Johann-Mathesius-Gymnasium
Rochlitz

Schuljahr 2017/2018

Komplexe Leistung

Thema : Darstellung der Funktionsweise von Verschlüsselungsverfahren

Fach : Mathematik

Verfasser : Moritz Enge

Betreuender Fachlehrer : Herr Fritz

Abgabetermin : 27.Oktober 2017

Inhaltsverzeichnis

1. **Einleitung**
2. **Mathematische Grundlagen**
   - 2.1. Modulo-Division
   - 2.2. Primzahlen
   - 2.3. Eulersche φ-Funktion
   - 2.4. Teilerfremdheit zweier Zahlen
   - 2.5. Multiplikatives Inverses bezüglich mod n
3. **Arten der Verschlüsselungen**
   - 3.1. Symmetrische Verschlüsselung
   - 3.2. Asymmetrische Verschlüsselung
   - 3.3. Hybride Verschlüsselung
4. **Vorstellung spezieller Verschlüsselungsverfahren**
   - 4.1. Vigenère-Chriffe
   - 4.2. RSA – Verschlüsselung
5. **Eigenleistung**
6. **Fazit**
7. **Anhang**
   - 7.1. Literaturverzeichnis
   - 7.2. Anlagen
   - 7.3. Selbstständigkeitserklärung

1. Einleitung

Eine der wichtigsten Errungenschaft der Sicherheit heutzutage ist die Verschlüsselung.
Ohne sie wäre eine erfolgreiche digitale Existenz unmöglich. Ob bei dem Übertragen wichtiger Staatsgeheimnisse oder dem Sichern der Firmendaten gegen Unbefugte, überall wird sie angewandt.
Die ersten bekannten Verschlüsselungsmethoden wurden ca.1900 vor Christus von den Ägyptern „erfunden“. Zu dieser Zeit wurden die Zeichen des Klartextes durch Hieroglyphen ersetzt. Nicht bekannt ist , ob es vor den Ägyptern schon andere Kryptoverfahren gab.
Im Laufe der Zeit verbesserten sich die Verfahren und die Sicherheit und somit wurde im 20. Jahrhundert von Arthur Scherbius die ENIGMA entwickelt.
Sie wurde zur sicheren Datenübertragung im dritten Reich benutzt, so lange bis der britische Logiker, Mathematiker, Kryptoanalytiker und Informatiker Alan Turing eine effiziente Methode fand, die mit ENIGMA verschlüsselten Nachrichten zu entschlüsseln.
Die Verschlüsselung lässt sich in drei Teile unterteilen. Erstens das Erzeugen eines Schlüssels, dies kann durch das „einfache“ Auswählen einer beliebigen Zahl oder durch komplexe mathematische Verfahren geschehen. Danach folgt das Verschlüsseln (Encoding) mit dem davor generierten Schlüssel, bei diesem Schritt erhält man aus dem Klartext den Geheimtext, das sogenannte Chiffrat. Um aus dem Geheimtext wieder den Klartext zu erzeugen benötigt man den letzten Schritt, das Entschlüsseln (Decoding), dazu wird der Geheimtext und ein Entschlüsselungs-Schlüssel benötigt. Je nachdem welches Verfahren angewandt wird, ist der Schlüssel zum Verschlüsseln und Entschlüsseln gleich oder unterschiedlich. Anhand diesen letzten Schrittes kann man die Verschlüsselungsverfahren klassifizieren.Es gibt die symmetrische, die asymmetrische und die hybride Verschlüsselung, auf welche später eingegangen wird.

2. Mathematische Grundlagen

2.1 Modulo-Division

Die Modulo-Division ist wie das, schon aus der Grundschule bekannte Teilen mit Rest.
Es wird aber nur der Rest betrachtet. Man kann Modulo wie jede andere Rechenart verwenden. Als Zeichen verwendet man „mod“, „modulo“ oder „%“.

Beispiel :
20 / 3 = 6 Rest **2** , das heißt 20 mod 3 = **2**

2.2 Primzahlen

Primzahlen sind natürliche Zahlen größer 1, welche nur durch sich selbst und 1 teilbar sind. Die Menge aller Primzahlen trägt das Symbol $\mathbb{P}$.
Sie spielen in der Verschlüsselung eine große Rolle, da die Primfaktorzerlegung des Produktes zweier großer Primzahlen heutzutage immer noch ein großes mathematisches Problem darstellt. Dieses Produkt nimmt in der Realität bis zu mehrere tausend Stellen an. Damit lassen sich die beiden Primzahlen in Lebenszeit nicht bestimmen.

2.3 Eulersche φ-Funktion

Die Eulersche φ-Funktion φ(n) gibt an, wie viele teilerfremde Zahlen es zu n gibt.

Allgemein gilt : $\varphi(n * m) = \varphi(n) * \varphi(m)$

Für Primzahlen ($n \in \mathbb{P}$) gilt : $\varphi(n) = n - 1$

Beispiele : $\varphi(11) = 11-1 = \mathbf{10}$, 11 ist prim

$\varphi(77) = \varphi(7*11) = \varphi(7) * \varphi(11) = 6 * 10 = \mathbf{60}$, 7 und 11 sind prim

2.4 Teilerfremdheit zweier Zahlen

Zwei Zahlen sind teilerfremd, wenn sie keinen gemeinsamen Primteiler (Teiler ist eine Primzahl) haben .

Die Zahlen 12 und 77 sind teilerfremd, denn ihre Primfaktorzerlegungen $12= 2\cdot2\cdot3$ und $77= 7\cdot11$ enthalten keine gemeinsamen Primfaktoren.(1)

2.5 Multiplikatives Inverses bezüglich mod n

Das multiplikative Inverse bezüglich mod n lässt sich durch den erweiterten euklidischen Algorithmus berechnen. Dies ist ein Algorithmus, der neben dem größten gemeinsamen Teiler von den natürlichen Zahlen a und b auch noch zwei natürliche Zahlen m und n berechnet, für die gilt:

$ggT(a,b) = a * m + b * n$

$ggT(a,n) = 1$

Die natürliche Zahl m stellt das multiplikative Inverse bezüglich mod n dar.

3. Arten der Verschlüsselung

3.1 Symmetrische Verschlüsselung

Die symmetrische Verschlüsselung ist ein Teilgebiet der Verschlüsselung, bei welchem beide Teilnehmer meistens denselben Schlüssel verwenden, d. h. $\text{Schlüssel}_{\text{Klartext}} = \text{Schlüssel}_{\text{Geheimtext}}$, oder sich die beiden Schlüssel leicht auseinander berechnen lassen.Man teilt die symmetrische Verschlüsselung nochmal in Block- und Stromchiffren auf. Bei Stromchiffren wird der Klar - bzw. Geheimtext Zeichen für Zeichen ver- bzw. entschlüsselt (2). Ein Beispiel für diese Verschlüsselung ist die Vigenère – Chiffre.Eine Blockchiffre arbeitet mit einer festen Blockgröße und ver- bzw. entschlüsselt mehrere Zeichen auf einmal. Die Größe der Blockgröße wird hinter den Namen des Algorithmus geschrieben, z.B. AES 256 (Advanced Encryption Standard) .
Der Nachteil der symmetrischen Verschlüsselung ist der Transport des Schlüssels zum Gegenüber. Da dieser ja im Klartext transportiert werden müsste und jeder „Angreifer“ diesen Schlüssel abfangen könnte und so der Geheimtext nicht mehr gegenüber dritten geheim ist. Man könnte den Schlüssel auch wieder symmetrisch verschlüsseln, aber da müsste ja der Schlüssel zum Entziffern des Schlüssels irgendwie übertragen werden. Dieses Problem kann man mit Hilfe der hybriden Verschlüsselung lösen (siehe 3.3).

Vorteile	Nachteile
• einfaches Schlüsselmanagement, da nur ein Schlüssel benötigt wird	• Schlüssel darf nicht zu Unbefugten gelangen
• relativ schnelle Ver- und Entschlüsselung	• durch sicheren Übertragungsweg entsteht ein Mehraufwand, z.B. Postweg

3.2 Asymmetrische Verschlüsselung

Die asymmetrische Verschlüsselung ist eine Art der Verschlüsselungsverfahren, bei dem es einen öffentlichen und einen privaten Schlüssel gibt.
Mit dem öffentlichen Schlüssel verschlüsselt man den Klartext zum Geheimtext. Aus dem Geheimtext und dem öffentlichen Schlüssel kann man nicht den Klartext entschlüsseln. Mit dem privaten Schlüssel kann man aus dem Geheimtext den Klartext berechnen. Die ersten asymmetrischen Verfahren wurden in den 1970er‘n gefunden.

Die asymmetrische Verschlüsselung besteht aus mindestens drei Algorithmen :

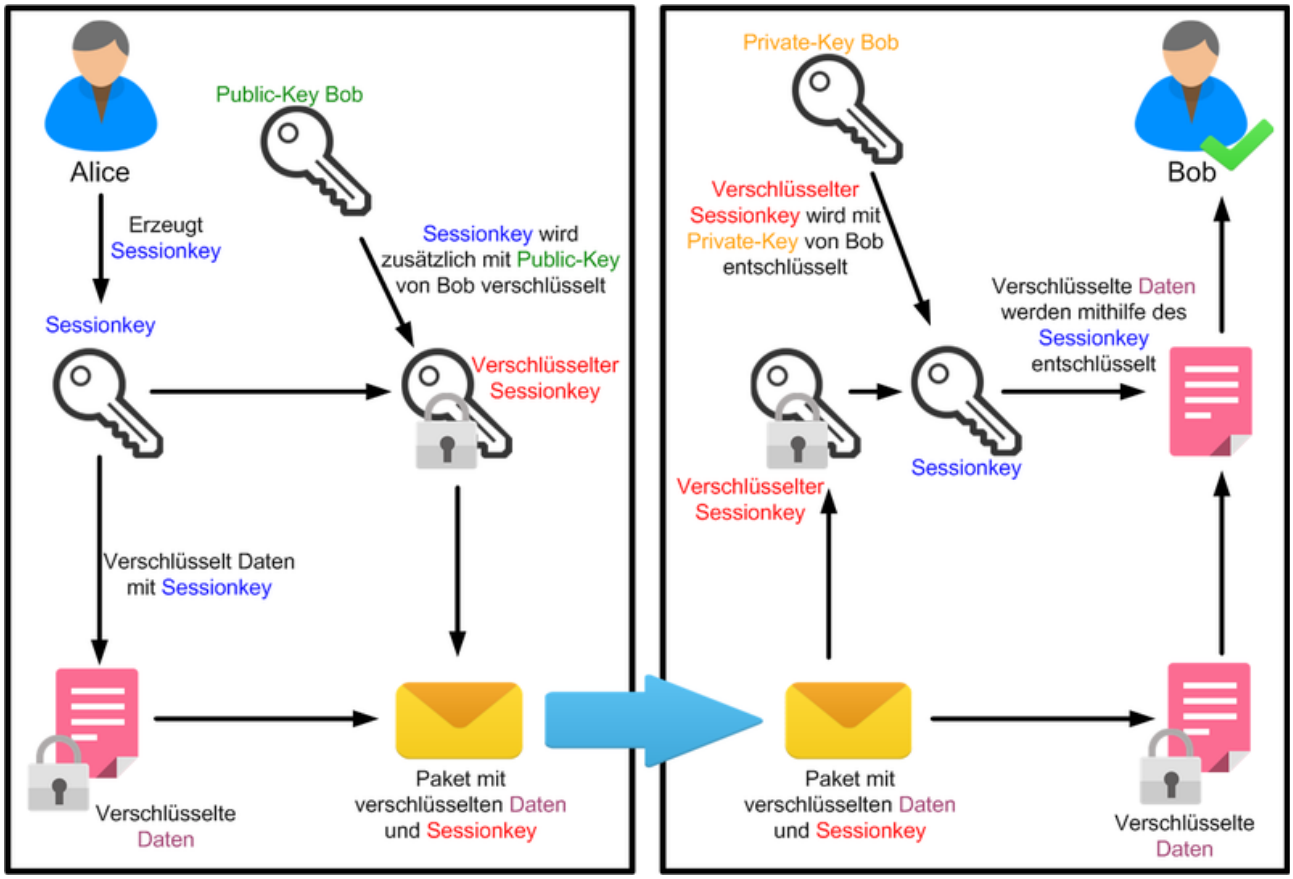
1. Algorithmus zur Schlüsselerzeugung
2. Verschlüsselungsalgorithmus
3. Entschlüsselungsalgorithmus

Dadurch, dass es einen öffentlichen Schlüssel gibt, existiert kein Übertragungsproblem wie bei der symmetrischen Verschlüsselung.
Da jeder, der den öffentlichen Schlüssel kennt, nicht aus dem Geheimtext den Klartext entschlüsseln kann.

Vorteile	**Nachteile**
• hohe Sicherheit • kein Übertragungsproblem	• relativ langsame Ver- und Entschlüsselung • für Sicherheit werden große Schlüssel benötigt

3.3 Hybride Verschlüsselungsverfahren

Das hybride Verschlüsselungsverfahren ist eine Kombination von der symmetrischen Verschlüsselung und der asymmetrischen Verschlüsselung. Bei dieser Verschlüsselung wird der Klartext symmetrisch verschlüsselt, aber der Schlüssel für die symmetrische Verschlüsselung asymmetrisch verschlüsselt übertragen. Man nimmt die symmetrische Verschlüsselung zur Verschlüsselung der Daten, da diese auch bei großen Datenmengen sehr schnell sind (diese sind meistens der Klartext).
Asymmetrische Verschlüsselungsverfahren sind sehr langsam und daher nur geeignet für kleine Datenmengen (der Schlüssel für den Klartext) .



4. Vorstellung spezieller Verschlüsselungsverfahren

4.1 Vigenère-Chriffe

4.1.1 Allgemeines

Die Vigenère-Chriffe ist eine Stromchiffre aus dem 16. Jahrhundert und benannt nach dem Diplomaten Blaise de Vigenère. Der Klartext wird in Einzelzeichen zerlegt und durch Geheimzeichen substituiert, die Mithilfe eines Schlüssels aus dem Vigenère-Quadrat ausgewählt werden. Bei diesem Vigenère-Qudarat handelt es sich um eine quadratische Anordnung von untereinander stehenden verschobenen Alphabeten. (3)

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*
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	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	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	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
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	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
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
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
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	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	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	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
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
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	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	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	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
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	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
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	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	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	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
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	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

Komplexe Leistung Moritz Enge 11CH1

4.1.2 Funktionsweise

Als Erstes wird das Vigenère-Quadrat (siehe 4.1.1) und ein Schlüssel benötigt. Der Schlüssel soll möglichst lang und zufällig sein.

Ist die Länge des Schlüssels = Länge des Klartextes, dann wird der Schlüssel nicht mehrfach verwendet und man erhält eine besondere Version der Vigenère-Chriffe, welche „One-Time-Pad“ genannt wird. Die entsprechenden Geheimtextbuchstaben kann man nun leicht mithilfe des Vigenère-Quadrats ermitteln. Dazu wird der Kreuzungspunkt gesucht, der durch den jeweiligen Schlüsselbuchstaben gekennzeichneten Zeile und der Spalte des Quadrats, die oben durch den Klartextbuchstaben gekennzeichnet ist (4).

4.1.3 Beispiel

Schlüssel	V	I	G	E	N	E	R	E		V	I	G	E	N	E	R	E
Klartext	K	O	M	P	L	E	X	E		L	E	I	S	T	U	N	G
Geheimtext	F	W	S	T	Y	I	I	I		G	M	O	W	G	Y	E	K

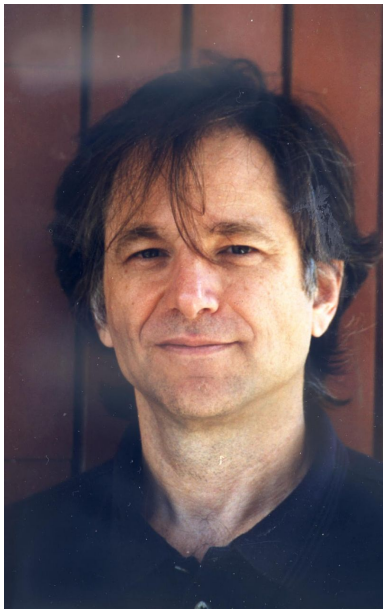
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I 8 6	J	K_{16}	L	M	N	O	P	Q	R	S	T_4	U	V	W_{12}	X	Y_{14}	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O_{11}	P	Q	R	S_3	T	U	V	W	X	Y	Z	A	B	C	D	E	F
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
I	J	K	L	M_{10}	N	O	P	Q	R	S	T	U	V	W_2	X	Y	Z	A	B	C	D	E	F	G	H
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	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	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	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
N	O	P	Q	R	S	T	U	V	W	X	Y_5	Z	A	B	C	D	E	F	G_{13}	H	I	J	K	L	M
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	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	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	S	T	U	V	W	X	Y	Z	A	B	C	D	E_{15}	F	G	H	I_7	J	K	L	M	N	O	P	Q
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	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
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	W	X	Y	Z	A	B	C	D	E	F_1	G_9	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
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	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
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	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

Komplexe Leistung Moritz Enge 11CH1

4.2.1 Allgemeines

Das RSA- Verfahren gilt heutzutage als eines der sichersten Verschlüsselungsalgorithmen der Welt, obwohl es das erste veröffentlichte asymmetrische Verschlüsselungsverfahren ist. Der Name „RSA“ geht auf die drei Erfinder Ronald Linn **R**ivest , Adi **S**hamir und Leonard Max **A**dleman zurück. Dieses Verfahren werden den asymmetrischen Verschlüsselungverfahren zugeordnet. Der private Schlüssel kann in Lebenszeit nicht aus dem öffentlichen Schlüssel berechnet werden.

Ronald L.. Rivest



Leonard M. Adleman

Adi Shamir

4.2.2 Funktionsweise

Erzeugung der öffentlichen und privaten Schlüssel

Als erstes werden zwei unterschiedliche unabhängige Primzahlen **p** und **q** gesucht, welche ungefähr die gleiche Größenordnung haben, aber nicht zu nah aneinander liegen.In der Praxis werden große Zahlen generiert und durch einen Primzahltest so lange geprüft und neu generiert, bis man zwei Primzahlen gefunden hat.

Danach wird das RSA-Modul N berechnet.

Dabei gilt : **N = p * q**

Als nächstes wird das Ergebnis der eulerschen φ-Funktion von N berechnet. Da N ein Produkt von zwei Primzahlen ist gilt:

φ(N) = (p - 1) * (q -1)

Als vorletztes wird eine zu φ(N) teilerfremde Zahl **e** gesucht, es gilt $1 < e < \varphi(N)$, meistens wird aus Effizienzgründen eine relativ kleine Zahl gewählt,

üblich ist die 5.Fermat-Zahl : 2^16 +1 (65537).

Als Letztes wird der Entschlüsselungsexponent **d** berechnet, dieser ist das multiplikative Inverse von e bezüglich des Moduls φ(N), das heißt :

$e * d \equiv 1 \bmod \varphi(N)$

Sind diese ganzen Zahlen berechnet, werden p, q und φ(N) gelöscht, da diese nicht mehr benötigt werden und aus Sicherheitsgründen vernichtet werden sollten. Die Wahl von kleinen Zahlen bei p, q und e führt dazu, dass das Verfahren relativ schnell und effizient geknackt werden kann.

Nach der Berechnung der Zahlen gibt es den öffentlichen Schlüssel **öffentlich_key**, welcher sich aus **e** und **N** zusammensetzt : öffentlich_key(e,N)

Außerdem existiert nun der private Schlüssel **privat_key**, welcher sich aus d und N zusammensetzt : privat_key(d,N)

Verschlüsseln von Nachrichten

Um die Nachricht **m** zu verschlüsseln, wird nach einer Formel der Geheimtext c berechnet. Dazu wird der öffentliche Schlüssel benötigt. Somit kann jeder eine Nachricht verschlüsseln.

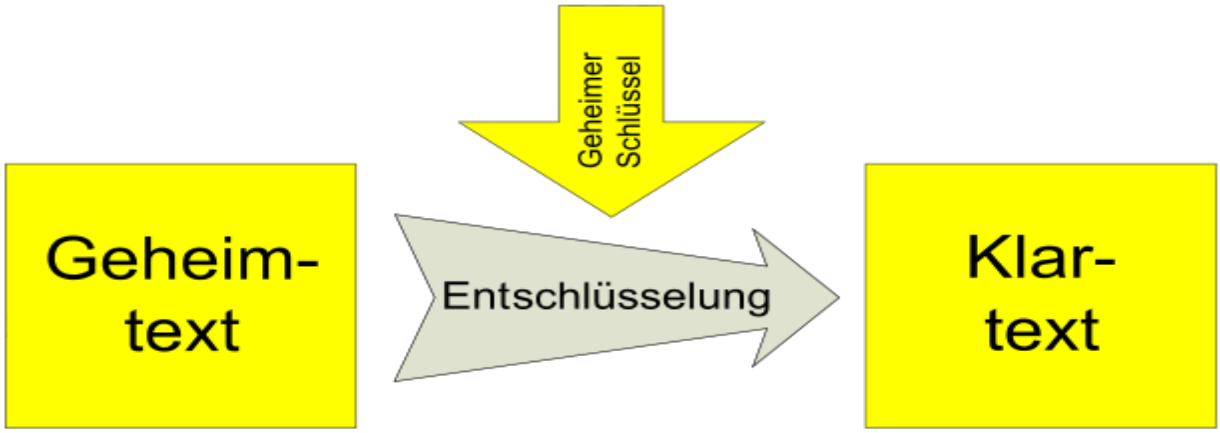
$c \equiv m^{e} \bmod N$

Die Zahl **m** muss außerdem kleiner sein als RSA-Modul N.

Entschlüsseln von Nachrichten

Um aus dem Geheimtext c wieder die Nachricht **m** zu berechnen, wendet man folgende Formel an. Außerdem wird dazu noch der private Schlüssel benötigt, so kann nur der „Ersteller der Schlüssel“ die Nachricht auch wieder entschlüsseln

$m \equiv c^{d} \bmod N$



4.2.3 Beispiel

„zufälliges“ p : 11 (aus rechnerischen Gründen klein gehalten)

„zufälliges“ q : 17 (aus rechnerischen Gründen klein gehalten)

N = p * q = 11 * 17 = 187

ϕ(N) = 11-1 * 17-1 = 10 * 16 = 160

e = 23 (aus rechnerischen Gründen klein gehalten)

e * d ≡ 1 mod ϕ(N) => d = 7

==> öffentlicher Schlüssel (23 , 187) und privater Schlüssel (7 ,187)

Klartext : RSA

Klartext in ASCII-Zahlen : 82 83 65

Verschlüsseln : 82^23 mod 187 = 125 ==> }

83^23 mod 187 = 161 ==> – nicht darstellbar –

65^23 mod 187 = 142 ==> – nicht darstellbar –

Geheimtext in Zahlen : 125 161 142

Entschlüsseln : 125^7 mod 187 = 82 ==> R

161^7 mod 187 = 83 ==> S

142^7 mod 187 = 65 ==> A

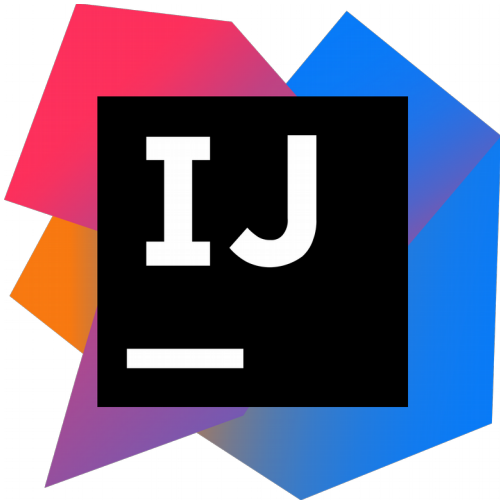
5. Eigenleistung

In meiner Eigenleistung hab eich eine Bibliothek für drei ausgewählte Verschlüsselungsalgorithmen programmiert. Passend dazu wurde auch eine graphische Visualisierung erstellt, dass sogenannte GUI (Graphical User Interface) . Zu den drei Algorithmen zählt die Cäsar Verschlüsselung, die Vigenère Chiffre und der RSA Algorithmus.

Die Bibliothek und das GUI wurden in der imperativ, objektorientierten High Level Sprache Java geschrieben. Ich habe Java als Programmiersprache gewählt, da sie plattformunabhängig ist, das heißt man kann das Programm auf jedem Betriebssystem ausführen, vorausgesetzt die JVM ist installiert. Die JVM ist die „Java Virtual Machine“ welche zur Ausführung von Java-Programmen benötigt wird. Diese JVM kann man für jedes Betriebssystem auf der Seite von „Oracle“ unter dem Name „JRE“ herunterladen und installieren .
JRE steht für „Java Runtime Environment“, dieses beinhaltet unter anderem die JVM.

Zur Realisierung des GUI´s wurde das Java interne Toolkit „JavaFX“ verwendet.

Zur besseren Programmierung wurde die Entwicklungsumgebung „Intelli J“ von JetBrains verwendet.Das Programm „JavaFX Scene Builder 2.0“wurde zur einfacheren Gestaltung der GUI genutzt.



Die Bibliothek besteht aus fünf Dateien :

- Caesar.java
- Vigenere.java
- RSA.java
- RSAKey.java
- Algorithmen.java

In den Dateien Caesar.java, Vigenere.java und RSA.java wurden die Ver- und Entschlüsselungsmethoden so wie gegebenenfalls benötigte Hilfsmethoden implementiert.

Die Datei RSAKey.java beschreibt eine Klasse, welche den Umgang mit den oben genannten RSA-Schlüsseln einfacher realisieren lässt.

Die GUI wurde mit Hilfe von zwei Dateien erstellt :

- GUI.java
- GUI.fxml

Die Datei GUI.fxml beschreibt anhand einer auf XML-basierenden Sprache das Aussehen der GUI. Die Funktionalität der GUI wurde innerhalb der Datei GUI.java unter Hilfe der Bibliothek implementiert.

Der Quelltext(Sourcecode) befindet sich in den Anlagen.

6.Fazit

Zusammenfassend kann man sagen, dass in dieser Komplexen Leistung einige der für die Verschlüsselung relevanten mathematischen Grundlagen gezeigt wurden, so zum Beispiel die eulersche ϕ-Funktion oder das multiplikative Inverse bezüglich mod n. Diese beiden ausgewählten Verfahren spielen in der höheren Mathematik außerdem eine wichtige Rolle.
Es wurde gezeigt welche Arten der Verschlüsselung es gibt, wie sie funktionieren und welche Vor- und Nachteile es gibt. Dieser Teil meiner Komplexen Leistung ist insbesondere für Laien sehr wichtig, das sie ohne das dort vermittelte Wissen, die weiteren Schritte schwer bis überhaupt nicht verstehen. Deswegen befindet sich dieser Teil auch relativ am Anfang.
Nach der Systematisierung folgt die Vorstellung zweier sehr unterschiedlicher Verfahren, zum einen das Vigenère-Verfahren und zum anderen die RSA-Verschlüsselung.
Darauf folgend habe ich meine Eigenleistung erklärt und gezeigt welchen Nutzen diese bringt und wie sie implementiert wurde.
Bei dem Thema Eigenleistung ist auch noch zu erwähnen, dass es in der Datei „RSA.java“ noch einen Bug (Ein Bug bezeichnet in der Programmier-Szene einen Programmfehler) gibt, welcher aber im Rahmen der Komplexen Leistung nicht zu lösen ist, da diese Fehlerbehebung zu komplex für diese Arbeit werden würde. Wenn man diese Eigenleistung erweitern und den Bug beheben würde, würde ich meiner Meinung nach auch von der Sprache Java auf die multiparadigmen Sprache „Scala“ setzen, welche in vielen Bereichen das Programmieren auf den ersten Blick verkompliziert, aber für den geübten Programmierer das eigentliche Programmieren vereinfacht, zum Beispiel dadurch dass alles ein Objekt ist, eine Typinteferenz trotz statischer Typisierung existiert, oder man die implizite Konvertierung nutzen kann.

Abschließend möchte ich sagen, dass das gesamte Thema der Verschlüsselung mit der Zeit immer größer und komplexer wird,da die benötigte Sicherheit in aller Welt gebraucht wird.
Diese Arbeit ist meiner Meinung nach ein guter Ansatz für eine noch komplexere und größere Ausarbeitung, zum Beispiel im Rahmen einer Bachelor, - beziehungsweise Masterarbeit oder einer Promotion.Dies würde aber über den Rahmen meiner Komplexen Leistung weit hinausgehen.

Am Ende dieser Arbeit bedanke ich mich vielmals für das aufmerksame Lesen meiner Komplexen Leistung und bei meinem Mentor Herrn Fritz für seine gute Beratung und seine hilfreichen Hinweise. Ich hoffe, dass meine Ausführungen Interesse geweckt haben und bei den Lesern zu einer Bereicherung ihres Wissens beigetragen haben.

7.Anhang

7.1 Literaturverzeichnis / Quellen

Literaturverzeichnis :

Anlage 1 : Wikipedia (26.05.2017)
https://de.wikipedia.org/wiki/Teilerfremdheit

Anlage 2 : Wikipedia (05.07.2017)

https://de.wikipedia.org/wiki/Symmetrisches_Kryptosystem

Anlage 3 & 4: Wikipedia (24.07.2017)

https://de.wikipedia.org/wiki/Vigen%C3%A8re-Chiffre

Quellen:

- „Einführung in die Zahlentheorie“ von Peter Bundschuh (ISBN : 9783540764908)
- „Programmieren lernen mit Java“ von Dipl.-Ing. Hans-Peter Habelitz (ISBN : 9783836228626)
- http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/formal_modulo_de
- https://de.wikipedia.org/wiki/Symmetrisches_Kryptosystem
- https://www.uibk.ac.at/mathematik/personal/pauer/vortragpauerwien2004.pdf
- https://de.wikipedia.org/wiki/Programmfehler
- https://de.wikipedia.org/wiki/Scala_(Programmiersprache)
- http://sebinside.de/2016/10/05/14-coole-scala-features-teil-1-von-3/
- http://sebinside.de/2016/10/05/14-coole-scala-features-teil-2-von-3/
- https://de.wikipedia.org/wiki/Primzahl
- https://www.elektronik-kompendium.de/sites/net/1910141.htm
- https://de.wikipedia.org/wiki/Eulersche_Phi-Funktion

Komplexe Leistung Moritz Enge 11CH1

- http://www.kryptowissen.de/geschichte-der-kryptographie.html
- http://www.kryptowissen.de/enigma.html
- https://de.wikipedia.org/wiki/Alan_Turing
- http://www.iti.fh-flensburg.de/lang/krypto/grund/inverses-element.htm
- http://www.kryptowissen.de/asymmetrische-verschluesselung.html
- https://de.wikipedia.org/wiki/RSA-Kryptosystem
- https://upload.wikimedia.org/wikipedia/commons/thumb/a/a2/Orange_blue_public_key_cryptography_de.svg/2000px-Orange_blue_public_key_cryptography_de.svg.png
- https://de.wikipedia.org/wiki/Symmetrisches_Kryptosystem#/media/File:Orange_blue_symmetric_cryptography_de.svg
- https://de.wikipedia.org/wiki/Hybride_Verschl%C3%BCsselung#/media/File:Hybride_Verschl%C3%BCsselung.png
- https://de.wikipedia.org/wiki/RSA-Kryptosystem#/media/File:Verschl%C3%BCsselung_(asymmetrisches_Kryptosystem)_Schema.svg
- https://de.wikipedia.org/wiki/RSA-Kryptosystem#/media/File:Entschl%C3%BCsselung_(symmetrisches_und_asymmetrisches_Kryptosystem)_Schema.svg
- https://de.wikipedia.org/wiki/Java_(Programmiersprache)#/media/File:Java-Logo.svg
- https://upload.wikimedia.org/wikipedia/commons/thumb/d/d5/IntelliJ_IDEA_Logo.svg/2000px-IntelliJ_IDEA_Logo.svg.png
- https://upload.wikimedia.org/wikipedia/commons/7/79/Ronald_L_Rivest_photo.jpg
- https://upload.wikimedia.org/wikipedia/commons/1/1e/Adi_Shamir_at_TU_Darmstadt_%282013%29.jpg
- http://www.kryptowissen.de/symmetrische-verschluesselung.html

7.2 Anlagen

Quelltext:

- Algorithmen.java :

```
class Algorithmen{

   private Caesar caesar = new Caesar();  // Cäsar Referenz wird initialisiert
   private Vigenere vigenere = new Vigenere(); // Vigenere Referenz wird initialisiert
   private RSA rsa = new RSA(); // RSA Referenz wird initialisiert

   String Caesar_encrypt(String input, int key){
      return caesar.encrypt(input,key); // Anwendung der encrypt-Methode der Cäsar Klasse
   }

   String Caesar_decrypt(String input, int key){
       return caesar.decrypt(input,key); // Anwendung der decrypt-Methode der Cäsar Klasse
   }

   String Vigenere_encrypt(String input, String key){
     return vigenere.encrypt(input,key); // Anwendung der encrypt-Methode der Vigenere Klasse
   }

   String Vigenere_decrypt(String input, String key){
     return vigenere.decrypt(input,key); // Anwendung der decrypt-Methode der Vigenere Klasse
   }

    String RSA_crypt(String input, RSAKey key){
       return rsa.crypt(input,key); // Anwendung der encrypt-Methode der RSA Klasse
    }
}
```

Komplexe Leistung Moritz Enge 11CH1

- Caesar.java :

```
class Caesar {
   String encrypt(String input, int key){ // Methode zur Verschlüsselung
      String ret = "";
      input = input.replaceAll("[^a-zA-Z]","").toUpperCase();
      for( int i = 0; i < input.length(); ++i )
      {
         if(input.charAt(i) == ' ') ret+= input.charAt(i);
            else{
         ret += (char) ((input.charAt(i) - 65 + key) % 26 + 65);
      }
      }
      return ret.toLowerCase();
   }

   String decrypt(String input, int key){ // Methode zur Entschlüsselung
      String ret = "";
      input = input.toUpperCase();
      for( int i = 0; i < input.length(); ++i ) {
         if (input.charAt(i) == ' ') ret += input.charAt(i);
         else {
            ret += (char) ((input.charAt(i) + 65 - key) % 26 + 65);
         }
      }
      return ret.toLowerCase();
   }

}
```

Komplexe Leistung Moritz Enge 11CH1

- Vigenere.java :

```
class Vigenere {

   String encrypt(String input, String key) { // Methode zur Verschlüsselung
      String ret = "";
      input = input.replaceAll("[^a-zA-Z]","").toUpperCase();
      for( int i = 0; i < input.length(); ++i )
      {
         ret += (char) ( ( input.charAt( i ) + 65 - key.charAt(i%key.length()) ) % 26 + 65 );
      }
      return ret.toLowerCase();
   }

   String decrypt(String input, String key) { // Methode zur Entschlüsselung
      String ret = "";
      input = input.replaceAll("[^a-zA-Z]","").toUpperCase();
      for( int i = 0; i < input.length(); ++i )
      {
         ret += (char) ( ( input.charAt( i ) - 65 + key.charAt(i%key.length()) ) % 26 + 65 );
      }
      return ret.toLowerCase();
   }
}
```

Komplexe Leistung Moritz Enge 11CH1

- RSA.java:

```
import java.math.BigInteger;

public class RSA {

    private long modinv(long m, long mod){  //Methode zur Berechnung des modularen inversen einer Zahl
        BigInteger n = new BigInteger(String.valueOf(m));
        BigInteger mo = new BigInteger(String.valueOf(mod));
        return n.modInverse(mo).longValueExact();
    }

    private long modpow(long i, long x, long n) { // Methode zur Berechnung des Modolus einer Potenz
        BigInteger imp = new BigInteger(String.valueOf(i));
        BigInteger xmp = new BigInteger(String.valueOf(x));
        BigInteger nmp = new BigInteger(String.valueOf(n));

        return imp.modPow(xmp,nmp).longValueExact();
    }

    RSAKey[] gen(long p, long q , long e){ // Methode zur Erzeugung der RSA-Schlüssel

        RSAKey pub = new RSAKey(e, p * q);
        RSAKey pri = new RSAKey(modinv(e,(p-1)*(q-1)),p*q);
        return new RSAKey[]{pub,pri};
    }

    String crypt(String message, RSAKey key){ // Methode zur Ver- / Entschlüsselung
        String output = "";

        for(char i : message.toCharArray()){
            output += (char) (modpow(i,key.x,key.n)% 128);
        }

        return output;
    }
}
```

- RSAKey.java :

```
public class RSAKey { // Klasse zum besseren Umgang mit RSA-Schlüsseln
    long x = 0;
    long n = 0;
    RSAKey(long x , long n){
        this.x = x;
        this.n = n;
    }
}
```

- GUI.java :

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

import javax.swing.*;
import java.util.Objects;

public class GUI extends Application{

    @Override
    public void start(Stage stage) throws Exception {

        // Algorithmen Bibliothek wird initalisiert
        Algorithmen alg = new Algorithmen();

        Parent root = FXMLLoader.load(getClass().getResource("guikl.fxml")); // Design der GUI wird
geladen
        stage.setResizable(false);  // Das Fenster kann nicht durch den Benutzer in der Größe verändert
werden
        stage.setTitle("Verschlüsselungverfahren"); // Titel wird gesetzt
        Scene scene = new Scene(root); // Scene wird initalisiert


        // Eingabefelder und Knöpfe werden initalisiert und Einstellungen werden getroffen
        TextArea txt_c = (TextArea) root.lookup("#ct1");
        TextArea txt2_c = (TextArea) root.lookup("#ct3");
        txt2_c.setEditable(false);  // das Ausgabefeld kann nich bearbeitet werden
        TextField key_c = (TextField) root.lookup("#ct2");
        Button btnv = (Button) root.lookup("#btnv");
        Button btne = (Button) root.lookup("#btne");

        // Eingabefelder und Knöpfe werden initalisiert und Einstellungen werden getroffen
        TextArea txt_v = (TextArea) root.lookup("#vt1");
        TextArea txt2_v = (TextArea) root.lookup("#vt3");
        txt2_v.setEditable(false);  // das Ausgabefeld kann nich bearbeitet werden
        TextField key_v = (TextField) root.lookup("#vt2");
        Button btnv2 = (Button) root.lookup("#btnv2");
        Button btne2 = (Button) root.lookup("#btne2");


        // Eingabefelder und Knöpfe werden initalisiert und Einstellungen werden getroffen
        TextArea txt_r = (TextArea) root.lookup("#rt1");
        TextArea txt2_r = (TextArea) root.lookup("#rt4");
        txt2_r.setEditable(false); // das Ausgabefeld kann nich bearbeitet werden
        TextField key_r = (TextField) root.lookup("#rt2");
        TextField key2_r = (TextField) root.lookup("#rt3");
        Button btnv3 = (Button) root.lookup("#btnv3");
        Button btne3 = (Button) root.lookup("#btne3");
```

```
//Aktion des Verschlüsselungsknopfes der Cäsar-Chiffre wird mittels Lamda Befehl definiert
    btnv.setOnAction(e -> {
        Integer key = Integer.parseInt(key_c.getText());
        if(key <= 0 || key > 25){ //Prüfung ob sich der "key" zwischen 0 und 25 befindet
            JOptionPane.showMessageDialog(null,"Bitte geben sie eine Zahl zwischen 1 und 25 ein");
            //Gegebenenfalls wird eine Fehlermeldung ausgegeben
        }
        else{
            txt2_c.setText( // Der Text des Ausgabefeldes wird gesetzt
                alg.Caesar_encrypt(txt_c.getText().toLowerCase(),key)); // Methode wird angewandt
        }
    });

    //Aktion des Entschlüsselungsknopfes der Cäsar-Chiffre wird mittels Lamda Befehl definiert
    btne.setOnAction(e -> {
        Integer key = Integer.parseInt(key_c.getText());
        if((key <= 0 || key > 25) && Objects.equals(txt_c.getText(), "")){ //Prüfung ob sich der "key"
                                                    // zwischen 0 und 25 befindet und ob das Feld leer ist
            JOptionPane.showMessageDialog(null,"Bitte geben sie eine Zahl zwischen 1 und 25 ein");
            //Gegebenenfalls wird eine Fehlermeldung ausgegeben
        }
        else{
            txt2_c.setText( // Der Text des Ausgabefeldes wird gesetzt
                alg.Caesar_decrypt(txt_c.getText().toLowerCase(),key)); // Methode wird angewandt
        }
    });


    //Aktion des Verschlüsselungsknopfes der Vigenerre-Chiffre wird mittels Lamda Befehl definiert
    btnv2.setOnAction(e -> {
        // Es wird geprüft ob das Eingabefeld des Schlüssels leer ist
        if(!txt_v.getText().isEmpty()) {
            txt2_v.setText(alg.Vigenere_encrypt(txt_v.getText().toLowerCase(), key_v.getText())); //
Methode wird angewandt
        }
    });


    //Aktion des Entschlüsselungsknopfes der Vigenerre-Chiffre wird mittels Lamda Befehl definiert
    btne2.setOnAction(e -> {
        if(!txt_v.getText().isEmpty()) { // Es wird geprüft ob das Eingabefeld des Schlüssels leer ist
            txt2_v.setText(alg.Vigenere_decrypt(txt_v.getText().toLowerCase(), key_v.getText())); //
Methode wird angewandt
        }
    });

    //Aktion des Entschlüsselungsknopfes der RSA-Verschlüsselung wird mittel Lamda Befehl definiert
    btnv3.setOnAction( e -> {
        if(!(key_r.getText().isEmpty() && key2_r.getText().isEmpty())) { // Prüfung ob die Eingabefelder leer
sind
            long p = Integer.parseInt(key_r.getText()); // Variable p wird aus dem Eingabefeld geholt
            long q = Integer.parseInt(key2_r.getText()); // Variable q wird aus dem Eingabefeld geholt
            RSAKey key = new RSA().gen(p, q, 65537)[0]; // Öffentlicher Schlüssel wird generiert ; 65537
ist eine gerne verwendete Zahl
```

```
            txt2_r.setText(alg.RSA_crypt(txt_r.getText(), key)); // Methode wird angewandt
        }
    });
    btne3.setOnAction(e -> {
        if(!(key_r.getText().isEmpty() && key2_r.getText().isEmpty())) { // Prüfung ob die Eingabefelder leer
sind
            long p = Integer.parseInt(key_r.getText());// Variable p wird aus dem Eingabefeld geholt
            long q = Integer.parseInt(key2_r.getText());// Variable q wird aus dem Eingabefeld geholt
            RSAKey key = new RSA().gen(p, q, 65537)[1];// Privater Schlüssel wird generiert ; 65537 ist
eine gerne verwendete Zahl
            txt2_r.setText(alg.RSA_crypt(txt_r.getText(), key)); // Methode wird angewandt
        }
    });


    // Das Fenster wird "fertig" gemacht und erscheint
    stage.setScene(scene);
    stage.show();
  }


  /**
   * @param args the command line arguments
   */

  // Das Programm wird ausgeführt
  public static void main(String[] args) {
    launch(args);
  }
}
```

7.3 Selbstständigkeitserklärung

Selbstständigkeitserklärung

Hiermit versichere ich , Moritz Enge, dass ich diese Arbeit selbstständig angefertigt, nur die angegebenen Hilfsmittel verwendet und alle Stellen,die dem Wortlaut oder nach anderen Werken entnommen sind, durch Angabe der Quellen als Übernahmen kenntlich gemacht habe.

Wechselburg, der 24.Oktober 2017