BSZ f. Technik, Wirtschaft, Gesundheit
„Erdmann Kircheis“ Oelsnitz/Erzgebirge
Berufliches Gymnasium
Fachrichtung Informations- und Kommunikationstechnologie

# Belegarbeit

Schuljahr 2013/2014

Thema: **Mathematik der Kryptologie**

Fach: Mathematik

Fachlehrerin: Frau Müller

vorgelegt von: Richard Siegel

Kurs/Hj. 13 - 1

Abgabedatum: 02.12.2013

# Inhaltsverzeichnis

Seite

# 1 Einleitung

In Zeiten von Datenhandel im Internet, in Zeiten von Onlinebanking und digitalen Warenhäusern, in Zeiten der allgemeinen Dauerüberwachung durch Regierungen und Geheimdienste der Erde; kurzum zu Beginn des einundzwanzigsten Jahrhunderts, stellt sich der Verfasser die Frage: „Kann man die moderne Technik überhaupt noch verwenden, um vertrauliche Inhalte zu besprechen?“
Das Ziel dieser Arbeit ist das Ermitteln (und Verstehen) einer sicheren, funktionellen, mathematischen Methode zur Geheimhaltung eines zu übermittelnden Textes.

# 2 Von der Kommunikation bis zur Kryptographie

## 2.1 Erste Meilensteine der Kommunikation

Die ersten gesprochenen Worte, wie simpel sie auch gewesen sein mögen, waren der erste große Meilenstein in der Entwicklung der menschlichen Kommunikation. Doch durch das Sprechen allein war es weder möglich, Informationen noch die Worte der Sprachen selbst über eine längere Zeit zu bewahren.
Der zweite große Meilenstein entstand also aus dem Bedürfnis der Menschen heraus, Informationen zu bewahren. Man entwickelte die ersten Schriftsysteme (siehe Anlange 5), welche sich nicht nur zur Bewahrung sondern auch zur Übertragung von Informationen eigneten. Daraufhin ergab sich, durch Konflikte zwischen den Völkern, die Notwendigkeit Informationsübertragungen für Außenstehende unverständlich zu gestalten. Um das Entschlüsseln geschriebener Geheimbotschaften zu erschweren, entschied man sich anfangs zur Verwendung verschiedener, dem Angreifer unbekannter Schriftsysteme, der Geheimschriften (siehe Anlange 7). Bei den meisten dieser Geheimschriften handelt es sich um einfache Ersatzalphabete welche entsprechend der Komplexität des Ursprungsalphabetes zu entschlüsseln sind (siehe 3.3).

## 2.2 Ursprung der Kryptosysteme

Schon durch feindliche Gelehrte, aber auch durch die aufstrebende Gesellschaft und der Entwicklung hin zu immer mehr Lese- und Schreibkenntnissen der breiten Masse, wurden Geheimschriften immer unsicherer, da sie ohnehin mit relativ geringem Zeitaufwand dechiffriert werden konnten. Aus dieser Notwendigkeit heraus kam es zur Entwicklung von Kryptosystemen.

Kryptosysteme wenden zur Verschlüsselung eines Textes (Klartext) Kodierungsmethoden an, welche durch den Einsatz verschieden aufwendiger mathematischer Algorithmen [1, Seite 12] möglich werden. Um die Kodierung als solches zu ermöglichen, muss das zu verschlüsselnde Alphabet einer logischen Ordnung (Ord) eindeutig zugeteilt sein. Eine derartige Zuordnung ist z.B. der ASCII-Code (siehe Anlage 1), welcher üblicherweise bei der Speicherung bzw. Umrechnung von Zeichen in Computerprogrammen angewendet wird. Jedes Zeichen entspricht also genau einem eindeutigen Zahlenwert.

### 2.3 Heutige Ziele der Verschlüsselung

Das Anwenden von mathematischen Algorithmen zur Kodierung eines Textes kann verschiedene Gründe haben. Unter Verwendung der richtigen Verfahrensweisen ist es einerseits möglich den logischen Lösungsweg zeitlich so zu gestalten, dass es einem Angreifer kaum möglich ist, eine Dekodierung abzuschließen (bevor die Informationen für diesen an Wert verloren haben). Andererseits ermöglicht man eingeweihten Empfängern eine vollfunktionelle Art des Entschlüsselns, ohne diesen zwangsläufig die Möglichkeit zu geben, selbst Informationen auf dem gleichen Weg verschlüsseln zu können.
Heutzutage benötigen wir allerdings selbst für den Betrieb eines jeden Computers funktionierende Kryptosysteme. Auch wenn diese wichtigen Grundlagensysteme der Computertechnik nicht geschaffen wurden um Informationen geheim zu halten, so ist es doch wichtig und unbestreitbar ein kryptologischer Kodierungs- und Dekodierungsprozess, wenn ein geschriebener Text nach algorithmischen Verfahren in die binäre Maschinensprache eines Computers umgesetzt wird und auf die eine oder andere Art danach wieder (dekodiert) abgerufen werden kann.

## 3 Einführung in die Verschlüsselung

### 3.1 Caesar Chiffre

Ein simples Chiffrierungsverfahren, bei dem das Verschlüsselungsalphabet mit dem Alphabet des Originaltextes in der Regel in Zeichenmenge und Art übereinstimmt, ist das Caesar-Verfahren. Obwohl es sich nicht um ein Kodierungsverfahren handelt, ist die Ordnung der beiden Alphabete die wichtigste Grundlage für die Chiffrierung. Daher gilt das Caesar-Verfahren als Grundlage für die Entwicklung komplexer Verschlüsselungsmethoden.
Beide Alphabete Lat. {0,1,... ,25} (siehe Anlage 10) können zur Vereinfachung im Kreis auf zwei übereinanderliegenden Scheiben dargestellt werden (siehe Anlage 4). Durch das

Verschieben der Scheiben lassen sich 25 verschiedene Ersatzalphabete erschaffen, wenn man das entstehende Alphabet bei nicht verschobenen Scheiben (also der Zuordnung A zu A bzw. B zu B usw.) nicht mit zählt. Alternativ lässt sich dieses Prinzip auch mit Hilfe zweier zu einander verschiebbarer Streifen verdeutlichen (siehe Anlage 4). Man sorgt also dafür, dass sowohl der Sender als auch der Empfänger den gemeinsamen Schlüssel (B für A zu B, C für A zu C usw.) erhalten. Während der Sender das neue Alphabet zur Erstellung des Chiffretextes verwendet, nutzt der Empfänger das Klartextalphabet, in gleicher Weise zur Dechiffrierung eines Chiffretextes um den Klartext zu erhalten.
Mathematisch betrachtet benötigt man zur Erstellung eines solchen Ersatzalphabetes jedoch immer einen Zahl-Schlüssel, welcher auf den **Ordnungswert** des jeweiligen Buchstabens des Ausgangsalphabetes aufaddiert wird. Alle Ordnungswerte, welche sich für das Verschlüsselungsalphabet ergeben, werden also direkt in den entsprechenden Buchstaben des Verschlüsselungsalphabetes dargestellt. Von allen scheinbar entstehenden Ordnungswerten, welche die Anzahl der im entsprechenden Alphabet existierenden Zeichen (im Beispiel 26) überschreiten, wird eben diese Anzahl vom Ordnungswert abgezogen. Wenn nötig wiederholt man diesen Prozess bis alle ermittelten Ordnungswerte innerhalb des definierten Spektrums liegen.

### 3.2 Vigenère Chiffre

Durch eine einfache Weiterentwicklung des, monoalphabetischen Caesar-Verfahrens (zu einem polyalphabetischen Verschlüsselungsverfahren) entsteht das Vigenere-Verfahren. Man wählt dazu keinen einfachen Schlüssel wie beispielsweise B oder C mehr, sondern eine Schlüsselkette bzw. ein Schlüsselwort wie z.B. „GEHEIM“. Dieses Schlüsselwort legt Buchstabe für Buchstabe immer neue Schlüssel fest. Das Prinzip der zwei zueinander verschiebbaren Streifen (siehe Anlage 4) wird außerdem aus praktischen Gründen durch das Vigenère-Quadrat (siehe Anlage 3) auf eine Gesamtübersicht der möglichen Verschlüsselungsalphabete ergänzt. Um einen Text mit dem Vigenère-Quadrat zu chiffrieren sucht man im Klartextalphabet beispielsweise nach dem siebenten Buchstaben dieses Klartextes (H) und nach dem entsprechenden siebenden Buchstaben des Schlüsselwortes. Wenn, wie in diesem Beispiel, das Wort nicht lang genug ist, wiederholt man es einfach. An Stelle des siebenten Buchstaben sucht man also nach dem ersten (G). Der Buchstabe, welcher sich gleichermaßen in der gefundenen Zeile und der gefundenen Spalte befindet, ist der entsprechende Buchstabe im Chiffretext. Um zu dechiffrieren sucht man zuerst den Schlüssel und daraufhin in der gefundenen Spalte nach dem, zum Schlüssel passenden, Chiffre-Buchstaben. Der Klartext-Buchstabe befindet sich nun in der gleichen Zeile im Klartext-Alphabet.

Mathematisch betrachtet wird zur Chiffrierung das Schlüsselwort in die entsprechenden Ordnungswerte, welche dann als Schlüssel fungieren, eingeteilt. Lautet der zu verschlüsselnde Text "GANZGEHEIMERTEXT" sollte die Chiffrierung mit Hilfe der Schlüsselkette Lat."GEHEIM" → Ord{6,4,7,4,8,12} wie folgt ablaufen.
Der erste Buchstabe "G" wird nach der oben beschriebenen Caesarmethode mit dem Schlüssel 6 verschlüsselt, für den Folgebuchstaben "A" gilt nun der Schlüssel 4 und weiterhin werden die Buchstaben N, Z, G und E mit den übrigen Schlüsseln 7, 4, 8 und 12 chiffriert. Nach dem Ende des Schlüsselwortes wiederholt man einfach die Werte, welche durch dieses vorgegeben werden, in gleichbleibender Reihenfolge. Man erhält so den chiffrierten Text "MEUDOQNIPQMDZIEX".
Durch das Subtrahieren des jeweiligen Schlüssels vom Chiffrierten Ordnungswert lassen sich die Ursprungstexte für den eingeweihten Empfänger ohne weiteres wieder dechiffrieren.

### 3.3 Dechiffrieren mittels „Brute-Force“

Der Sinn des Verschlüsselns jeder Art von Information ist das Geheimhalten eben dieser. Leider bieten die beiden zuvor näher erläuterten Verschlüsselungsmethoden keine echte Sicherheit vor Kryptoanalytikern, welche die Information abhören (siehe Anlage 9). Es ist nicht einmal in jedem Fall nötig kryptoanalytisch vorzugehen. Caesar-Chiffre-Texte lassen sich am einfachsten durch eine Brute-Force-Attacke [7] entziffern. Das bedeutet, dass ein Computer alle möglichen Schlüssel einfach ausprobiert bis die erhaltene Lösung sinnvolle Worte bzw. Zeichenketten beinhaltet. Derartige Zeichenketten wären z.B. im Deutschen „DER“, „DIE“, „DAS“ oder „SCH“ und im Englischen typischerweise „THE“ oder „TH“ und ähnliches. Auch die Suche nach besonders häufig auftretenden Einzelbuchstaben innerhalb eines Ergebnis-Textes im Vergleich zur Sprache des Klartextes (siehe Anlage 8) bieten eine Grundlage für die systematische Lösungsfindung (durch kryptoanalytisches Vorgehen). Im Deutschen taucht beispielsweise der Buchstabe E sehr häufig auf, wie man auch im Vigenère Beispiel (siehe Anlage 3) gut erkennen kann. Brute-Force-Attacken auf Caesar-Chiffre-Texte sind sehr simpel und lassen sich sogar bequem von Hand ausführen, da man lediglich 25 Schlüssel ausprobieren muss.
Auch die Anwendung des Vigenère-Verfahrens verhindert in der Regel nicht die Lösbarkeit der Verschlüsselung via Brute-Force, erschwert diese allerdings mit zunehmender Länge des Schlüsselwortes. Das ist der Fall, da man, selbst beim eher kleinen Beispiel-Alphabet Lat.{0, 1, …, 25}, auf kombinatorische Weise erkennen kann, dass es für Worte mit einer Länge von (wie im Beispiel) sechs Buchstaben immerhin $26^6$ (= 308915776) Buchstabenkombinationsmöglichkeiten gibt.

Darüberhinaus handelt es sich bei beiden Verschlüsselungsmethoden um sogenannte symmetrische Verschlüsselungsverfahren. Das bedeutet, dass der gleiche Schlüssel zur Ver- und Entschlüsselung verwendet wird. Das in den meisten Fällen notwendige Versenden dieses einen Schlüssels, auf dem gleichen Kanal wie die eigentliche Nachricht, liefert also denn Schlüssel für den Angreifer gleich mit.

# 4 Das RSA-Verfahren

## 4.1 Das asymmetrische Verschlüsselungsverfahren RSA

Die Notwendigkeit einen zur Entschlüsselung geeigneten Schlüssel an den Empfänger (oder Sender) zu senden, zerstört die Möglichkeit, selbst mit sehr guten symmetrischen Verschlüsselungssystemen Daten sicher zu übermitteln. Aus diesem Grund entwickelten Ronald Rivest, Adi Shamir und Leonard Adelman das nach ihnen benannte RSA-Verfahren. [1, Seite 326]
Das RSA-Verfahren ist ein asymmetrisches Verschlüsselungsverfahren (siehe Anlage 2), welches sich den Umstand zu Nutze macht, dass es nach heutigem Kenntnisstand der mathematischen Zahlentheorie als unmöglich gilt (ohne enormen Zeitaufwand, selbst für Computer), zwei große Primzahlen (p und q) nur mittels deren Produkt zu bestimmen.
Die RSA-Schlüsselbildung besteht aus fünf Schritten (siehe Anlage 12, Zeile 39 - 49). Die daraus resultierenden Werte e (wie encode [Dt. kodieren]) und n bilden dabei den öffentlichen Schlüssel, werden also zur Kodierung von Werten dem Sender übermittelt. Der Sender ist dann in der Lage eine Nachricht an den Erzeuger von e, welcher den passenden Schlüssel d (wie decode [Dt. Dekodieren]) besitzt, zu senden. Möchte der Sender eine Antwort von eben diesem Empfänger erhalten, so muss der Sender ebenfalls einen Schlüsselsatz (n,e,d) erstellen, um zu gewährleisten, dass der geheime Dekodierungsschlüssel (d) immer geheim bleibt. Alle asymmetrischen Kryptosystme verfügen über derartig genutzte Schlüssel. Die Geheimhaltung der gewählten Werte p und q sowie des daraus errechneten $\varphi(n)$ und des, ebenfalls errechneten, Private-Key d; sind die wichtigsten Grundlagen für die Aufrechterhaltung der Sicherheit im System.
Für alle festzulegenden Werte gilt, der Zahlenwert muss im Bereich der positiven natürlichen Zahlen bestimmt werden. Korrekte resultierende Schlüssel und Kryptotexte werden daher ebenfalls als positive natürliche Zahlen ausgedrückt.

## 4.2 Mathematische Grundlagen des RSA-Verfahrens

### 4.2.1 Modulare Addition

Das modulare Rechnen ist eine der wichtigsten Grundlagen für die Erschaffung eines asymmetrischen Verschlüsselungssystems. Die Modulare Addition gehört in den Bereich der ganzzahligen Division, der Art von Division also, bei welcher es keine Ergebnisse außerhalb des Bereichs der ganzen Zahlen gibt.

z.B.: $8 : 4 = 2$ oder $6 : 2 = 3$

Treten bei der ganzzahligen Division Brüche auf, welche sich nicht in dieser Art lösen lassen, so ermittelt man das Ergebnis des nächst kleineren möglichen Dividenden durch den gleichbleibenden Divisor und den sogenannten Rest, welcher bei der Rückrechnung (der Multiplikation des Ergebnis mit dem Divisor), aufaddiert werden muss.

z.B.: $9 : 2 = 4 \quad \text{Rest } 1$ bzw. $9 = 4 \cdot 2 + 1$

Das Ermitteln eines derartigen ganzzahligen Bruch-Ergebnisses und des dazugehörigen Restes lässt sich mit Hilfe der Operationen „div“ und „mod“ (für die Modularisierung) verallgemeinert darstellen.

Es gilt: $a = (a \text{ div b}) \cdot b + (a \bmod b)$

z.B.: $9 = (9 \text{ div } 2) \cdot 2 + (9 \bmod 2)$

Wenn die Ergebnisse der Operation „mod“ für einen Teiler deckungsgleich sind, lässt sich dieser Zustand mittels der Kongruenz-Schreibweise darstellen.

z.B. $9 \bmod 2 = 7 \bmod 2 = 1 \quad \rightarrow \quad 9 \equiv 7 \pmod 2$

Allgemein ausgedrückt:

$a \equiv b \pmod x$ gilt, wenn $a \bmod x = b \bmod \text{x}$

### 4.2.2 Größter gemeinsamer Teiler

Der ggT ist eine mathematische Funktion, welche den größtmöglichen Divisor zweier Zahlen für eine ganzzahlige Division mit Rest 0 ermittelt (siehe Anlage 12, Zeile 123).

$\text{ggT}(a, b) = c \quad \rightarrow \quad a \bmod c = b \bmod c = 0$

Der ggT lässt sich durch Primfaktorzerlegung oder mit Hilfe des Algorithmus von Euklid ermitteln. Für die Primfaktorzerlegung gilt: „Der größte gemeinsame Teiler (ggT) zweier oder mehrerer Zahlen ist das Produkt der gemeinsamen Primfaktoren.

z.B.: $\text{ggT}(16{,}24) = 2 \cdot 2 \cdot 2 = 8$ “ [5]

Die Primfaktoren ermittelt man durch das Teilen der Werte a und b durch die je kleinstmögliche Primzahl. Die so gefundene Primzahl ist der erste Primfaktor der jeweiligen Zahl a oder b. Alle weiteren Primfaktoren werden in gleicher Art auf Basis des entstandenen Quotienten an Stelle der Werte a bzw. b gebildet bis der Quotient schließlich 1 lautet.
Die gemeinsamen, so ermittelten, Primfaktoren werden nun lediglich multipliziert (siehe Anlage 11).
Der Algorithmus von Euklid ist etwas effizienter, da man keine Primzahlen zur Teilung ausprobieren muss. Man ermittelt den Rest von a durch b und im nächsten Schritt den Rest von b durch den zuvor ermittelten Rest, bis der Rest 0 ist (siehe Anlage 12, Zeile 122).

$$a \bmod b = r_1 \quad \rightarrow \quad b \bmod (a \bmod b) = r_2$$

$$\rightarrow \quad (a \bmod b) \bmod \big(b \bmod (a \bmod b)\big) = r_3 \quad \rightarrow \quad \ldots$$

Der letzte Rest, welcher ungleich Null ist, entspricht dem ggT der Werte a und b. Bei der schriftlichen Rechnung lässt sich der Euklidische Algorithmus stark vereinfacht darstellen. (siehe Anlage 6)

### 4.2.3 Die Eulersche Funktion φ

„Die Eulersche Phi-Funktion ist eine zahlentheoretische Funktion. Sie ordnet jeder natürlichen Zahl n die Anzahl der natürlichen Zahlen a von 1 bis n zu, die zu n teilerfremd sind (also $\mathrm{ggT}(a, n) = 1$).“ [4]
Da Primzahlen nur durch sich selbst und 1 geteilt werden können, erkennen wir, dass die Eulersche Funktion φ einer jeden Primzahl alternativ durch das Subtrahieren von 1 gebildet werden kann.

$$\varphi(p) = p - 1$$

Außerdem ist die Eulersche Funktion φ multiplikativ, kann also in ihre Bestandteile (mittels Multiplikation) zerlegt werden.

$$\varphi(a) = \varphi\left(\frac{a}{x}\right) \cdot \varphi(x)$$

Das heißt wenn,

$$c = a \cdot b$$

gilt:

$$\varphi(c) = \varphi(a) \cdot \varphi(b)$$

### 4.2.4 Der erweiterte Euklidische Algorithmus

Der Euklidische Algorithmus ist uns bereits durch das Lösen des ggT (siehe 4.2.2) bekannt.
Darüberhinaus wird der Euklidische Algorithmus zur Bestimmung des sogenannten multiplikativen Inversen einer Zahl modular zu einer anderen Zahl nützlich.

Das multiplikative Inverse i einer Zahl x multipliziert mit dieser Zahl hat immer das Ergebnis 1.

$$x = \frac{n}{m} \quad \rightarrow \quad \frac{n}{m} \cdot \frac{m}{n} = 1 = x \cdot i \quad \rightarrow \quad i = \frac{m}{n}$$

Wendet man dieses Prinzip auf eine modulare Kongruenz an, so heißt das:

$$x \cdot i \equiv 1 \pmod{a}$$

Da 1 modular zu a ($a \in \mathbb{N} | a > 0$) immer das Ergebnis 1 hat, können wir ableiten, dass wir nach einem Inversen i (von x) suchen, welches multipliziert mit x modular zu a das Ergebnis 1 hat.

$$x \cdot i \bmod a = 1$$

Diese Zahl i erhält man durch den erweiterten Euklidischen Algorithmus. Man beginnt diese Rechnung ganz einfach mit dem Bilden des ggT von x und a (siehe 4.2.2). Sobald man den Rest 0 bestimmt hat, beginnt man diese Rechnung rückwärts zu ergänzen (siehe Anlage 6).
Man beginnt damit, den ermittelten ggT der Werte x und a durch die anderen errechneten Werte mittels Linearfaktorzerlegung darzustellen.
Zum besseren Verständnis hat der Verfasser eine Übersicht zur Arbeit mit dem Erweiterten Euklidischen Algorithmus (siehe Anlage 6) erstellt, welche die schrittweise Vorgehensweise, bis zum Ergebnis, verdeutlichen soll.

## 4.3 Schlüsselberechnung

### 4.3.1 Schritt 1: Festlegen von p und q

Zunächst müssen für die Bildung funktioneller Schlüsselsätze die Zahlen p und q bestimmt werden.
Beide Zahlen p und q müssen Primzahlen sein und sollten möglichst groß gewählt werden um die Sicherheit der zu errechnenden Werte zu erhöhen.

Für diese Primzahlen gilt: $p \neq q$

### 4.3.2 Schritt 2: Die Berechnung von n

Ein weiterer Nutzen möglichst großer p-q-Werte ist die Vergrößerung der kompatiblen zu verschlüsselnden Werte. Dieser Effekt entsteht, weil n, das Produkt von p und q, nicht vom zu verschlüsselnden Wert überschritten werden darf. (siehe Anlage 12, Zeile 43)

$$n = p \cdot q$$

### 4.3.3 Schritt 3: Die Berechnung von φ(n)

Aufgrund des günstigen Umstandes, dass es sich bei p und q (den Faktoren für n) um Primzahlen handelt, können wir entnehmen, dass die Primzahl-Sonderregel für die Berechnung von φ(n) (siehe 4.2.3) in Kraft tritt. Unter Anwendung dieser Regel lässt sich also auch in der Praxis die Berechnung sehr einfach umsetzen (siehe Anlage 12, Zeile 45)

$$\varphi(n) = (p - 1) \cdot (q - 1)$$

### 4.3.4 Schritt 4: Festlegen von e

Der e-Wert wird, ähnlich wie p und q, zufällig gewählt (siehe Anlage 12, Zeile 111)
Zur Ermittlung eines geeigneten e-Wertes muss die gewählte Zahl zwei notwendige Bedingungen erfüllen.

1. Der gewählte Wert liegt zwischen 1 und φ(n)

$$e \in \mathbb{N} \mid 1 < e < \varphi(n)$$

2. Der ggT von e und φ(n) beträgt 1 (bzw. e ist eine Primzahl).

$$\mathrm{ggT}(e, \varphi(n)) = 1$$

### 4.3.5 Schritt 5: Die Berechnung von d

Das gewünschte mathematische Ziel, dass das Produkt aus e und d kongruent zu 1 Modulo n wird, bildet die Grundlage für die Berechnung von d.

$$e \cdot d \equiv 1 \bmod n \qquad \text{oder} \qquad e \cdot d \bmod n = 1$$

Wir können also schlussfolgern, dass d das Inverse (siehe 4.2.4) von e ist. Die Berechnung des d-Wertes wird also erst mit Hilfe des erweiterten Euklidischen Algorithmus möglich (siehe Anlage 6).
Bei der praktischen Umsetzung, z.B. in Computer-Programmen (siehe Anlage 12, Zeile 185), tritt regelmäßig das Problem auf, dass d eine negative Zahl zu sein scheint.

Um dieses Problem zu lösen, wird auf d (negativ) der zuvor ermittelte Wert n aufaddiert (siehe Anlage 12, Zeile 189).

$$d \in \mathbb{Z} \mid d > 0 \quad \rightarrow \quad \rightarrow \quad d$$

$$d \in \mathbb{Z} \mid d < 0 \quad \rightarrow \quad d + n \quad \rightarrow \quad d$$

### 4.4 Kodieren und Dekodieren mittels RSA

Für die Kodierung und Dekodierung wird praktisch das gleiche Verfahren verwendet (siehe Anlage 12, Zeile 193 - 203). Der e oder d Teil des Schlüssels wird je nach Kodierungsrichtung gewählt und als Exponent für den zu bearbeitenden Wert eingesetzt. Diese Potenz muss nun lediglich mit dem zweiten Teil des Schlüssels (n) modularisiert werden.

$$x^e \bmod n = y \qquad \text{bzw.} \qquad y^d \bmod n = x$$

Theoretisch sind nun alle Grundlagen gegeben, um mit der RSA-Verschlüsselung arbeiten zu können. Auch in praktischen Beispielen mit geringen p-q-Werten wird nicht zwangsläufig sichtbar, dass das Potenzieren (solange möglich mittels Taschenrechner) in der Realität für Probleme sorgt. Da jedoch, aus Sicherheitsgründen, wirklich sehr große p-q-Werte verwendet werden, erhält man schnell noch größere Schlüssel. Schon Exponenten mit vier bis sechs Stellen (wie im vom Verfasser erstellten RSA-Beispiel) sorgen für ein drastisches Problem, da sie ohne enormen Zeitaufwand selbst mit Computern (aufgrund der Speichergröße) nicht zu lösen sind.
Glücklicherweise müssen diese Potenzen nicht gelöst werden, da sie mit Hilfe der binären Modulation [1, Seite 299] in Teilpotenzen zerlegt werden können. Diese Teilpotenzen werden ebenfalls mit dem zweiten Teil des Schlüssels (n) modularisiert. Die erste zu lösenden Potenz hat den Exponenten 1, die zweite den Exponenten 2, die dritte den Exponenten 4, die vierte den Exponenten 8 und in gleicher Weise immer weitere Verdoppelungen der vorhergehenden Exponenten. Der Rest aus der je vorhergehenden Rechnung ins Quadrat entspricht dem Ergebnis der darauf folgenden Potenz, welche mit dem zweiten Teil des Schlüssels (n) wieder ihrerseits modularisiert wird. Im nächsten Schritt sucht man unter den Teil-Exponenten nach denen, welche, addiert man sie, den ursprünglichen Exponenten (bzw. den Schlüssel) ergeben. Hat man diese Potenzen lokalisiert, so müssen die Ergebnisse aus den entsprechenden Modularisierungen multipliziert und danach, ein weiteres mal durch n, modularisiert werden.
Bei der praktischen Umsetzung dieser letzten Multiplikation zeigte sich dem Verfasser, dass das entstehende Produkt die Speichermöglichkeiten der größtmöglichen üblichen Datentypen zur Zahlenspeicherung (z.B. Integer oder sogar Int64) übersteigt. Um das Problem zu lösen, hat der Verfasser die Methode entwickelt nach jeder Multiplikation (mit

nie mehr als zwei Faktoren) das Produkt zu modularisieren um es daraufhin in gleicher Art mit dem nächsten Wert zu multiplizieren, woraufhin auch dieses Produkt wieder modularisiert werden muss (siehe Anlage 12, Zeile 166)
Die letzte Modulation bildet also das Ergebnis des ursprünglichen Potenz-Modulus.

## 5 Fazit

Es gibt viele veraltete Verfahren zur Verschlüsselung von Informationen. Heutzutage fällt es uns leicht, die Texte, welche mit Hilfe von diesen Verfahrensweisen (z.B. Caesar-Chiffre) verschlüsselt wurden, (kryptoanalytisch) zu entschlüsseln.
Doch auch diese Verfahrensweisen galten einst als sicher. Der Verfasser bemerkt also, dass auch die heutzutage als sicher geltenden Systeme zur Verschlüsselung eines Tages (z.B. durch die Entwicklung noch schnellerer Computertechnik) veraltet sein werden.
Dennoch gibt es in unserer Zeit sichere Verfahren, auf welche man sich verlassen kann.
Der Verfasser stellte sich in der Einleitung die Frage: „Kann man die moderne Technik überhaupt noch verwenden, um vertrauliche Inhalte zu besprechen?“ und kommt nun zur Überzeugung, dass die vertraulichen Informationen selbst sicher sind und (unter Anwendung von RSA) dies während der Übertragung auch bleiben. Leider lässt sich nicht verhindern, dass die Übertragung als solches festgestellt werden kann. Daher wird es Angreifern immer möglich sein, den Zeitpunkt eines Informationsaustausches (und die beteiligten Computer) festzustellen, und allein daraus unter Umständen gewisse Rückschlüsse ziehen zu können.
Die Wahl der Kommunikationsart sollte in jedem Fall der Vertraulichkeit der zu behandelnden Informationen entsprechend abgesichert werden.

## Anhang

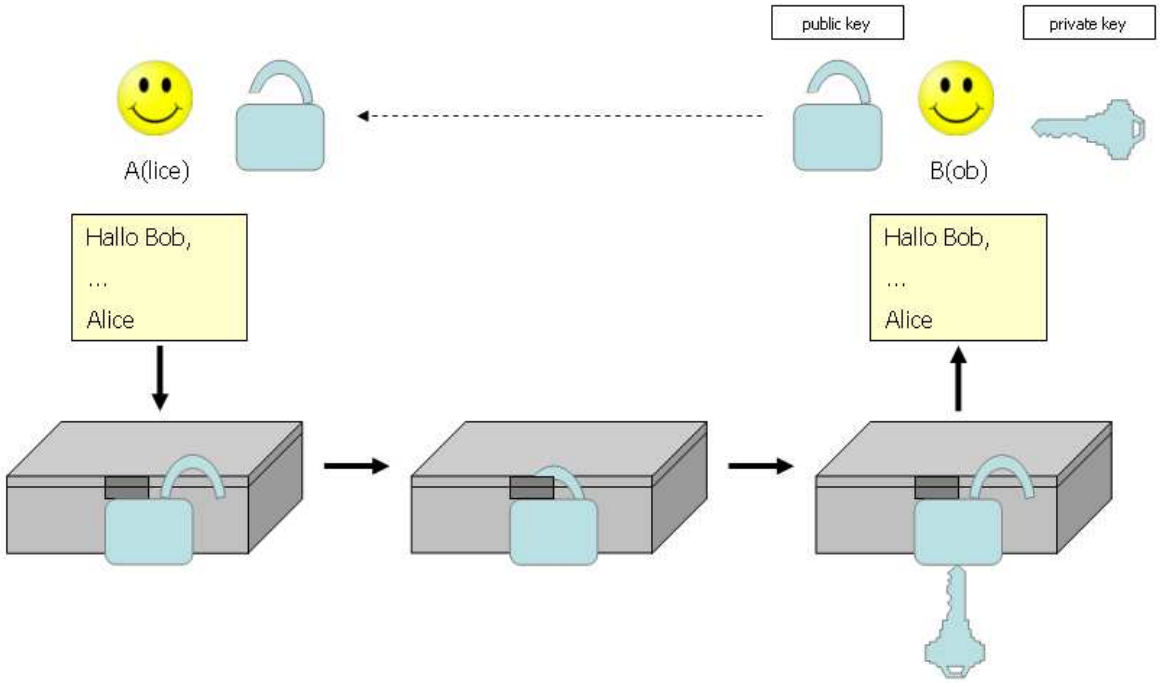
### Worterklärungen

| | | |
|---|---|---|
| Kryptosystem | = | ein Verschlüsselungssystem auf mathematischer Basis |
| Kryptoanalyse | = | das schematische suchen bzw. finden eines Schlüssels |
| Chiffrieren | = | das übersetzen von Klartext zu Geheimalphabet |
| Kodieren | = | mathematisches ermitteln des Geheimalphabet Zeichen |
| Monoalphabetisch | = | mit nur einem Alphabet |
| Polyalphabetisch | = | mit mehr als einem Alphabet |
| Symmetrisch | = | zwei gleiche Seiten (bzw. Schlüssel) |
| Asymmetrisch | = | zwei unterschiedliche Seiten (bzw. Schlüssel) |

### Anlagenverzeichnis

| Binary | Oct | Dec | Hex | Glyph | Binary | Oct | Dec | Hex | Glyph | Binary | Oct | Dec | Hex | Glyph |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 010 0000 | 040 | 32 | 20 | | 100 0000 | 100 | 64 | 40 | @ | 110 0000 | 140 | 96 | 60 | ` |
| 010 0001 | 041 | 33 | 21 | ! | 100 0001 | 101 | 65 | 41 | A | 110 0001 | 141 | 97 | 61 | a |
| 010 0010 | 042 | 34 | 22 | " | 100 0010 | 102 | 66 | 42 | B | 110 0010 | 142 | 98 | 62 | b |
| 010 0011 | 043 | 35 | 23 | # | 100 0011 | 103 | 67 | 43 | C | 110 0011 | 143 | 99 | 63 | c |
| 010 0100 | 044 | 36 | 24 | $ | 100 0100 | 104 | 68 | 44 | D | 110 0100 | 144 | 100 | 64 | d |
| 010 0101 | 045 | 37 | 25 | % | 100 0101 | 105 | 69 | 45 | E | 110 0101 | 145 | 101 | 65 | e |
| 010 0110 | 046 | 38 | 26 | & | 100 0110 | 106 | 70 | 46 | F | 110 0110 | 146 | 102 | 66 | f |
| 010 0111 | 047 | 39 | 27 | ' | 100 0111 | 107 | 71 | 47 | G | 110 0111 | 147 | 103 | 67 | g |
| 010 1000 | 050 | 40 | 28 | ( | 100 1000 | 110 | 72 | 48 | H | 110 1000 | 150 | 104 | 68 | h |
| 010 1001 | 051 | 41 | 29 | ) | 100 1001 | 111 | 73 | 49 | I | 110 1001 | 151 | 105 | 69 | i |
| 010 1010 | 052 | 42 | 2A | * | 100 1010 | 112 | 74 | 4A | J | 110 1010 | 152 | 106 | 6A | j |
| 010 1011 | 053 | 43 | 2B | + | 100 1011 | 113 | 75 | 4B | K | 110 1011 | 153 | 107 | 6B | k |
| 010 1100 | 054 | 44 | 2C | , | 100 1100 | 114 | 76 | 4C | L | 110 1100 | 154 | 108 | 6C | l |
| 010 1101 | 055 | 45 | 2D | - | 100 1101 | 115 | 77 | 4D | M | 110 1101 | 155 | 109 | 6D | m |
| 010 1110 | 056 | 46 | 2E | . | 100 1110 | 116 | 78 | 4E | N | 110 1110 | 156 | 110 | 6E | n |
| 010 1111 | 057 | 47 | 2F | / | 100 1111 | 117 | 79 | 4F | O | 110 1111 | 157 | 111 | 6F | o |
| 011 0000 | 060 | 48 | 30 | 0 | 101 0000 | 120 | 80 | 50 | P | 111 0000 | 160 | 112 | 70 | p |
| 011 0001 | 061 | 49 | 31 | 1 | 101 0001 | 121 | 81 | 51 | Q | 111 0001 | 161 | 113 | 71 | q |
| 011 0010 | 062 | 50 | 32 | 2 | 101 0010 | 122 | 82 | 52 | R | 111 0010 | 162 | 114 | 72 | r |
| 011 0011 | 063 | 51 | 33 | 3 | 101 0011 | 123 | 83 | 53 | S | 111 0011 | 163 | 115 | 73 | s |
| 011 0100 | 064 | 52 | 34 | 4 | 101 0100 | 124 | 84 | 54 | T | 111 0100 | 164 | 116 | 74 | t |
| 011 0101 | 065 | 53 | 35 | 5 | 101 0101 | 125 | 85 | 55 | U | 111 0101 | 165 | 117 | 75 | u |
| 011 0110 | 066 | 54 | 36 | 6 | 101 0110 | 126 | 86 | 56 | V | 111 0110 | 166 | 118 | 76 | v |
| 011 0111 | 067 | 55 | 37 | 7 | 101 0111 | 127 | 87 | 57 | W | 111 0111 | 167 | 119 | 77 | w |
| 011 1000 | 070 | 56 | 38 | 8 | 101 1000 | 130 | 88 | 58 | X | 111 1000 | 170 | 120 | 78 | x |
| 011 1001 | 071 | 57 | 39 | 9 | 101 1001 | 131 | 89 | 59 | Y | 111 1001 | 171 | 121 | 79 | y |
| 011 1010 | 072 | 58 | 3A | : | 101 1010 | 132 | 90 | 5A | Z | 111 1010 | 172 | 122 | 7A | z |
| 011 1011 | 073 | 59 | 3B | ; | 101 1011 | 133 | 91 | 5B | [ | 111 1011 | 173 | 123 | 7B | { |
| 011 1100 | 074 | 60 | 3C | < | 101 1100 | 134 | 92 | 5C | \ | 111 1100 | 174 | 124 | 7C | \| |
| 011 1101 | 075 | 61 | 3D | = | 101 1101 | 135 | 93 | 5D | ] | 111 1101 | 175 | 125 | 7D | } |
| 011 1110 | 076 | 62 | 3E | > | 101 1110 | 136 | 94 | 5E | ^ | 111 1110 | 176 | 126 | 7E | ~ |
| 011 1111 | 077 | 63 | 3F | ? | 101 1111 | 137 | 95 | 5F | _ | | | | | |

Anlage 1: Tabelle – The American Standard Code for Information Interchange [2]




Anlage 2: Grafik – Ein Asymmetrisches Kryptosystem [3]

## Vigenère-Quadrat

| Klartext-Alphabet | Schlüsselwort-Alphabet | | | | | | | | | | | | | | | | | | | | | | | | | Chiffre-Text | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | M | E | U | D | O | Q | N | I | P | Q | M | D | Z | I | E | X |
| A | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | | E | | | | | | | | | | | | | | |
| B | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | | | | | | | | | | | | | | | | |
| C | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | | | | | | | | | | | | | | | | |
| D | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | | | | | | | | | | | | | | | | |
| E | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | | | | | | Q | | I | | | M | | | I | | |
| F | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | | | | | | | | | | | | | | | | |
| G | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | M | | | | O | | | | | | | | | | | |
| H | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | | | | | | | N | | | | | | | | | |
| I | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | | | | | | | | | P | | | | | | | |
| J | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | | | | | | | | | | | | | | | | |
| K | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | | | | | | | | | | | | | | | | |
| L | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | | | | | | | | | | | | | | | | |
| M | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | | | | | | | | | | Q | | | | | | |
| N | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | | | U | | | | | | | | | | | | | |
| O | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | | | | | | | | | | | | | | | | |
| P | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | | | | | | | | | | | | | | | | |
| Q | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | | | | | | | | | | | | | | | | |
| R | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | | | | | | | | | | | | D | | | | |
| S | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | | | | | | | | | | | | | | | | |
| T | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | | | | | | | | | | | | | Z | | | X |
| U | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | | | | | | | | | | | | | | | | |
| V | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | | | | | | | | | | | | | | | | |
| W | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | | | | | | | | | | | | | | | | |
| X | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | | | | | | | | | | | | | | | E | |
| Y | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | | | | | | | | | | | | | | | | |
| Z | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | | | | D | | | | | | | | | | | | |

Anlage 3: Grafik – Beispiel zum Vigenère-Quadrat

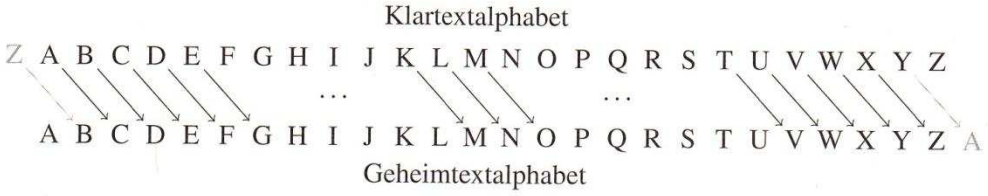



**Abbildung 1.6** Darstellung der Verschlüsselung mit CAESAR mit dem Schlüssel 2.

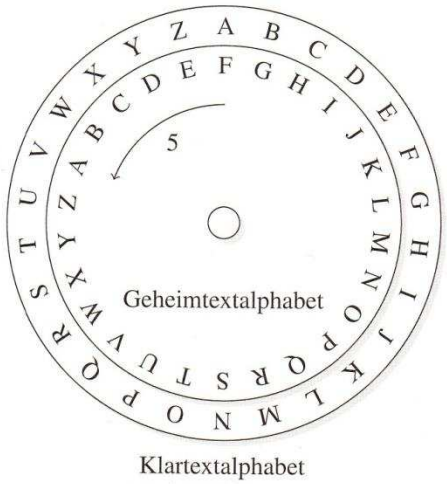
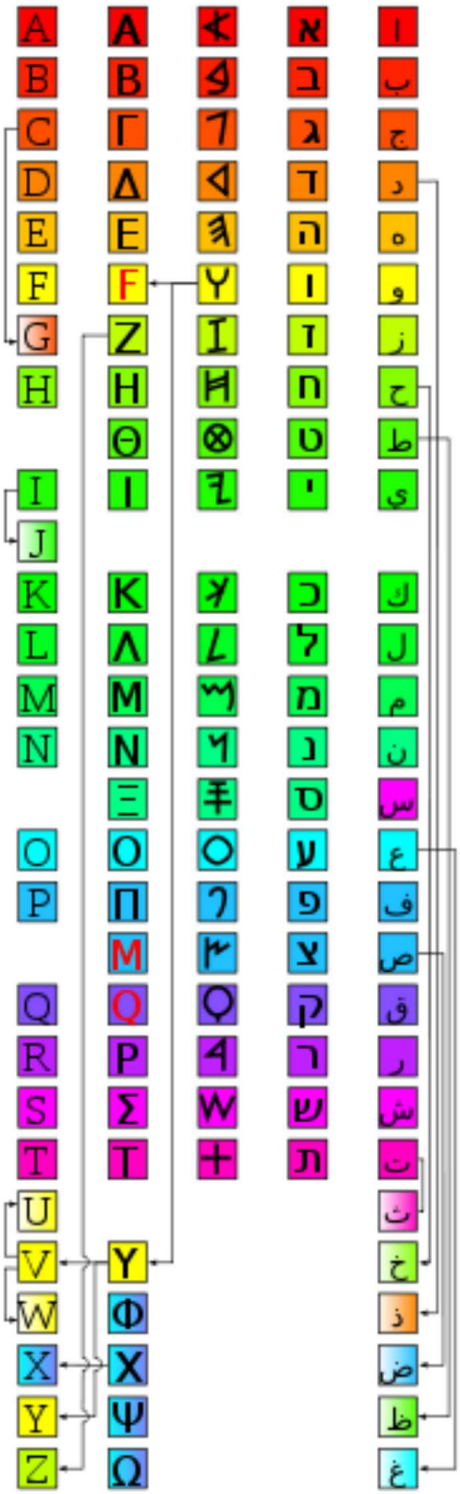



**Abbildung 1.7** Mit diesen beiden Scheiben kann die Ver- und Entschlüsselung mit CAESAR einfach ausgeführt werden. Der Schlüssel (hier 5) bestimmt die Anzahl der Buchstaben, um die die innere Scheibe im Gegenuhrzeigersinn gedreht wird. Danach werden die Klartextbuchstaben auf der äusseren Scheibe durch die Geheimtextbuchstaben der inneren Scheibe codiert.

Anlage 4: Grafik/ Text – Caesar Verschlüsselung [1, Seite 25]



*Das phönizische Alphabet (mittlere Säule) ist die Mutter verschiedener heutiger Alphabete. V.l.n.r.: lateinisch, griechisch, phönizisch, hebräisch, arabisch. Die modernen Äquivalente der phönizischen Buchstaben stehen auf selber Höhe wie die „Originale“ in der mittleren Spalte. Verwandte Buchstaben sind im gleichen Farbton hinterlegt. Pfeile ordnen Buchstaben ihrem jeweiligen Äquivalent zu.*

Anlage 5: Grafik/ Text – Entwicklung der Schriftsysteme [6]

# Erweiterter Euklidischer Algorithmus

ggT( a, b ) = 2

| | | | | | | |
|---|---|---|---|---|---|---|
| 128 | = | 34 | * | 3 | + | 26 |
| 34 | = | 26 | * | 1 | + | 8 |
| 26 | = | 8 | * | 3 | + | 2 |
| 8 | = | 2 | * | 4 | + | 0 |

Euklidischer Algorithmus

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | = | **0** | * | 8 | + | **1** | * | 2 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **-3** | = | 0 | - | 3 | * | **1** |
| **4** | = | 1 | - | 1 | * | **-3** |
| **-15** | = | -3 | - | 3 | * | **4** |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | = | 4 | * | 128 | + | -15 | * | 34 |

^
Ergebniss

Vereinfacht dargestellt

| a | b | q | r | x | y | |
|---|---|---|---|---|---|---|
| 128 | 34 | 3 | 26 | 4 | *-15* | |
| 34 | 26 | 1 | 8 | -3 | *4* | < Erg. |
| 26 | 8 | 3 | **2** | 1 | *-3* | |
| 8 | 2 | 4 | 0 | 0 | *1* | |

**Bemerkung**

4*128 mod 34 = 2 = ggT(128,34)
Es handelt sich im Beispiel **nicht** um die Bildung eines multiplikativen Inversen.
Dieses wird nur erzeugt, wenn der ggT(a,b) = 1

Anlage 6: Grafik – Der Erweiterte Euklidische Algorithmus

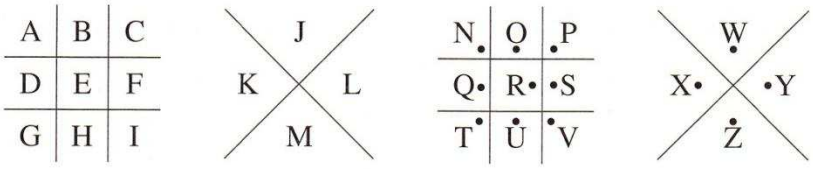



**Abbildung 1.5** Das Schema für die Chiffrierung von Buchstaben mit der Geheimschrift FREIMAURER. Die Buchstaben werden jeweils durch die Linien und Punkte in ihrer Umgebung chiffriert. So ist zum Beispiel A = ⊥, E = □, M = ∧, R = ⊡, Y = ⋖ .

Anlage 7: Grafik/ Text – Freimaurer als Beispiel für Geheimschriften [1, Seite 22]

**Tabelle 3.1** In dieser Tabelle sind die erwarteten relativen Häufigkeiten der Buchstaben in deutschen Texten aufgeführt. Die relativen Häufigkeiten sind in Prozent angegeben. Die Umlaute Ä, Ö, Ü wurden wie AE, OE, UE behandelt und ß wurde durch SS ersetzt [aus 15].

| Buchstabe | Relative Häufigkeit (%) | Buchstabe | Relative Häufigkeit (%) | Buchstabe | Relative Häufigkeit (%) | Buchstabe | Relative Häufigkeit (%) |
|---|---|---|---|---|---|---|---|
| E | 17,74 | H | 5,22 | O | 2,39 | V | 0,64 |
| N | 10,01 | D | 5,12 | B | 1,85 | J | 0,23 |
| I | 7,60 | U | 4,27 | W | 1,73 | Y | 0,04 |
| R | 6,98 | L | 3,49 | F | 1,56 | X | 0,02 |
| S | 6,88 | C | 3,26 | K | 1,40 | Q | 0,01 |
| A | 6,43 | M | 2,75 | Z | 1,10 | | |
| T | 5,94 | G | 2,69 | P | 0,64 | | |

Anlage 8: Tabelle – Häufigkeit der Buchstaben in deutschen Texten [1, Seite 85]

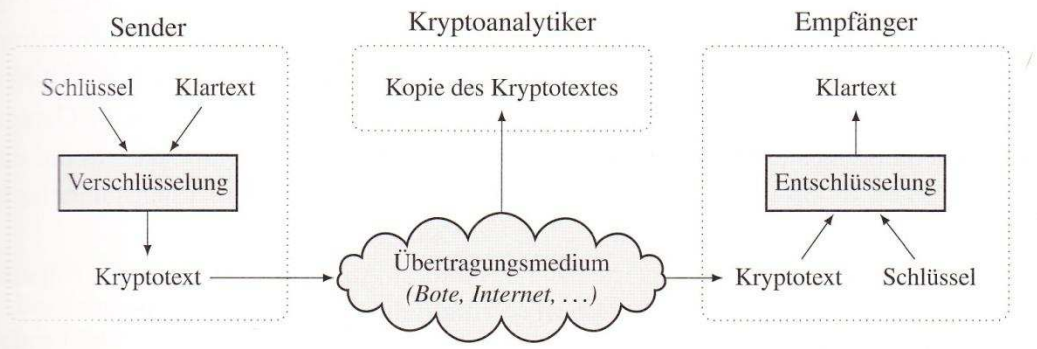



**Abbildung 2.1** Dieses Kommunikationsschema zeigt die Übertragung einer verschlüsselten Nachricht (dem Kryptotext). Bei der Übertragung kann eine Kopie des Kryptotextes in die Hände einer unbefugten dritten Person – genannt Kryptoanalytiker – gelangen.

Anlage 9: Grafik – Ein Kommunikationsschema [1, Seite 37]

**Tabelle 1.1** Die Ordnungen der Buchstaben aus dem Alphabet Lat.

| Buchst. | Ord. | Buchst. | Ord. | Buchst. | Ord. | Buchst. | Ord. | Buchst. | Ord. |
|---|---|---|---|---|---|---|---|---|---|
| **A** | 0 | **G** | 6 | **L** | 11 | **Q** | 16 | **V** | 21 |
| **B** | 1 | **H** | 7 | **M** | 12 | **R** | 17 | **W** | 22 |
| **C** | 2 | **I** | 8 | **N** | 13 | **S** | 18 | **X** | 23 |
| **D** | 3 | **J** | 9 | **O** | 14 | **T** | 19 | **Y** | 24 |
| **E** | 4 | **K** | 10 | **P** | 15 | **U** | 20 | **Z** | 25 |
| **F** | 5 | | | | | | | | |

Anlage 10: Tabelle – Die Ordnungen aus dem Alphabet Lat. [1, Seite 13]

**1.** Wir zerlegen zuerst die beiden Zahlen 16 und 24 in **Primfaktoren**.

**2.** Primzahlen, die in beiden Zerlegungen vorkommen, werden **unterstrichen**.

In unserem Fall kommt der Faktor 2 in einer Zerlegung 4mal vor, in der anderen Zerlegung nur 3mal. Man muss ihn daher 3mal unterstreichen.

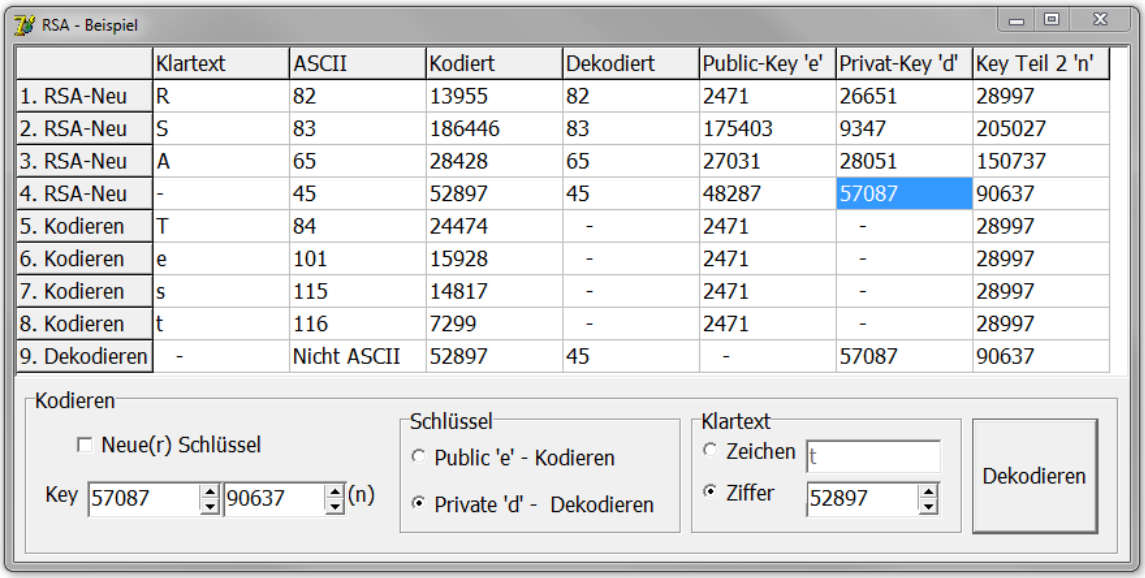
Der Faktor 3 kommt in einer Zerlegung 1mal vor, in der anderen Zerlegung gar nicht. Daher wird hier nichts mehr unterstrichen.

**3.** Die gemeinsamen Faktoren werden nun miteinander **multipliziert**, um den größten gemeinsamen Teiler zu erhalten.

| | |
|---|---|
| 16 | 2 |
| 8 | 2 |
| 4 | 2 |
| 2 | 2 |
| 1 | |

| | |
|---|---|
| 24 | 2 |
| 12 | 2 |
| 6 | 2 |
| 3 | 3 |
| 1 | |

Anlage 11: Text/ Grafik – Die Primfaktorenzerlegung [5]



RSA - Beispiel

| | Klartext | ASCII | Kodiert | Dekodiert | Public-Key 'e' | Privat-Key 'd' | Key Teil 2 'n' |
|---|---|---|---|---|---|---|---|
| 1. RSA-Neu | R | 82 | 13955 | 82 | 2471 | 26651 | 28997 |
| 2. RSA-Neu | S | 83 | 186446 | 83 | 175403 | 9347 | 205027 |
| 3. RSA-Neu | A | 65 | 28428 | 65 | 27031 | 28051 | 150737 |
| 4. RSA-Neu | - | 45 | 52897 | 45 | 48287 | 57087 | 90637 |
| 5. Kodieren | T | 84 | 24474 | - | 2471 | - | 28997 |
| 6. Kodieren | e | 101 | 15928 | - | 2471 | - | 28997 |
| 7. Kodieren | s | 115 | 14817 | - | 2471 | - | 28997 |
| 8. Kodieren | t | 116 | 7299 | - | 2471 | - | 28997 |
| 9. Dekodieren | - | Nicht ASCII | 52897 | 45 | - | 57087 | 90637 |

Kodieren

Neue(r) Schlüssel

Key 57087 90637 (n)

Schlüssel

Public 'e' - Kodieren

Private 'd' - Dekodieren

Klartext

Zeichen t

Ziffer 52897

Dekodieren

Anlage 12: Skript –RSA-Unit zur Implementierung von RSA in Delphi

**Quellenverzeichnis**

Literatur

[1] Hromkovic, Juraj (Hrsg.); Autorenkollektiv: Einführung in die Kryptologie. Heidelberg Berlin: Vieweg + Teubner Verlag | Springer Fachmedien Wiesbaden GmbH. 2010

Internet

[2] ASCII - From Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/ASCII (22.10.2013)

[3] Asymmetrisches Kryptosystem
http://www.inf-schule.de/content/kommunikation/kryptologie/modernechiffriersysteme/einstieg_asymmetrischeschiffriersystem/asymmetrisches_kryptosystem.png (26.10.2013)

[4] Die Eulersche Phi-Funktion
http://www.uni-protokolle.de/Lexikon/Eulersche_Phi-Funktion.html (29.10.2013)

[5] Größter gemeinsamer Teiler
http://www.mathe-lexikon.at/arithmetik/natuerliche-zahlen/teilbarkeit/groesster-gemeinsamer-teiler-ggt.html (24.10.2013)

[6] Phönizische Schrift
http://de.wikipedia.org/wiki/Ph%C3%B6nizische_Schrift (23.10.2013)

[7] Yearly Report on Algorithms and Keysizes (2011-2012)
http://www.ecrypt.eu.org/documents/D.SPA.20.pdf (30.10.2013)

**Selbstständigkeitserklärung**

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Lugau, 02.12.2013

```
 1: unit uRSA;  // Diese Unit kodiert bzw. dekodiert je eine Zahl.
 2: interface
 3: uses Dialogs, SysUtils, Math;
 4:
 5: type
 6:   TRSA = class
 7:   private
 8:     p:Integer;
 9:     q:Integer;
10:     n:Integer;
11:     FIofN:Integer;
12:     e:Integer;
13:     d:Integer;
14:     function findP:integer;
15:     function findQ(myP:integer):integer;
16:     function findE(FIn:integer):integer;
17:     function findD(eKey,FIn:integer):integer;
18:     function GCD(a,b : integer):integer;
19:     function InttoBin(Number:Integer):string;
20:     function ModExp(Number,Exponent,Divisor:Integer):Integer;
21:   public
22:     constructor Create(NewKey,eKey:Boolean;nKey,Key:Integer);
23:     function get_nKey:Integer;
24:     function get_eKey:Integer;
25:     function get_dKey:Integer; // Nutzer wird angewiesen diesen Wert geheim zu
    halten.
26:     function encode(Num:Integer):integer;  //kodieren von Werten bis 1369, da p & q
    minimal 37
27:     function decode(Num:Integer):integer;  //dekodiert alle möglichen
    verschlüsselten Werte.
28:   end; // alle anderen Werte bleiben geheim!
29:
30: var RSA:TRSA;
31:
32: implementation
33:
34: constructor TRSA.create(NewKey,eKey:Boolean;nKey,Key:Integer);
35: begin
36:   if NewKey=True then   //Wenn ein Neuer Schlüssel erstellt werden soll dann..
37:   begin
38:     Randomize;          //Echten Zufall generieren. (wegen Zufallszahlen)
39:     // 1. Zwei beliebige Primzahlen wählen!
40:     p:=findP;
41:     q:=findQ(p);
42:     // 2. Zweitbestandteil des Schlüssels errechnen
43:     n:=p*q;
44:     // 3. Errechne FIofN
45:     FIofN:=(p-1)*(q-1);
46:     // 4. Generieren von Kpub bzw. e
47:     e:=findE(FIofN);
48:     // 5. Generieren von Kprv bzw. d
49:     d:=findD(e,FIofN);
50:   end else
51:   begin
52:     n:=nKey;
53:     if eKey=True then
54:     begin
55:     e:=Key;
56:     end else
57:     begin
58:     d:=Key;
59:     end;
60:   end;
61: end;
62:
63: function TRSA.get_nKey:Integer;
64: begin
65:   if n=0 then ShowMessage('Nicht möglich! ''n'' ist unbekannt.')
66:          else Result:=n;
67: end;
68:
69: function TRSA.get_eKey:Integer;
70: begin
71:   if e=0 then ShowMessage('Nicht möglich! ''e'' ist unbekannt.')
72:          else Result:=e;
73: end;
74:
```

```
 75: function TRSA.get_dKey:Integer;
 76: begin          // Der Teil 'd' von KeyPrivate sollte (normalerweiße) NIE außerhalb
      sichtbar werden
 77:   if d=0 then ShowMessage('Nicht möglich! ''d'' ist unbekannt.')
 78:          else Result:=d;   // zum Beweiss der Scriptfunktion wird dieser Wert
      trotzdem ausgegeben.
 79: end;
 80:
 81: function TRSA.findP:integer;
 82: var i,myP:Integer;
 83:     Prime:Boolean;
 84: begin
 85:   repeat                  //Versuch p zu finden.
 86:     myP:=random(500)+32;  //p wählen (Eingeschränkt um RAM nicht zu überlasten)
 87:     Prime:=True;          //These: p ist Primzahl
 88:     for i:=2 to myP-1 do      //Prüfe ob p durch Zahlen von 2 bis p-1 teilbar ist.
 89:     begin
 90:       if myP-i*(myP div i) = 0 then Prime:=False;
 91:     end;
 92:   until Prime = True;  //Versuch erfolgreich beenden, wenn p eine Primzahl ist.
 93:   Result:=myP;
 94: end;
 95:
 96: function TRSA.findQ(myP:integer):integer;
 97: var i,myQ:Integer;
 98:     Prime:Boolean;
 99: begin  // das Gleiche für q wie für p
100:   repeat
101:     myQ:=random(500)+32; Prime:=True;
102:     for i:=2 to myQ-1 do if myQ-i*(myQ div i) = 0 then Prime:=False;
103:   until (Prime = True) and (myQ <> myP); //Prüfe ob q eine andere Primzahl als p
104:   Result:=myQ;
105: end;
106:
107: function TRSA.findE(FIn:integer):integer;
108: var myE:Integer;
109: begin
110:   repeat
111:     myE:=random(FIn-1)+1;  //Zufällige Wahl von e (von 1 bis FIn)
112:   until (GCD(myE,FIn)=1);  //Prüfe ob ggT(e,FIn) = 1
113:   Result:=myE;
114: end;
115:
116: function TRSA.GCD(a, b: Integer): Integer;
117: var rest: Integer;
118: begin
119:   repeat //ggT Berechnung nach dem Algorithmus von Euklid
120:     rest := a mod b;
121:     a    := b;
122:     b    := rest;   //Restbildung
123:   until (rest = 0); //bis Rest = 0
124:   Result := abs(a);
125: end;
126:
127: function TRSA.InttoBin(Number:Integer):string; //Ermitteln des binären Zahlwertes
128: var bin:string;
129:     i:Integer;
130: begin
131:   i:=Number; bin:='';
132:   repeat
133:     if (i/2 <> i div 2) then bin:= '1' + bin else bin:= '0' + bin ;
134:     i:=i div 2;       // Wenn es einen Rest gibt 1, sonst 0 anfügen.
135:   until i=0;
136:   result:=bin;
137: end;
138:
139: function TRSA.ModExp(Number,Exponent,Divisor:Integer):Integer;
140: var turnbinExp,test:string;
141:     i,count,j:Integer;
142:     value:array[1..256] of Integer;
143:     easy:Int64;
144: begin
145:   count:=0;
146:   i:=1; //Erster möglicher Teil-Exponent
147:   repeat
148:     count:=count+1;
149:     if i<3 then
```

```
150:     begin      // Number^i mod Divisor
151:       value[count]:= Round(Power(Number, i)) mod Divisor;
152:     end else
153:     begin      // Ergebnis der vorhergehenden Operation^2 mod Divisor
154:       value[count]:= Round(Power(value[count-1],2)) mod Divisor;
155:     end;
156:     i:=i*2; //Verdoppeln des Teil-Exponenten
157:   until i > Exponent;
158:
159:   easy:=0; //Der Wert wird vor der Verwendug zur Sicherheit geleert.
160:   for j:=1 to Length(InttoBin(Exponent)) do turnbinExp:=InttoBin(Exponent)[j]+
      turnbinExp;
161:   for j:=1 to Length(turnbinExp) do
162:   begin
163:     if turnbinExp[j]='1' then //Bestimmen der Bestandteile des Ziel-Exponenten durch
      Binärzahl.
164:     begin
165:       if easy = 0 then easy:=value[j]
166:       else easy:= easy*value[j] mod Divisor
167:     end;
168:   end;
169:   result:=easy mod Divisor;
170:   //easy ist das Produkt der Reste, aus den (Restberechnungen mit)
      Teil-Exponenten(i),
171:   //deren Summe dem (original) Exponent entspricht.
172: end;
173:
174: function TRSA.findD(eKey,FIn:integer):integer;
175: var i:Integer;
176:     q,r,y: array[1..256] of Integer;
177: begin
178:   r[1]:=eKey; r[2]:=FIn; i:=2;
179:   repeat //Euklidischer Alg.
180:     i:=i+1;
181:     q[i]:=r[i-2] div r[i-1];
182:     r[i]:=r[i-2] mod r[i-1];
183:   until r[i]=0; // bis Rest 0
184:   y[i+1]:=0; y[i]:=1;
185:   repeat //ab hier erweiterter Euklidischer Alg.
186:     i:=i-1;
187:     y[i]:=y[i+2]-q[i]*y[i+1]
188:   until i=4;
189:   if y[i]<0 then Result:= y[i]+ FIn //Verhindern, dass die Zahl negativ wird.
190:             else Result:= y[i];
191: end;
192:
193: function TRSA.encode(Num:Integer):Integer;
194: begin
195:   if e=0 then ShowMessage('Nicht möglich! ''e'' ist unbekannt.')
196:          else Result:= ModExp(Num,e,n);
197: end;
198:
199: function TRSA.decode(Num:Integer):integer;
200: begin
201:   if d=0 then ShowMessage('Nicht möglich! ''d'' ist unbekannt.')
202:          else Result:= ModExp(Num,d,n);
203: end;
204:
205: end.
```