilfe von Wörterbüchern und statistischen Eigenheiten natürlicher Sprachen doch die Lösung einer solchen Aufgabe ermöglicht werden.

[7] s. Abschnitt 2.3, S. 4

[8] Einige Autoren bezeichnen eine allgemeine monoalphabetische Chiffrierung (s. 2.3, S. 4) über einem lateinischen Standardalphabet als *Caesar*, s. z. B. [Lei.Entz]

[9] genauer: monoalphabetische Substitution über dem standardlateinischen Alphabet

[10] s. Abschnitt 2.2, S. 3

[11] *engl.*: Ausschöpfende Suche; *Anm. d. Aut.*

Es ist einleuchtend, dass die Entschlüsselung mittels Brute-Force hauptsächlich für Verschlüsselungsmethoden mit wenigen Schlüsseln prädestiniert ist. Auf andere Chiffren lassen sich wesentlich effektivere Algorithmen anwenden. An Hand der oben beschiebenen Beispiele bedeutet dies: Die Caesar-Verschlüsselung mit 26 möglichen Schlüsseln lässt sich mittels Brute-Force einfach entschlüsseln; für eine monoalphabetische Substitution mit $26! \approx 4 \times 10^{26}$ möglichen Schlüsseln würde sich ein Brute-Force Angriff als wesentlich schwieriger erweisen.[12]

Am Beispiel der Caesar-Chiffre soll nun die Entschlüsselung eines Textes mittels Brute-Force beschrieben werden. Uns sei der Text „`UZVJZJKVZEXVYVZDVIKVOK`“ in die Hände gefallen, von dem wir wissen, dass er mit Hilfe einer Caesar-Chiffre verschlüsselt wurde. Auch wenn es abwegig erscheint, so ist die Verschlüsselungsmethode dem Angreifer meist bekannt; es gibt sogar statistische Methoden das verwendete Verfahren festzustellen.[14] Wir stellen nun eine Tabelle auf, in der wir das zu jedem möglichen Schlüssel gehörige Dechiffrat festhalten:

Schlüssel	Klartext	Schlüssel	Klartext
A	`uzvjzjkvzexvyvzdvikvok`	N	`hmiwmwximrkilimqivxibx`
B	`tyuiyijuydwuxuycuhjunj`	O	`glhvlvwhlqjhkhlphuwhaw`
C	`sxthxhitxcvtwtxbtgitmi`	P	`fkgukuvgkpigjgkogtvgzv`
D	`rwsgwghswbusvswasfhslh`	Q	`ejftjtufjohfifjnfsufyu`
E	`qvrfvfgrvatrurvzregrkg`	*R*	*`diesisteingeheimertext`*
F	`puqeuefquzsqtquyqdfqjf`	S	`chdrhrsdhmfdgdhldqsdws`
G	`otpdtdeptyrpsptxpcepie`	T	`bgcqgqrcglecfcgkcprcvr`
H	`nsocscdosxqoroswobdohd`	U	`afbpfpqbfkdbebfjboqbuq`
I	`mrnbrbcnrwpnqnrvnacngc`	V	`zeaoeopaejcadaeianpatp`
J	`lqmaqabmqvompmqumzbmfb`	W	`ydzndnozdibzczdhzmozso`
K	`kplzpzalpunlolptlyalea`	X	`xcymcmnychaybycgylnyrn`
L	`jokyoyzkotmknkoskxzkdz`	Y	`wbxlblmxbgzxaxbfxkmxqm`
M	`injxnxyjnsljmjnrjwyjcy`	Z	`vawkaklwafywzwaewjlwpl`

Bei Entschlüsselung mit dem Schlüssel „`R`“ ergint sich also ein sinnvoller Text. Der Code ist geknackt; der Ursprungstext lautete „Dies ist ein geheimer Text“.

[12] Durch eine Kombination aus Brute-Force und Polygrammanalyse[13] wird ein Angriff jedoch dennoch möglich; s. u. (Abschnitt 2.4.2, S. 7

[14] Dies ist SHANNONs Maxime: „Der Feind kennt das benutzte System“, s. [Bau.Entz], S. 224

2.4.2 Die Entschlüsselung durch Polygrammanalyse

Als *Polygramm* bezeichnet man eine Gruppe aufeinanderfolgender Buchstaben. *N-Gramme* sind Folgen von N Buchstaben. Zum Beispiel wäre in dem Text „EINBEISPIELTEXT“ „P“ ein Monogramm, „EI“ ein Bigramm, „NBE“ ein Trigramm, etc.; jede dieser Kombinationen wäre allgemein ein Polygramm.

In natürlichen (d.h. menschlichen, nicht „künstlich“ konstruierten) Sprachen treten nicht alle Polygramme mit der gleichen Häufigkeit auf. In vielen europäischen Sprachen tritt z. B. der Buchstabe „E“ am häufigsten auf; im Deutschen ist dieser Effekt besonders ausgeprägt: ungefähr jeder fünfte Buchstabe ist ein „E“.[15] Außerdem gibt es gewisse Bigramme, die im Deutschen überhaupt nicht auftreten, z. B. „QX“.

Diese für eine Sprache charakteristische statitstische Verteilung kann man sich bei der Entschlüsselung, besonders bei der Entschlüsselung von monoalphabetischen Chiffren, zur Hilfe nehmen.

Das naivste Verfahren wäre, die Buchstaben in der natürlichen Sprache und dem Chiffrat nach ihrer Häufigkeit zu sortieren und dann jeweils die häufigsten, zweithäufigsten, etc. Buchstaben zu Paaren zusammenzustellen. Insbesondere bei kurzen Texten, bei denen die Hüfigkeiten stark von den Standardhäufigkeiten abweichen können, ist dieses Verfahren nicht zu empfehlen.

Wenn man einen leistungsfähigen Computer zur Verfügung hat, kann man auch einen auf der Häufigkeitenanalyse basierenden Brute-Force Algorithmus durchführen. Dazu wird zu jeder möglichen Permutation π_{26} der 26 Buchstaben des Alphabetes mit den Häufigkeiten $p_K(n)$ des n-ten Buchstaben des Klartextalphabetes (z. B. $n = 0$ für „A“) und $p_C(n)$ des n-ten Buchstaben des Chiffratalphabetes das *Abstandsquadrat*[16]

$$d(\pi_{26}) = \sum_{i=0}^{25} \left[p_K(i) - p_C\left(\pi_{26}(i)\right)\right]^2$$

berechnet; die Permutation mit dem niedrigsten Wert für d und solche innerhalb eines gewissen relativen Spielraumes (z. B. bis zu 10% mehr) werden als mögliche Geheimtextalphabete ausgegeben. Das Verfahren scheitert leider an der Leistungsfähigkeit moderner Computersysteme; unten wird jedoch ein ähnlicher Ansatz mit größeren Erfolgsaussichten vorgestellt.

Effektiver ist das schon erwähnte Verfahren, nach nicht auftretenden Kombinationen von vorhandenen Buchstaben zu suchen. Im Deutschen ist der Buchstabe „Q“ ein sehr guter Ansatzpunkt. Er taucht nur in der Bigrammform „QU“ auf; man kann dieses Bigramm außerdem anhand der

[15] auch auf Grund der Konvention, Umlaute durch zwei Buchstaben zu ersetzen, z. B. „ü“ durch „ue“, s. Anhang B, S. 15

[16] praktisch wird nur ein Teil verwendet, dies dient jedoch nur der Erhöhung der Effizienz der Berechnungen; das Ergebnis blebt das gleiche

Bigrammhäufigkeiten und der Häufigkeit des „U“ sehr gut von anderen Bigrammen mit ähnlichen Eigenschaften unterscheiden.

Das mit Abstand am häufigsten verwendete Verfahren partitioniert die Menge der Buchstaben einer Sprache nach ihrer Häufigkeit, so dass Elemente aus verschiedenen Partitionen (auch *Cliquen* genannt) in „praktisch“ nur einer Reihenfolge vorkommen können, und Elemente aus der selben Partition in beiden möglichen Reihefolgen erscheinen können. Diese Partitionierung differiert selbstverständlich je nach Textlänge; bei kurzen Texten gibt es weitaus größere Abweichungen in den Häufigkeiten und deshalb weniger Partitionen als bei längeren Texten.

Die Partitionierung verrigert die Anzahl der möglichen Permutationen erheblich: Bei der im Deutschen geläufigen Partition {e}{n}{irsat}{dhu}{lgocm}{bfwkz}{pv}{jyxq} reduziert sich die Anzahl der möglichen Permutationen von $26! \approx 4,03 \cdot 10^{26}$ auf $1! \cdot 1! \cdot 5! \cdot 3! \cdot 5! \cdot 5! \cdot 2! \cdot 4! \approx 4,98 \cdot 10^{8}$, d. h. um einen Faktor von $8,10 \cdot 10^{17}$. Der Vorteil wird deutlich, wenn man berechnet, wie lange ein Computer mit einer Taktgeschwindigkeit von 1 GHz brauchen würde, um alle Kombinationen auszuprobieren. Bei einer sehr optimistischen Schätzung von 100 Taktzyklen pro Permutation und einer Ausnutzung der vollen Prozessorleistung für diese Aufgabe würde der Computer ohne Partitionierung $1,31 \cdot 10^{12}$ Jahre rechnen, also länger als das Alter des Universums. Bei einer Partitionierung wäre derselbe Computer nach 49,8 s mit seiner Aufgabe fertig.

Bei allen mit der Polygrammanalyse zusammenhängenden Verfahren ist zu beachten, dass die Hüfigkeitenverteilung je nach untersuchter Textart schwanken kann. Es ist zum Beispiel nicht zu erwarten, dass militärische Einsatzbefehle und philosophische Abhandlungen die gleichen Häufigkeitenverteilungen aufweisen. Im Allgemeinen finden sich jedoch genug Referenztexte, um Häufigkeitsverteilungen aufzustellen.

2.4.3 Die Entschlüsselung durch negative Mustersuche

Wenn ein Wort – oder besser, eine Zeichenfolge – aus dem zu einem verschlüsselten Text gehörigen Klartext bekannt ist, kann man bei den meisten Verschlüsselungsverfahren, so auch bei der Caesar-Chiffrierung, gewisse Schlüssel ausschließen.

Es sei der Text „DHMEZBGDZADQDEEDJSHUDUDQEZGQDM...“ mit einer Caesar-Chiffrierung verschlüsselt. Wir wissen, dass das Wort „einfacheabereffektive“ im Klartext vorkommt. Nun ist es unmöglich, dass einer der Buchstaben durch sich selbst verschlüsselt wurde; in diesem Fall wäre die Verschlüsselung eine Identität und wir könnten den Klartext direkt aus dem Chiffrat herauslesen.

Wir stellen nun das Chiffrat mit dem jeweils verschobenen Klartext gegenüber:

```
  DHMEZBGDZADQDEEDJSHUDUDQEZGQDM...
→einfacheabereffektive
   einfacheaberffektive
    einfacheaberffektive
     einfacheabereffektive
      einfacheabereffektive
       einfacheabereffektive
        einfacheabereffektive
         ...
```

Die kursiven Buchstaben stimmen mit denen im verschlüsselten Text überein; die Position des Klartextwortes kann dementsprechend nicht mit der korrekten Position übereinstimmen.

Wir betrachten nun die Verschlüsselung. Wie oben erwähnt, wird die Caesar-Chiffrierung durch eine Addition *modulo* 26, wobei jeder Buchstabe mit einer nichtnegativen ganzen Zahl entsprechend seiner Position im Alphabet identifiziert wird (`A` $= 0$, `B` $= 1$, etc.). Die Verschlüsselung lässt sich somit durch die Gleichung $K + S \equiv V \bmod 26$ darstellen, wobei K für den Klartextbuchstaben, S für den Schlüsselbuchstaben und V für den Buchstaben des verschlüsselten Textes steht. Diese Gleichung kann in die Form $S \equiv V - K \bmod 26$ gebracht werden; aus dem (bekannten) Geheimtextbuchstaben und dem (vermuteten) Klartextbuchstaben kann somit der Schlüsselbuchstabe errechnet werden.

Da es meist mehrere mögliche Klartextpositionen gibt, muss der Schlüssel nun noch auf Korrektheit überprüft werden. Sofern sich nicht verschiedene Schlüsselbuchstaben aus den Geheimtext-/Klartextbuchstabenpaaren in der vermuteten Position ergeben, muss der Geheimtext mit dem erhaltenen Schlüssel probeweise entschlüsselt werden und der so erhaltene Klartext daraufhin untersucht werden, ob er in einer natürlichen Sprache verfasst wurde. Maschinell kann dies beispielsweise mit dem im Folgenden vorgestellten Kappa-Test erfolgen.

Es lassen sich sehr häufig wahrscheinliche Klartextpassagen aufstellen; ein gern aufgeführtes Beispiel sind die britischen Kryptoanalytiker im zweiten Weltkrieg. Beim Brechen der deutschen Codes benutzten sie den Umstand, dass praktisch alle Texte mit „`heilhitler`“ abgeschlossen wurden; ebenso lassen sich standartisierte Formeln wie „`sehrgeehrte`“, „`mitfreundlichengruessen`“ oder „`bezugnehmendaufihrschreibenvom`“ in praktisch jedem Text auffinden. Sie sind besonders gefährlich, wenn, wie bei den erwähnten Beispielen, ihre ungefähre Position im Text bekannt ist.

Da die Überprüfung von Übereinstimmungen in Textpassagen effektiv maschinell durchgeführt werden kann, eignet sich die negative Mustersuche besonders zur maschinellen Dechiffrierung von

verschlüsselten Texten. Leider lässt sich die negative Mustersuche nicht bei allen Chiffrierverfahren anwenden; ansonsten ist sie sehr effektiv und leicht automatisierbar.

3 Polyalphabetische Chiffrierungen

Bei den bisher betrachteten Chiffrierverfahren wurde jeder Buchstabe unabhängig von seiner Position im Text mit dem selben Schlüssel, oder anders ausgedrückt, nach dem selben Alphabet, verschlüsselt. Chiffren mit dieser Eigenschaft werden als *monoalphabetische Chiffren* bezeichnet.

Im Gegensatz zu diesen monoalphabetischen Chiffren werden bei den im Folgenden betrachteten sog. *polyalphabetischen Chiffren* Buchstaben je nach ihrer Position mit unterschiedlichen Schlüsseln, oder Alphabeten, verschlüsselt.

3.1 Der Vigenère-Code

Der Vigenère-Code wurde im Jahre 1586 von dem französichen Diplomaten Blaise de Vigenère der Öffentlichkeit zugänglich gemacht. Es handelt sich um eine polyalphabetische Chiffrierung, deren einzelne Chiffrierschritte Caesar-Chiffren sind. Der Schlüssel ist ein Wort; es kann sich um ein künstliches, wie zum Beispiel „`DFERS`“ oder um ein natürliches, wie „`HUND`“ handlen.

Sei $\mathrm{K} = (k_i)_{i \in \mathbb{I}_n}$ das Schlüsselwort der Länge $n \in \mathbb{N}$, und der Klartext $\mathrm{P} = (p_i)_{i \in \mathbb{I}_m}$ der Länge $m \in \mathbb{N}$, wobei die k_is und die p_is einzelne Buchstaben darstellen, d. h., $k_i, p_i \in \mathbb{N}_{25}^0$ und $\mathbb{I}_n$ eine null-indizierte Indexmenge ist, also alle ganzen Zahlen zwischen 0 (inklusive) und n (exklusive) enthält. Die Verschlüsselung ist dann eine Abbildung

$$\begin{aligned} e : \left(\mathbb{N}_{25}^0\right)^m &\longmapsto \left(\mathbb{N}_{25}^0\right)^m \\ (p_i)_{\mathbb{I}_m} &\longrightarrow (p_i + k_{i \bmod n} \bmod 26)_{\mathbb{I}_m} \end{aligned}$$

Anschaulich bedeutet dies, dass wir über den Klarext das Schlüsselwort so oft hintereinanderschreiben, bis über jedem Klartextbuchstaben ein Schlüsselbuchstabe steht. Wir erhalten dann den zu einem Klartextbuchstaben gehörigen Geheimtextbuchstaben, indem wir ihn nach dem Caesar-Verfahren mit dem über ihm stehenden Schlüsselbuchstaben verschlüsseln.

Ein Beispiel: Wir wollen den Text „`eininteressantesverfahren`“ mit dem Schlüssel „`HUND`“ nach dem Vigenère-Verfahren verschlüsseln:

Schlüssel:	`HUNDHUNDHUNDHUNDHUNDHUNDHUND`
Klartext:	`eininteressantesverfahren`
Geheimtext:	`LCALUNRULMFDUNRVCYEIHBEHU`

Obwohl das Verfahren sicherer als die Caesar-Methode ist – es ist leicht einzusehen, dass es sich mithilfe der bisher beschriebenen Methoden nicht brechen lässt –, werden wir im folgenden Abschnitt dennoch Methoden kennenlernen, mit denen seine scheinbare Sicherheit hintergangen werden kann.

3.2 Angriffe auf polyalphabetische Chiffrierungen

Der Vorteil von polyalphabetischen Chiffrierungen ist, dass sie mehr Möglichkeiten für die Schlüsselwahl als die entsprechenden monoalphabetischen Chiffrierungen bieten, und somit ein Brute-Force-Angriff fast unmöglich gemacht wird. Es kann jedoch ausgenutzt werden, dass sie durch die begrenzte Schlüssellänge zumeist eine gewisse Periodizität besitzen;[17] die Vigenère-Chiffre bietet aus heutiger Sicht nicht mehr Sicherheit als eine monoalphabetische Chiffre.

Die Verfahren zum Brechen der Vigenère-Chiffrierung finden meist nur die Schlüssellänge, der Text kann dann in Teiltexte aufgespalten werden, die jeweils mit einem Buchstaben verschlüsselt wurden, und deshalb nach der Abstandsquadratsmethode entschlüsselt werden können. Wenn der Schlüssel zum Beispiel 3 Buchstaben hat, so teilt man den Text in drei Teiltexte (oder besser, Buchstabensequenzen auf): Buchstaben (1, 4, 7, ...), Buchstaben (2, 5, 8, ...) und Buchstaben (3, 6, 9, ...), die einzeln entschlüsselt werden können.

3.2.1 Schlüssellänge durch Wiederholungen nach Kasiski

Wir betrachten unser Beispiel für die Vignére-Codierung auf Seite 9. Bei genauerer Betrachtung fällt auf, dass das Trigramm „`nte`“ zweimal auftaucht (in „`i(nte)ressa(nte)s`“), beide Male mit dem Schlüsselsegment „`HUN`“ verschlüsselt wird. Dies äußert sich im verschlüsselten Text darin, dass sich beide Male „`UNR`“ ergibt.

Wenn man nun davon ausgeht, dass beide Trigramme tatsächlich mit demselben Segment des Schlüssels verschlüsselt wurden, so muss der Abstand zwischen ihnen ein Vielfaches der Länge des Schlüsselwortes sein. In unserem Beispiel beträgt der Abstand 8 Buchstaben, also muss die Schlüssellänge ein Teiler von 8 sein. Da $8 = 2^3$ gilt, ist es eine der Längen 1, 2, 4 oder 8; die Schlüssellänge 4 stimmt mit der tatsächlichen Schlüssellänge überein.

Es lässt sich natürlich nicht feststellen, ob sich eine Wiederholung aufgrund eines solchen Zusammentreffens oder aus reinem Zufall ergibt; als Faustregel läst sich sagen, dass es um so wahrscheinlicher eine wirkliche Wiederholung ist, je länger das sich wiederholende Polygramm

[17] Eine Ausnahme bilden die sog. *One-Time Keypads*, Schlüsselwürmer mit der gleichen Länge wie der zu verschlüsselnde Text. One-Time Keypads sind nachgewiesernermaßen die einzige 100%ig sichere Chiffriermethode; auch sie können jedoch gebrochen werden, falls unterschiedliche Texte mit dem selben Keypad verschlüsselt werden. Siehe hierzu [Sin.Code], S. 376

ist. In der Praxis wird man in verschlüsselten Texten mindestens 5–10 Wiederholungen (bei kurzen Texten) finden; anhand der Primfaktorzerlegung lässt sich ein wahrscheinliches Vielfaches der Schlüssellänge finden, meistens sogar die Schlüssellänge selbst.

Dies kann an einem (konstruierten) Beispiel verdeutlicht werden. Angenommen, wir hätten in einem mit dem Vigenère-Verfahren verschlüsselten Text folgende Wiederholungen entdeckt:

Polygramm	Abstand		Primfaktorzerlegung
SDF	15	=	$3 \cdot 5$
SFDEDG	20	=	$2 \cdot 2 \cdot 5$
DFEH	91	=	$7 \cdot 13$
DHE	30	=	$2 \cdot 3 \cdot 5$
DJNE	35	=	$5 \cdot 7$
DFG	16	=	$2 \cdot 2 \cdot 2 \cdot 2$

Die Primfaktorzerlegung deutet auf eine Schlüssellänge von 5 hin; die Polygramme bei „`DFEH`“ und „`DFG`“ sind also „Ausrutscher“. Diese „Ausrutscher“ zu bestimmen ist eine Sache der Intuition oder des richtigen Algorithmus. Diesen Algorithmus hier anzuführen würde zu weit führen; er findet sich jedoch im Vigenère-Cracker im Anhang wieder.

Zu Ehren eines ihrer Entdecker wird diese Methode *Kasiski*[18]*-Methode* genannt.

3.2.2 Schlüssellänge mit Kappa-Index

Wir betrachten einen deutschen Text[19], und vergleichen ihn mit seinem um einige Buchstaben verschobenen Bild:

```
weitdraussenindenunerforschteneinoedeneinestotalausdermodegekommene
ussenindenunerforschteneinoedeneinestotalausdermodegekommenenauslae

nauslaeufersdeswestlichenspirlarmsdergalaxisleuchtetunbeachteteinek
ufersdeswestlichenspirlarmsdergalaxisleuchtetunbeachteteinekleinege

leinegelbesonne ...
lbesonneumsiekr ...
```

Man bezeichnet das Verhältnis von doppelten Buchstaben zu insgesamt vorhandenen Buchstaben in dem unrsprünglichen und dem um n Buchstaben verschobenen Text als κ_n. Bei dem verwendeten Textausschnitt ist also

$$\kappa_7 = \frac{12}{142} \approx 0,084\,507$$

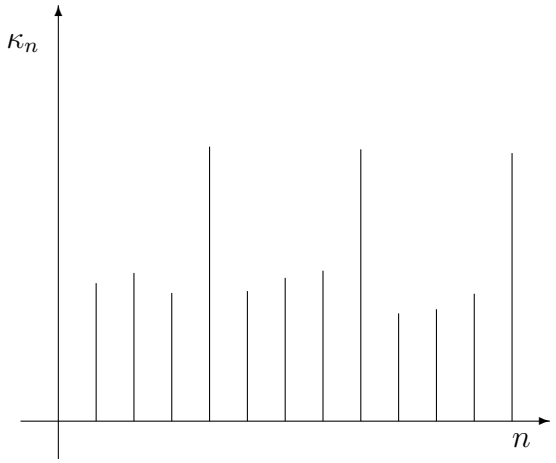
[18] Kasiski, Friedrich W. (1805–1881) – Major des preussischen Heeres
[19] zitiert aus [Ada.Anha], S. 7

Bei Texten aus der selben Sprache haben diese Kappa-Werte (bei längeren Texten) ungefähr den selben Wert, im Deutschen gilt $\kappa_{\text{Deutsch}} \approx 0,0762$, im Englischen $\kappa_{\text{Englisch}} \approx 0,0667$. Dieser charakteristische Wert wird als *Koinzidenzindex* der jeweiligen Sprache bezeichnet.

Der Wert hängt mit den statistischen Eigenheiten der entsprechenden Sprache zusammen. Bei komplett zufälligen Texten, das heißt, bei Texten, in denen jeder Buchstabe an jeder Position mit der Wahrscheinlichkeit $\frac{1}{26}$ erscheint, ist $\kappa_{\text{Zufall}} = \frac{1}{26} \approx 0,0387$. Man kann also anhand des Kappa eines Textes unterscheiden, ob es sich um einen deutschen, englischen oder einen zufälligen Text handelt.

Wir betrachten nun einen mithilfe des Vigenère-Verfahrens verschlüsselten, 3722 Zeichen langen Text, der hier aus Platzgründen nicht angeführt ist.[20] Eine Tabelle der verschiedenen Kappas sieht wie folgt aus:

n	Länge	Treffer	κ_n
1	3721	136	0,0365
2	3720	146	0,0392
3	3719	126	0,0339
4	**3718**	**270**	**0,0726**
5	3717	128	0,0344
6	3716	141	0,0379
7	3715	148	0,0398
8	**3714**	**267**	**0,0719**
9	3713	106	0,0285
10	3712	110	0,0296
11	3711	125	0,0337
12	**3710**	**263**	**0,0709**



Es fällt auf, dass die fett gekennzeichneten Kappa-Werte ungefähr dem Koinzidenzindex eines deutschen Textes (0,0762) entsprechen; die anderen Werte pendeln um den Koinzidenzindex eines zufälligen Textes (0,0387).

Wenn das Kappa eines Textes, der um ein Vielfaches der Schlüssellänge verschoben wurde, berechnet wird, so stimmt dieses Kappa ungefähr mit dem des Klartextes überein. Dies hängt damit zusammen, dass zwei gleiche Buchstaben, mit dem gleichen Schlüsselbuchstaben verschlüsselt, wieder identisch sind. Deshalb liegt der Wert für das Kappa jedes der Textausschnitte, die mit dem gleichen Buchstaben verschlüsselt wurde, nahe bei dem Kappa-Wert des Klartextes. Aus

[20] Der Text ist in Anhang A auf S. 14 zitiert.

der Berechnungsvorschrift des Kappa ergibt sich, dass der Gesamtwert für Kappa gleich dem Durchschnittswert der Teilkappas ist, also immer noch nahe beim Klartextkappa.

Wird der Text hingegen nicht um ein Vielfaches der Schlüssellänge verschoben, so vermindert sich die Wahrscheinlichkeit, dass die verschlüsselten Buchstaben übereinstimmen, erheblich, da sie wahrscheinlich mit verschiedenen Schlüsselbuchstaben verschlüsselt wurden.

Bei obiger Tabelle ist das Kappa gerade bei Verschiebungen um ein Vielfaches von 4 nahe dem Koinzidenzindex für die deutsche Sprache; dies legt eine Schlüssellänge von 4 nahe. Der Schlüssel lässt sich, wie oben beschrieben, durch Zerlegung in Caesar-Texte finden, das Kappa hilft dabei sogar bei der Identifikation der benutzten Sprache – es sollte ungefähr mit ihrem Koinzidenzindex übereinstimmen. Tatsächlich wurde der Text mit dem Schlüssel „`GELB`“ der Länge 4 verschlüsselt.

Der verwendete Text war sehr lang; dadurch wurden die statistischen Eigenheiten sehr ausgeprägt und leichter identifizierbar. Die Kappa-Methode ist jedoch auch bei kurzen Texten einfach anwendbar und sehr effizient; in Kombination mit anderen Verfahren ermöglicht sie eine sehr schnelle Entschlüsselung von Vigenère-verschlüsselten Texten.

4 Schlussfolgerung

Nicht alle verschlüsselten Daten sind so sicher, wie wir es von ihnen vermuten; unentzifferbar wirkendes Buchstabengewirr kann schon mit relativ einfach wirkenden Methoden ohne Kenntnis des Schlüssels entschlüsselt werden. Die vorgestellten Prinzipien basieren auf Grundkenntnissen der Faktorisierung und Teilbarkeit (Zahlentheorie), Häufigkeiten (Statistik), bloßem Ausprobieren aller Möglichkeiten und raffiniertem Beschränken der verfügbaren Möglichkeiten (Kombinatorik), sowie teilweise auf ein bisschen Rechnen (Arithmetik). Fortgeschrittene Verfahren beziehen auch andere, nicht im Schulunterricht behandelte Bereiche der Mathematik ein, zum grundlegenen Verständnis des kryptoanalytischen Prozesses und zur Verwendung einfacher Dechiffriermethoden reicht jedoch die Schulmathematik vollkommen. Hervorzuheben ist insbesondere die Bedeutung der von Nicht-Mathematikern häufig unterschätzten Gebiete der Zahlentheorie und Kombinatorik; sie tragen meist wesentlich zum Gelingen der Methoden bei.

Grundsätzlich ist es möglich, jeden Code zu brechen, durch unsachgemäße Verschüsselung wird es jedoch zu einem Kinderspiel; der zum Brechen aufgewandte Zeitaufwand wird minimal im Vergleich zum verursachten Schaden. Mit dieser Facharbeit verbindet sich die Hoffnung, dass der Leser nach ihrer Lektüre hierauf besser achten wird als zuvor.

A Text für Abschnitt 3.2.2

Der Text aus Goethes Faust, der auf S. 12 nicht angegeben werden konnte. Es handelt sich um den Prolog im Himmel mit Regieanweisungen und Titel ([Goe.Faust]) verschlüsselt mit dem Schlüssel „GELB“:

VVZMUKTNNMXNKPOFXLPSXHTFNMXNRMDDNIYIKICTILLSKRYBILSFXQPQNMDUUTSFRIDEOIOSKMPSF
IYHKPESKXPOBSCSGTSBKPOJKWZOTIEOZRLDNEWUKVHFOWPJTFCVJICTVLCFTAPUZKPTGRRVTHTIXIGPX
KPTILCJKFYFXITTKZZMRIYEKXDJKQTUJSYOKVRBTKTIXEYCRMNLMMMUJIYFTKPMTWESQIHFTRVFORPSY
MPFXKCOJIYNGKOJKYYCKKCFOJWJILSPNIYXKVVFYMYENICSRMNICMPBSICTZIYUGKRBHVTFRYYEYGSOK
PWVTHFOHIRSKMQMOGSTILYFRPPEXISUYMNIAQSFXHPSKVOFVVLDNXPTCINIYIWUVECBJMPTKWSFRPPNO
XEJKJPSYGSBAICWUPWFXRLDNXPTYGSVSXOBYQPFXMYCXITUKRQMYWPOGQEJKJPOMVFOJHPSLIWTKRLVL
YYELIWTARONKICXOVOGUVEHKVTTYIYJSIHJMWNITIWMKQDQNVPOREFGSMNIGIWVTHDUXQPCXEFTKRFNJ
MPXKXEFBSXNKICBAJDMGROWUQWBTHLVLWXFKVFOJFTMJIYXZIYEKMYFQIEUKHPSZMPGYXPOCMCLARRSO
RRTAQSFXHLGREXNZITOHPTUFIYEKWGFXLPFXIYEKQAGGHPWUVOFYHZOTICTILWBMWOPILOFORPCUXPON
ICSBICFNVPOJEDTGRQUKALOJIWOJITOKWEBMWKVJVPJJICBTFWJIORJHXOFTIYHKPYTZVVFJEVFORPSJ
MNIKVRSTHPOSERVTHLMRIOFORPIULPOCICLKWTOJLPSXPTDNATFGQPSYXPOZERNKTSJYXZQNIWFYHLEA
SSFXVOJILPJTQLMCMPEKVYBNWEVTHQSGKDUCMPBRPPTYMNIHITVTWMFLMYEKYYEJYXJILDPTWEHKASOR
MNIMICOKWLIYXDPYMPIYXOVSMNIGYNIAREFXHPNMIDJTHPWKVKFOLTDNOLOTRTDNXSPNIHPXXPNGGSFT
YYECIYOSMNIGYNIJICHGRKFQVPJYZPSNLYUSITOVEEIUWMSILEFJMNIMIHJFYXMGGSFTLEUYXOVJMCOO
GSUJEDMGGSFTEMHKASOZZZOYSYOAROXKPEFTAPJOGSOOGSUYDFTGKPOOGSTKLPOAVHJKWTDNHTFSIYTI
LPOVPLHKROFXOWFORPHUXEEKVHFRXMMKMMUYXPUYZZOMPPJILPNYGSMGKFOJMDUYSHVTHPSRMNIGPDXO
ILNKVDUKREBMITOCIYJMFPTYICXXHPSRIMFTLEUYXOVOLXOOGSUJIYTILPJTHPTNMXNKPDMOGSUYKPHK
FPOKVYFTRETBICOARQUAROCXEFDNXDBRPPJTRFSZMPSOWNIKVLMYNPEKWEJKVKVYITOKVDDNITOZQTSS
MEWKVWBAFGPTIFFXKYBJIYXOIPJTIOFXPLOMFPJTMRFTDTLGHPOJMPJSQPSLPTFMXFOJJWJKKPOJWASO
RRUAROHRITDNMXHXEDJNVLMZIDMOIODNIYTORRUAROMMICOAVYPILTNSICJTHPNMVLTKMYKKHPOWYLSQ
FPHXFEFXWPJTIYBYIOFXLPSXLLTZHFNOVHFOXPSTMNIZWKVYERFTOZNSWEEARFSOQXFXEYAAOWBMIYJY
XLVLHPSKVOFKATHJMCOOGSUYVPDNXXFVLTTZSAIKPPTTITONICSOGSGOROFYHZSZATFOQXFXLPSFPTDN
WNIRINIZHTFSIYTILPOJEFFXRXJILTOOLCFTNLNSICUGKPOOGSNGKDPMECEOILSSIYTKPMTZRTDNXAMG
KPOJICIKVCLKRYTZHFEKRQBAWENKTSJYXZQNIWFYHPOJSVUUVOFXLPSXQPJTIYLTINIZQPQNMDUUTSFR
IDGXALIXICEOIYUKYNIGYQCKWZOJVPXKMDFTMNIZMCEOWNIOWEEKWEPXIYUXEYLTSNIYTPJYITITXCFO
FEEOIRSARRJTHTFLICOKICJYXDJILDFORPSZSWMNITUNEWCHIHVZZZNNMXNKPQPXHPSZICEOIDDNRDUK
RDUKVYFAROWUROFXICEKNPEKLNIYXPMAWEVTHLMRIYIAROBRPPGKVYFHIQSOIOJMXYJILEEOIEJKJMFC
IRUKFCVYXOFXLPSXAPOTICNOVLVILYVXZPSCSCSKROJKRETUAPSJMNIOLYCGPOJTHTFQPLSNITULLCFT
APJJSNIJICHXXYFXAPOTHLTHYXDNIYHXREEGWMMZYYELVFDNXOJKOYGZKPOPESSKDTFXIYNKTSJYXZQN

IWFYALTCIEUKXTIXHPOYSWMZMSSTSNIBICMOICFTAPOTMSSSMCEOIPSREFCTMDHKFEJNRXFORPTZVLFY
ENIZDFGNVPOJICIKVCTUPLOMICBAJOFXICEKPPCZWZMGRRFYITEOVDOOGSUBICCUXPOKWTSXXOFXQPOY
GSTUPLOMICTZVPCZQPQNMDUUTSFRIDEGHLOQMNIKYNIJIYOSMEEKREPZIYIGFTDNQTDNRTFSEWTMICOH
IQBTKPOGQXFOWEFTPTFHMNISMCEOIGPRPPOLVTTILPOCEYHKRQSKMYFSPPJILYBSFTOOGSOOGSUFYSBA
WXJXKPIZIDXOIOFXOLUFIXJZHPSSEFTJICIKVCOARRVZIDTKMOJXFPSREDTKRKJKLOJKWPOMITTZZZOY
ITOKQFSWYPMREMVTHQIXMSOQEYOYXOVOLYFXJLTYIYBAJOFORPNCIRFSMEIKVLCAROTZISCKWNISXHFT
ROVHIVFTRPOSYEFORRVZICNKRDDNMYTKMYFSHFOQPPOJVLOMITTZWTDNHPTXINIZIYXKKPTCSSMHIHVZ
QPQNMDUUTSFRIDTILZOMYEOAVOBAICUKWYJILEMGRRFSMCJYXQSSITOKAPUZIRBXRTDNXMBTKPXKRYJI
LKVSITOKQKXKGVHKPLOMIPSREFCZMSSSMCUXMFNVLLVYZZMRICCXYDUYXLVHWZMRICGXIDTKRFOJQTUR
YDUCMPNKMYFSYSNKHTFHICISXPTILWBTKPEKVSFXVOVJECGYXLVILOBTYCGXITFXWNIKMYFTMNINEMFJ
ITOKWRMKMNIKRYJKKPIGXGPTEWMKRRFOWEFXROJKZPSTITOKRTTZQTSJICTILLMQEXXKRTHYXPOFYCMG
WEEKWXFTWNIKREUOKVFOXVBTRLMRDFMKMNIZICTILWBLJPOKVWJKFETOGSCGPOEOIFOHIOJTKEFXYSEX
YXHKFTDNKPSTMSNJIYHKWPMRIYAAHPSXITAZYYECMCLZYYESYLMYXPVLIWTILLGLIYEUGSJNVOJKINIZ
IYHZXPSYLYFKVQSKYEFAGSEKVWFHIYEOKCFOGSFTWNITIOBYAPSJIYEKHLTKATHCMCLZYYERIMUAQQBY
WPVILXJZHPSRMPCKLZMJIYTILCBTOPOAROXGWTOYGSXGRVFTHPSKVDDNITOARRTILHFHXMFLIDUOKENO
XOBAICOJIYHKHLOQIYEKVSJSQPMYGSMOIEEOIPSFIYHKPGFXXPJRIYTOGSNKTSJYXZQNIWFYEWMKMYWU
RKFOXKVFITUYISJILOFTEWUKRRFXRFOJLEFSMNISMEJNQKVHVPDNIYFYMDUMECIHWNIBSYFORPNMVZFT
LPSXRDPSIYTILWJILXJZHPNZIFGKPDFRFDUFYDQXINIKR

3722 Zeichen

B Häufigkeitstabellen

Bei allen hier angeführten Häufigkeiten ist zu beachten, dass sich die Häufigkeiten je nach Autor und Erhebungsquelle unterscheiden.[21]

Im Deutschen ist bei den Häufigkeitsverteilungen zu beachten, dass, kryptographischen Gewohnheiten folgend, Umlaute wie in Kreuzworträtseln umschrieben werden: `ä = ae`, `ö = oe`, `ü = ue`, sowie `ß = ss`.

Bei den Cliquen ist die erste Möglichkeit sehr grob, die dritte sehr fein; die zweite eignet sich für normale Texte.

B.1 Buchstabenhäufigkeiten

nach [Bau.Entz], S. 304

[21] vgl. auch [Bau.Entz], S. 297 f.

Buchstabe	Häufigkeit		Buchstabe	Häufigkeit	
	Deutsch	Englisch		Deutsch	Englisch
a	6,47 %	8,03 %	n	9,84 %	7,09 %
b	1,93 %	1,54 %	o	2,98 %	7,60 %
c	1,68 %	3,06 %	p	0,96 %	2,00 %
d	4,83 %	3,99 %	q	0,02 %	0,11 %
e	17,48 %	12,51 %	r	7,54 %	6,12 %
f	1,65 %	2,30 %	s	6,83 %	6,54 %
g	3,06 %	1,96 %	t	6,13 %	9,25 %
h	4,23 %	5,49 %	u	4,17 %	2,71 %
i	7,73 %	7,26 %	v	0,94 %	0,99 %
j	0,27 %	0,16 %	w	1,48 %	1,92 %
k	1,46 %	0,67 %	x	0,04 %	0,19 %
l	3,49 %	4,14 %	y	0,08 %	1,73 %
m	2,58 %	2,53 %	z	1,14 %	0,09 %

B.2 Bigrammhäufigkeiten

B.2.1 Deutsch

Die 18 häufigsten Bigramme im Deutschen[22]:

Bigramm	Häufigkeit	Bigramm	Häufigkeit
er	409 %%	en	400 %%
ch	242 %%	de	227 %%
ei	193 %%	nd	187 %%
te	185 %%	in	168 %%
ie	163 %%	ge	147 %%
es	140 %%	ne	122 %%
un	119 %%	st	116 %%
re	112 %%	an	102 %%
he	102 %%	be	101 %%

B.2.2 Englisch

Die 19 häufigsten Bigramme im Englischen[23]:

[22] nach [Bau.Entz], S. 308
[23] nach [Bau.Entz], S. 309

Bigramm	Häufigkeit	Bigramm	Häufigkeit
th	315 %%	st	121 %%
he	251 %%	en	120 %%
an	172 %%	nd	118 %%
in	169 %%	or	113 %%
er	154 %%	to	111 %%
re	148 %%	nt	110 %%
es	145 %%	ed	107 %%
on	145 %%	is	106 %%
ti	128 %%	ar	101 %%
at	124 %%		

B.3 Cliquen

B.3.1 Deutsch

nach André Lange und E.-A. Soudart, 1935[24]

{e}{nirsatdhu}{lgocmbfwkz}{pvjyxq}

{e}{n}{irsat}{dhu}{lgocm}{bfwkz}{pv}{jyxq}

{e}{n}{ir}{sat}{dhu}{lgo}{cm}{bfwkz}{pv}{jyxq}

B.3.2 Englisch

nach L. D. Smith, 1943[25]

{etaoin}{srh}{ld}{cumfpgwyb}{vk}{xjqz}

{e}{t}{aoin}{srh}{ld}{cumf}{pgwyb}{vk}{xjqz}

{e}{t}{ao}{in}{srh}{ld}{cu}{mf}{p}{gwy}{b}{v}{k}{xjqz}

B.4 Koinzidenzindices

Deutsch:	κ_{Deutsch}			$\approx$	$0,0762$
Englisch:	κ_{Englisch}			$\approx$	$0,0667$
Zufall:	κ_{Zufall}	$=$	$\frac{1}{26}$	$\approx$	$0,0387$

C Quellcode

```
/* Dies ist die Datei "krytools.h". Sie stellt in der Facharbeit des Autors
 * besprochene kryptoanalytische Methoden zur Verf"ugung.
```

[24] zitiert nach [Bau.Entz], S. 302
[25] zitiert nach [Bau.Entz], S. 302

```
 * \copy Jan Olligs, 2003
 */

#ifndef __KRYTOOLS_H__
#define __KRYTOOLS_H__

// Hilfsfunktionen

char *condense(char *inString) {
   int newLen = 0;
   char *tempPtr, *irishPtr, *outVar;
   for (tempPtr = inString; *tempPtr; ++tempPtr)
      if (('a' <= *tempPtr && *tempPtr <= 'z') || ('A' <= *tempPtr && *tempPtr <= 'Z'))
         ++newLen;
   outVar = new char[newLen + 1];
   for (tempPtr = inString, irishPtr = outVar; *tempPtr; ++tempPtr)
      if (('a' <= *tempPtr && *tempPtr <= 'z') || ('A' <= *tempPtr && *tempPtr <= 'Z')) {
         *irishPtr = *tempPtr;
         ++irishPtr;
      }
   outVar[newLen] = 0;
   return outVar;
}

void allLowerCase(char *inString) {
   char *tempPtr;
   for (tempPtr = inString; *tempPtr; ++tempPtr)
      if ('A' <= *tempPtr && *tempPtr <= 'Z')
         *tempPtr = *tempPtr + 'a' - 'A';
}

void allUpperCase(char *inString) {
   char *tempPtr;
   for (tempPtr = inString; *tempPtr; ++tempPtr)
      if ('a' <= *tempPtr && *tempPtr <= 'z')
         *tempPtr = *tempPtr + 'A' - 'a';
}

// Chiffren

char oneLetterCaesar(char inChar, int key) {
   if ('a' <= inChar && inChar <= 'z')
      inChar -= 'a';
   else if ('A' <= inChar && inChar <= 'Z')
      inChar -= 'A';
   else
      return inChar;
   inChar += key;
   if (inChar < 0)
      inChar += 26;
   inChar %= 26;
   return 'A' + inChar;
}

char *Caesar(char *inString, int key) {
   int length;
   for (length = 0; inString[length]; ++length);
   char *tempPtr, *outPtr, *outString = new char[length + 1];
   outString[length] = 0;
   for (tempPtr = inString, outPtr = outString; *tempPtr; ++tempPtr, ++outPtr)
      *outPtr = oneLetterCaesar(*tempPtr, key);
   return outString;
}

char *Vigenere(char *inString, char *key, bool encrypt) {
   int keyLength, stringLength, i = 0;
   for (keyLength = 0; key[keyLength]; ++keyLength) {
      if ('a' <= key[keyLength] && key[keyLength] <= 'z')
         key[keyLength] -= 'a';
      else if ('A' <= key[keyLength] && key[keyLength] <= 'Z')
         key[keyLength] -= 'A';
      else
         key[keyLength] = 0;
      if (!encrypt)
         key[keyLength] *= -1;
   }
   for (stringLength = 0; inString[stringLength]; ++stringLength);
```

```
   char *tempPtr, *outPtr, *outString = new char[stringLength + 1];
   outString[stringLength] = 0;
   outPtr = outString;
   for (tempPtr = inString; *tempPtr; ++tempPtr) {
      *outPtr = oneLetterCaesar(*tempPtr, key[i]);
      ++i %= keyLength;
      ++outPtr;
   }
   return outString;
}

// Konstanten

const double GermanIOC  = 0.0762;
const double EnglishIOC = 0.0667;

const double GermanFreqs[]  = {0.0647, 0.0193, 0.0168, 0.0483, 0.1748, 0.0165,
                               0.0306, 0.0423, 0.0773, 0.0027, 0.0146, 0.0349,
                               0.0258, 0.0984, 0.0298, 0.0096, 0.0002, 0.0754,
                               0.0683, 0.0613, 0.0417, 0.0094, 0.0148, 0.0004,
                               0.0008, 0.0114};
const double EnglishFreqs[] = {0.0803, 0.0154, 0.0306, 0.0399, 0.1251, 0.0230,
                               0.0196, 0.0549, 0.0726, 0.0016, 0.0067, 0.0414,
                               0.0253, 0.0709, 0.0760, 0.0200, 0.0011, 0.0612,
                               0.0654, 0.0925, 0.0271, 0.0099, 0.0192, 0.0019,
                               0.0173, 0.0009};

const int  numGermanCliques       = 8;
const char GermanLetterSequence[] = {'e', 'n', 'i', 'r', 's', 'a', 't', 'd', 'h',
                                     'u', 'l', 'g', 'o', 'c', 'm', 'b', 'f', 'w',
                                     'k', 'z', 'p', 'v', 'j', 'y', 'x', 'q'};
const int  GermanCliques[]        = {1, 1, 5, 3, 5, 5, 2, 4};


const int  numEnglishCliques       = 9;
const char EnglishLetterSequence[] = {'e', 't', 'a', 'o', 'i', 'n', 's', 'r', 'h',
                                      'l', 'd', 'c', 'u', 'm', 'f', 'p', 'g', 'w',
                                      'y', 'b', 'v', 'k', 'x', 'j', 'q', 'z'};
const int  EnglishCliques[]        = {1, 1, 4, 3, 2, 4, 5, 2, 4};

// Entschl"usselungshilfen

void getFreqs(double freqArray[26], char *inString) {
   double total = 0;
   int i;
   for (i = 0; i < 26; ++i)
      freqArray[i] = 0;
   for (; *inString; ++inString) {
      if ('a' <= *inString && *inString <= 'z') {
          ++freqArray[*inString - 'a'];
          ++total;
      }
      else if ('A' <= *inString && *inString <= 'Z') {
          ++freqArray[*inString - 'A'];
          ++total;
      }
   }
   for (i = 0; i < 26; ++i)
      freqArray[i] /= total;
}

void getOffsetFreqs(double freqArray[26], char *inString, int offset, int gap) {
   double total = 0;
   int i;
   offset %= gap;
   for (i = 0; i < 26; ++i)
      freqArray[i] = 0;
   for (i = 0; *inString; ++inString) {
      i %= gap;
      if (i == offset) {
         if ('a' <= *inString && *inString <= 'z') {
            ++freqArray[*inString - 'a'];
            ++total;
         }
         else if ('A' <= *inString && *inString <= 'Z') {
            ++freqArray[*inString - 'A'];
            ++total;
```

```
         }
      }
      ++i;
   }
   for (i = 0; i < 26; ++i)
      freqArray[i] /= total;
}

/* Es gilt: d = sigma (c - k)^2 = sigma c^2 - 2sigma ck + sigma k^2
 *            = const. -2sigma ck
 * d wird also am kleinsten, wenn sigma ck am gr"o"sten ist
 */

double deviation(const double realAlphabet[26], const double textAlphabet[26]) {
   double diff = 0;
   for (int i = 0; i < 26; ++i)
      diff += realAlphabet[i] * textAlphabet[i];
   return diff;
}

double kappa(char *inString, int offset) {
   double length = 0, match = 0;
   char *ptr1, *ptr2;
   ptr1 = inString;
   ptr2 = inString + offset;
   for (; *ptr2; ++ptr2) {
      if (*ptr1 == *ptr2)
         ++match;
      ++ptr1;
      ++length;
   }
   if (length > 0)
      return (match / length);
   else
      return 1;
}

double chi(const double *langFreqs) {
   double temp = 0;
   int i;
   for (i = 0; i < 26; ++i)
      temp += langFreqs[i] * langFreqs[i];
   return temp;
}

// Entschl"usselungsfunktionen

char CaesarKey(char *inString, const double langFreq[26]) {
   int length, i, j, currKey = 0;
   for (length = 0; inString[length]; ++length);
   char *tempString = new char[length + 1];
   for (i = 0; i <= length; ++i)
      tempString = inString;
   allUpperCase(tempString);
   double freqs[26];
   getFreqs(freqs, tempString);
   double currDev = deviation(langFreq, freqs), temp;
   for (i = -1; i > -26; --i) {
      temp = freqs[25];
      for (j = 25; j > -1; --j)
         freqs[j] = freqs[j - 1];
      freqs[0] = freqs[25];
      temp = deviation(langFreq, freqs);
      if (temp > currDev) {
         currDev = temp;
         currKey = i;
      }
   }
   delete [] tempString;
   return ((currKey * (-1)) + 'A');
}

char *VigenereKey(char *inString, const double langFreq[26], const int maxKeyLength, const double acceptance) {
   int textLength, keyLength = -1, i, j, k;
   for (textLength = 0; inString[textLength]; ++textLength);
   char *tempString = new char[textLength + 1], *key;
   for (i = 0; i <= textLength; ++i)
```

```
      tempString = inString;
   allUpperCase(tempString);
   double *kappas = new double[maxKeyLength];
   double minIOC = chi(langFreq) * acceptance, temp;
   for (i = 0; i < maxKeyLength; ++i)
      kappas[i] = kappa(tempString, i + 1);
   for (i = 0; i < maxKeyLength; ++i) {
      temp = 0.0;
      k = 0;
      for (j = i; j < maxKeyLength; j += i + 1) {
         temp += kappas[j];
         ++k;
      }
      temp /= k;
      if (temp >= minIOC) {
         keyLength = i + 1;
         break;
      }
   }
   delete [] kappas;
   if (keyLength == -1) {
      delete [] tempString;
      return NULL;
   }
   key = new char[keyLength + 1];
   key[keyLength] = 0;
   for (k = 0; k < keyLength; ++k) {
      double freqs[26];
      int currKey = 0;
      getOffsetFreqs(freqs, tempString, k, keyLength);
      double currDev = deviation(langFreq, freqs), temp;
      for (i = 1; i < 26; ++i) {
         temp = freqs[25];
         for (j = 25; j > -1; --j)
            freqs[j] = freqs[j - 1];
         freqs[0] = freqs[25];
         temp = deviation(langFreq, freqs);
         if (temp > currDev) {
            currDev = temp;
            currKey = i;
         }
      }
      key[k] = 25 - currKey + 'A';
   }
   delete [] tempString;
   return key;
}

#endif
```

D Selbstständigkeitserklärung

Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Krefeld, den 26. März 2003

Literatur

[Ada.Anha] Adams, D.: **Per Anhalter durch die Galaxis**. 6. Aufl. München: Heyne. 1999. ISBN: 3 453 14697 2

[Bau.Entz] Bauer, F. L.: **Entzifferte Geheimnisse** – *Methoden und Maximen der Kryptologie*. 3., überarb. u. erw. Aufl. Berlin/Heidelberg: Springer. 2000. ISBN: 3 540 67931 6

[Beu.Krypt] Beutelspacher, A.: **Kryptologie** – *Eine Einführung in die Wissenschaft vom Verschlüsseln, Verbergen und Verheimlichen*. 6., überarb. Aufl. Braunschweig/Wiesbaden: Vieweg. 2002. ISBN: 3 528 58990 6

[Buc.Krypt] Buchmann, J.: **Einführung in die Kryptographie**. 2., erw. Aufl. Berlin/Heidelberg: Springer. 2001. ISBN: 3 540 41283 2

[Goe.Faust] Goethe, J. W. von: **Faust** – *Der Tragödie erster Teil* – Prolog im Himmel. `<http://www.gutenberg2002.de/goethe/faust1/faust004.htm>` – *Stand 15.03.2003*

[Kip.Versc] Kippenhahn, R.: **Verschlüsselte Botschaften** – *Geheimschriften, Enigma und Chipkarte*. 2., Aufl. Reinbek: Rowohlt. 2001. ISBN: 3 499 60807 3

[Lei.Entz] Leiberich, O.: **Entzifferung von Geheimtexten** in: *Spektrum der Wissenschaft – Dossier – Kryptographie*. S. 19–31. Heidelberg: Spektrum. 2001. ISSN: 0947 7349

[New.Encyc] Newton, D. E.: **Encyclopedia of Cryptology**. Oxford: ABC-CLIO. 1998. ISBN: 1 85109 323 0

[Pad.Elem] Padberg, F.: **Elementare Zahlentheorie**. 2., überarb. u. erw. Aufl. Heidelberg/Berlin: Spektrum. 2001. ISBN: 3 86025 453 7

[Sin.Code] Singh, S.: **The Code Book** – *The Secret History of Codes & Code-breaking*. London: Fourth Estate. 2000. ISBN: 1 85702 889 9

[Wel.Codes] Welsh, D.: **Codes and Cryptography**. Oxford: Oxford University Press. 2000. ISBN: 0 19 853287 3

[Wob.Aben] Wobst, R.: **Abenteuer Kryptographie** – *Methode, Risiken und Nutzen der Datenverschlüsselung*. 3. Aufl. München: Addison-Wesley. 2001. ISBN: 3 8273 1815 7