Entschlüsselungsprogramm - Decryptor

Ein Projekt von Ben Siebert

Kurzfassung

Das Ziel meiner Arbeit war es, eine Möglichkeit zu schaffen, einen Text am Computer ganz einfach durch verschiedenste Verschlüsselungen zu ver- und natürlich auch zu entschlüsseln und sich so auch mit den unterschiedlichen Vor- und Nachteilen von Verschlüsselungsmethoden zu beschäftigen.

Dazu habe ich ein Computerprogramm geschrieben, welches bisher 3 Verschlüsselungen unterstützt. Ich habe hierzu mein Java-Wissen ausgebaut und dies als Programmiersprache genutzt. Durch die Arbeit an diesem Projekt habe ich viel über objektorientierte Programmierung gelernt.

Zentrale Aspekte meines Projektes:

- Eine verständliche Oberfläche (später auch GUI genannt) für den Anwender zu schaffen
- Ein möglichst großes Spektrum von Verschlüsselungen (der aktuelle Projektstand ist noch ausbaufähig)
- Die Bereitstellung hoch performanter Verschlüsselungen
- Entwicklung von eigenen Verschlüsselungsmethoden

Und natürlich die performante Zusammenarbeit der oben genannten Komponenten.

Inhaltsverzeichnis

1. Einleitung

1.1 Ideenfindung & Lösungsansatz

Anfang 2019 waren wir im Urlaub in München, dort haben wir das Deutsche Museum besucht, in dem ja auch die ENIGMA (die berühmte Verschlüsselungsmaschine aus dem zweiten Weltkrieg) zu besichtigen ist. Das fand ich spannend. Dort wurde auch erklärt, dass Verschlüsselung im Allgemeinen in unserem Alltag sehr wichtig ist, da wir insbesondere im Internet ganz viele Daten verschicken, wie z.B. Kontodaten oder Passwörter. Hier ist es wichtig, dass diese Daten sicher sind.

In dem Museum wurden auch noch andere Verschlüsselungsmethoden erklärt und ich wollte versuchen, so etwas selbst nach zu stellen.

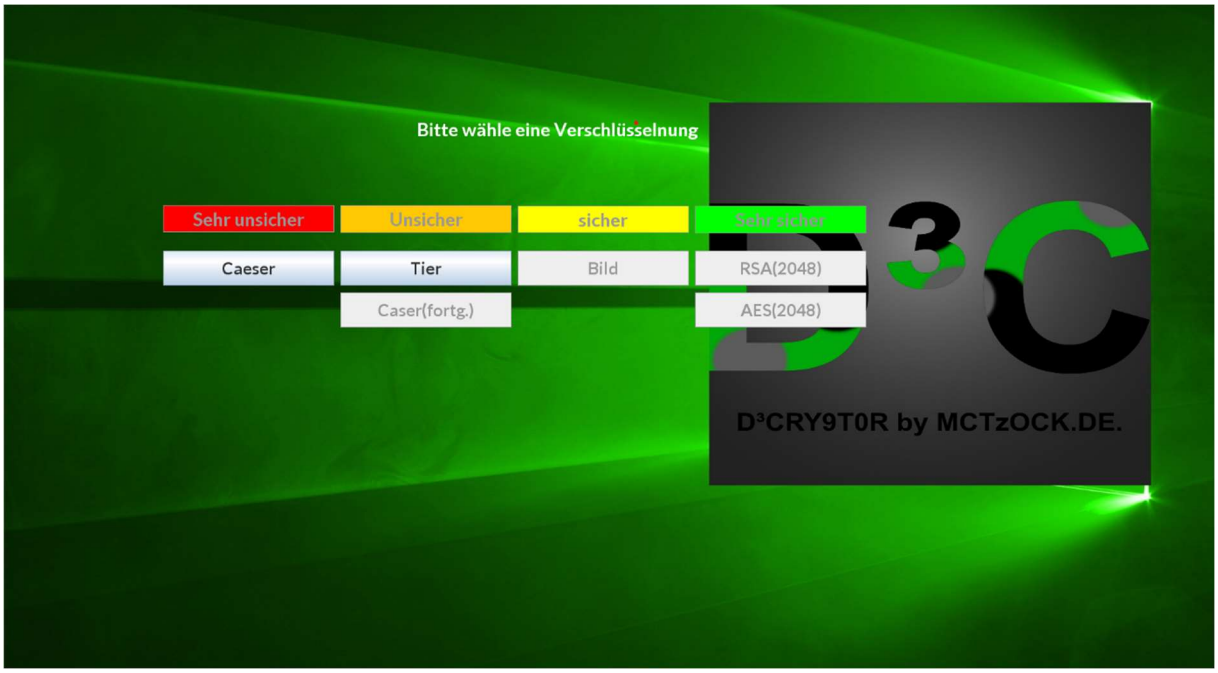
Für mich war schnell klar, dass ich das mit Hilfe eines Computers versuchen wollte, und daher stand auch fest, dass ich meine Grundkenntnisse in der Programmierung verbessern wollte. Nach einiger Recherche im Internet und vielen YouTube-Tutorials zu Java wollte ich versuchen, meine Ideen mit Java zu verwirklichen und dafür als Programmierumgebung Eclipse zu benutzen.

2. Vorgehensweise & Verschlüsselungsmethoden

2.1 Entwicklung der Oberfläche

Für mich war es wichtig, dass die Oberfläche einfach und ansprechend ist. Außerdem sollte sie modular aufgebaut sein, so dass ich sozusagen einen Rahmen für alle möglichen Verschlüsselungsmodule baue, und der Nutzer von diesem Startpunkt aus selber entscheiden kann, welche Verschlüsselungsmethode er ausprobieren möchte. Ich habe sie mithilfe von Window-Builder (einem Eclipse Plugin) umgesetzt. Immer wenn ich eine neue Verschlüsselung fertigstelle, kann ich sie einfach über einen zusätzlichen Button dem Nutzer zur Verfügung stellen. Der Hintergrund der Oberfläche ist auch individuell anpassbar, indem man das entsprechende Bild einfach austauscht (es befindet sich unter : „/assets/bin/fs.jpg/“)

Meine aktuelle Version sieht so aus (der Hintergrund ist ein Logo, was ich mir für den Decryptor ausgedacht habe).



2.2 Verschiedene Verschlüsselungen

Das Herz des Programms sind die einzelnen Verschlüsselungsmodule. Hier habe ich mich zuerst gefragt, welche es wohl gibt und ob ich wohl auch eine eigene entwickeln kann. Einen guten ersten Einblick in die unterschiedlichen Verschlüsselungs-methoden gibt unter anderem ein YouTube-Video von „Brickscience-TV“, in dem mit Hilfe von Lego-Animationen die Kryptographie, also die „Lehre der Verschlüsselung“, sehr anschaulich erklärt wird.

Dort wurde als erste Verschlüsselungsmethode überhaupt die sogenannte CAESAR-Verschlüsselung genannt. Die wollte ich als erstes nachbauen, auch um zu verstehen, wie ich das Ver- und Entschlüsseln in Java überhaupt umsetzen kann.

2.2.1 Caesar-Verschlüsselung

Die Caesar Verschlüsselung wurde von Julius Caesar erfunden, es handelt sich um eine reine Verschiebungsmethode; d.h. wenn z.B. eine Verschiebung um 2 Stellen erfolgen soll, dann wird ein „A“ aus dem Originaltext zu einem „C“ im verschlüsselten Text.

In meiner Programmierung habe ich die Verschiebung um 2 Stellen gemacht, d.h., mein Geheim-Alphabet in der Java Programmierung sieht wie folgt aus:

Original	A	B	C	D	…	Y	Z
Verschlüsselung	C	D	E	F	…	A	B

In Java habe ich das realisiert, in dem folgende Schritte passieren:

- Der Nutzer gibt einen Text ein
- Das Programm fragt, ob der Text ver- oder entschlüsselt werden soll
- Dann wird der Nutzer auf den Auswahlbildschirm geführt, und wenn er die Caesar-Verschlüsselung auswählt, wird der Text im Programm in die „CAESAR-Klasse“ weitergeben
- Unten ist das wesentliche Programmmodul abgebildet. In den beiden FOR-Schleifen wird der gesamte Text durchsucht und für jeden Buchstaben wird dessen Position im Alphabet ermittelt. Diese Position wird im CHAR-ARRAY

ver gesucht und der Buchstabe, der sich dort befindet, wird in den Ausgabetext geschrieben.

- Nach dem er den gesamten Text durchlaufen hat, gibt er den verschlüsselten Text (CHAR-ARRAY ***aus***) an den Nutzer aus.
-

```
long startTime = System.currentTimeMillis();
String eingabe = text;
 char[] eingabe_c = eingabe.toCharArray();
 char[] alp = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j
 char[] ver = {'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l
 char[] aus = new char[eingabe_c.length];
 for(int i = 0; i < eingabe_c.length; i++)
 {
     for(int j = 0; j < alp.length; j++)
     {
         System.out.println(i);
         System.out.println(j);
         if(eingabe_c[i] == alp[j])
         {
             aus[i] = ver[j];
         }
     }
 }
 String str = String.copyValueOf(aus);
 long stopTime = System.currentTimeMillis();
 long elapsedTime = stopTime - startTime;
 System.out.println("Time: " + elapsedTime / 1000 + "s");
 System.out.println(str);
 Result.show(str);
```

Die Verschlüsselung ist aber sehr unsicher, und dass aus zwei Gründen:

1. Ein Buchstabe wird immer mit dem gleichen Buchstaben verschlüsselt, d.h. ein „A“ ist hier immer ein „C“. Da in einem Text bestimmte Buchstaben besonders oft vorkommen (z.B. Vokale), kann man vielleicht schon an der Häufigkeit der Buchstaben Rückschlüsse auf den Original-Buchstaben machen.
2. Die Reihenfolge in der Verschlüsselung ist gleich der Reihenfolge der Buchstaben im Alphabet, d.h. wenn man einen Buchstaben herausgefunden hat, dann kann man sich alle anderen herleiten.

Nachdem ich diese Verschlüsselungsmethode erfolgreich in Java programmiert hatte, wollte ich in einem zweiten Schritt dem Nutzer etwas mehr Auswahl geben. Gleichzeitig habe ich gemerkt, dass die Methode, die ich hier benutzt habe, sehr langsam ist, dies wollte ich noch verbessern.

2.2.2 Caesar – Weiterentwicklung „Fast Caesar“

Dies ist eine Weiterentwicklung der originalen Caesar Verschlüsselung. Sie ist wesentlich schneller und man kann die Anzahl, um die das Alphabet verschoben wird, individuell anpassen (0-56). 56 Stellen wegen der 56 Zeichen, die verschlüsselt werden, da hier Groß- und Kleinschreibung unterschieden wird. Sie ist schneller, weil ich mittlerweile die Programmierweise angepasst habe. Sie geht jetzt nicht mehr wie in der klassischen Caesar Verschlüsselung jeden Buchstaben durch, sondern sucht im Text gezielt nach Buchstaben.

Diesmal sieht die Verschlüsselung wie folgt aus:

```
public void encrypt(String text)
{
    System.out.println(text);
    for(Integer i = 0; i < text.length(); i++)
    {
        itext.add(text.toCharArray()[i]);
        System.err.println(itext.get(i));
    }
    for(Integer i = 0; i < itext.size(); i++)
    {
        int itext_pos = i;
        int alp_pos = getPosition(itext.get(itext_pos));
        System.out.println(alp.get(alp_pos));
        itext.set(i, ver.get(alp_pos));
    }
    System.out.println(itext);
    String otext = itext.toString();
    System.out.println(otext);
    //to-string
    otext = "";
    for(int i = 0; i < itext.size(); i++)
    {
        otext = otext + itext.get(i);
    }
    System.out.println(otext);
}
```

Folgende Schritte passieren:

- Der Nutzer gibt wieder einen Text ein
- Das Programm fragt, ob der Text ver- oder entschlüsselt werden soll
- Dann wird der Nutzer auf den Auswahlbildschirm geführt, und wenn er die Caesar-Verschlüsselung auswählt, wird der Text im Programm in die „PERF-CAESAR-Klasse“ weitergeben
- Oben ist das wesentliche Programmmodul abgebildet.
- In den drei FOR-Schleifen wird in dem eingegebenem Text (***itext***) nach speziellen Buchstaben gesucht, und diese werden anschließend in den Ausgabetext (***otext***) verschlüsselt eingetragen.
- Damit ist gemeint, dass es erst nach allen „a“ sucht und diese ersetzt, dann nach allen „b“ und so weiter. Daher muss er insgesamt nur 56-mal (Groß- und Kleinschreibung) suchen, egal wie lang der Text ist.

Diese neue Programmier-Methode macht das Programm deutlich schneller, was insbesondere bei längeren Texten ins Gewicht fällt.

Grundsätzlich ist diese Verschlüsselung natürlich immer noch genauso unsicher wie die normale Caesar-Verschlüsselung. Allerdings kann man durch die unterschiedliche Verschiebung eine größere Abwechslung schaffen. So kann man, wenn man z.B. mehrere Texte verschlüsseln will, es so etwas schwieriger machen, über die Häufigkeitsverteilung der Buchstaben alle Texte zu entschlüsseln.

2.2.3 Eigenentwicklung - Tier-Verschlüsselung

Diese Verschlüsselung habe ich mir selbst ausgedacht und so wie alles selber umgesetzt. Hier wollte ich die Sicherheit erhöhen, indem ich die Reihenfolge der Buchstaben sehr variabel gestalte und zusätzlich über ein „Passwort“ die Möglichkeit gebe, zwischen unterschiedlichen, bis jetzt 64 verschiedenen Verschlüsselungs-Codes auszuwählen.

Das Passwort wollte ich möglichst anschaulich gestalten, so dass der Nutzer es sich gut merken kann, und es für einen Außenstehender auch unverständlich (nämlich absurd) ist. Deshalb habe ich mir Tiere ausgedacht, die verschiedene Kombinationen von Kopfbedeckungen, Schuhen und Farben haben können. Also z.B. ein blauer Löwe mit einem Zylinder und Sandalen.

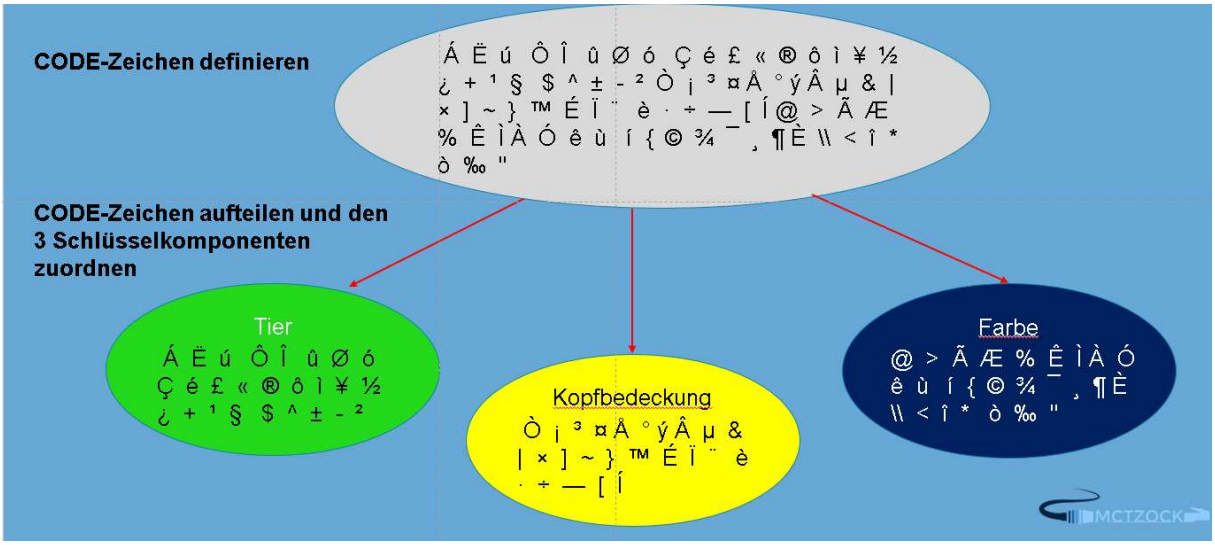
Die Funktionsweise ist wie folgt:

Nachdem der Nutzer einen Text eingegeben hat, muss er sich ein Tier aus den folgenden Möglichkeiten auswählen:

Komponente	Auswahl 1	Auswahl 2	Auswahl 3	Auswahl 4
Tier	Löwe	Pinguin	Giraffe	Hase
Farbe	Schwarz	Blau	Lila	Gelb
Kopfbedeckung	Zylinder	Kappe		
Schuhe	Turnschuhe	Sandalen		

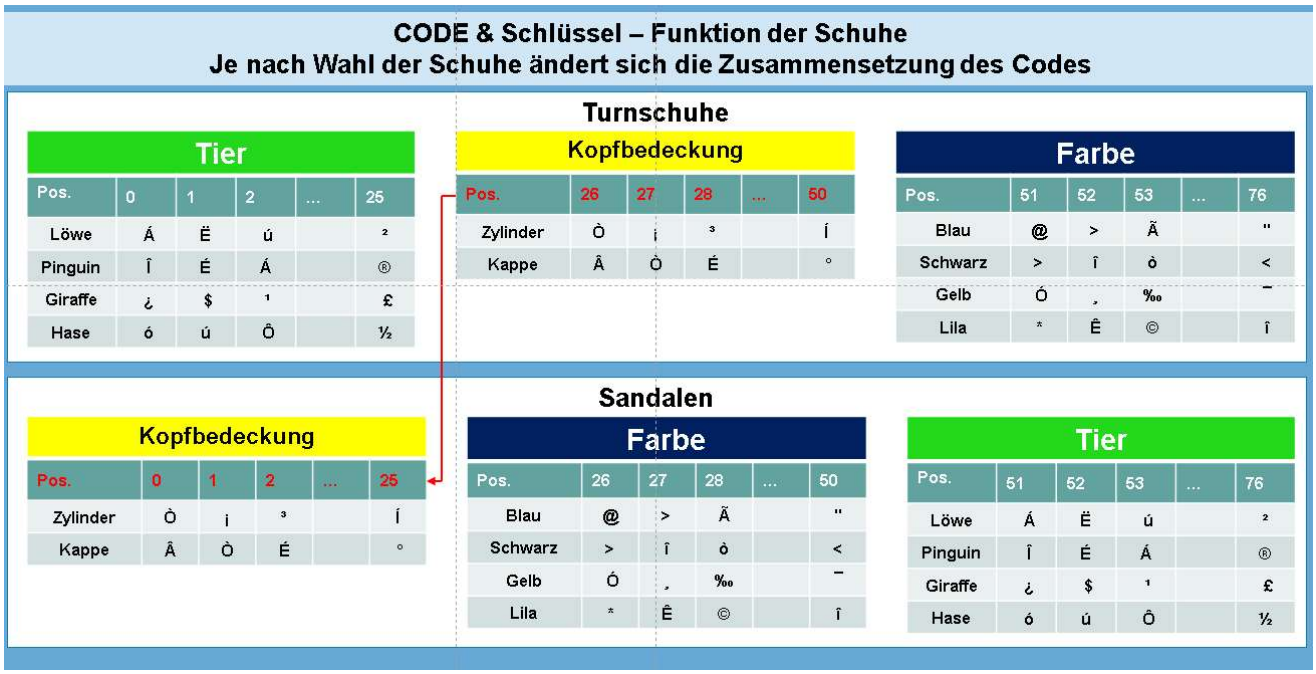
Er erhält dann als zusätzliche Merkhilfe ein Tier-Bild.

Was machen jetzt die einzelnen Komponenten? Zuerst werden die Code-Zeichen in drei verschiedene „Töpfe“ geteilt.



Für jedes Auswahlmöglichkeit wird die Reihenfolge der Verschlüsselungszeichen geändert (siehe im folgenden Bild z.B. die Reihenfolge bei den Tieren).

Danach entscheidet die vierte Auswahl, nämlich die Schuhe über die Reihenfolge des Codes:



Damit werden 64 individuelle CODES erstellt, mit dem der Text verschlüsselt werden kann.

Die Vorgehensweise ist wie folgt:

- Der Nutzer gibt einen Text ein
- Das Programm fragt, ob der Text ver- oder entschlüsselt werden soll
- Dann wird der Nutzer auf den Auswahlbildschirm geführt, und wenn er die Caesar-Verschlüsselung auswählt, wird der Text im Programm in die „TIER-Klasse“ weitergeben
- Als erstes wird der Text einmal „umgedreht“ (d.h. der erste Buchstabe wird der letzte Buchstabe und umgekehrt), damit er selbst wenn der Code geknackt wird, noch nicht direkt auf den ersten Blick lesbar ist.
- Die eigentliche Verschlüsselung innerhalb der Tier-Methode funktioniert genauso wie bei der FAST-CAESAR-Verschlüsselung, lediglich die CHAR-ARRAYS der einzelnen CODES sind natürlich anders. Damit ist die Tier-Verschlüsselung auch sehr performant.

Im Moment werden 76 Zeichen unterstützt (Groß-/Kleinschreibung, Ziffern und wesentliche Satzzeichen). Hier könnte man noch eine zusätzliche Erweiterung um weitere Zeichen machen, und so die Variabilität noch erhöhen. Zusätzlich könnte diese Methode noch mit eigenen Tierkombinationen erweitert werden.

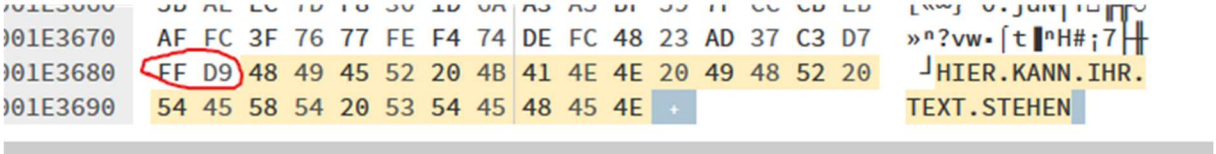
Der Vorteil der Tier-Verschlüsselung im Vergleich zur Caesar-Verschlüsselung ist, dass hier zwei Verschlüsselungsschritte passieren; zuerst das Umdrehen des Original-Textes und dann die Verschlüsselung mit einem der 64 Codes. Zusätzlich ist sie sicherer, weil die Reihenfolge der Verschlüsselungszeichen nicht vorhersehbar ist, und sich auch je nach gewähltem Tier ändert.

2.2.4 Eigenentwicklung – Bild-Verschlüsselung

Diese Verschlüsselungsmethode habe zum jetzigen Zeitpunkt noch nicht fertig programmiert, aber die Idee ist es hier, den verschlüsselten Text in einem Bild „zu verstecken“.

Diese Verschlüsselung verschlüsselt den Text zuerst mit der Fast Caeser Verschlüsselung (oder jeder anderen Verschlüsselung) und schreibt dann den Hex Code des verschlüsselten Textes in den HEX-CODE eines Bildes (im jpg-Format). Die Besonderheit ist, dass der HEX-CODE einer jpg-Datei immer mit einer speziellen Zeichenfolge endet, und ein Bildbearbeitungsprogramm alle Zeichen, die danach kommen, nicht liest. Damit kann man hinter dieses Zeichen („FF D9“) den verschlüsselten Text verstecken, und nur, wer sich die jpg-Datei über einen HEX-CODE Editor ansieht, kann den verschlüsselten Text sehen, und ihn dann wieder mit dem Programm entschlüsseln.

Hier ein Beispiel für den versteckten Text im HEX-CODE:



Dann könnte das Bild einfach verschickt werden, und wer nicht um die versteckte Nachricht weiß, für den ist es einfach ein Bild.

Hier arbeite ich aktuell noch am Einlesen der HEX-Daten der jpg-Datei und der Übergabe der Daten.

3. Zusammenfassung

Das Projekt hat mir viel Spaß gemacht, und ich habe sehr viel über Programmierung gelernt. Im Moment versuche ich noch, auch die Bild-Verschlüsselung fertig zu programmieren, und diese noch zu integrieren.

Ich habe festgestellt, dass die aktuell sicherste Verschlüsselung die sogenannte RSA-Verschlüsselung ist, für die es sogar schon eine spezielle Methode in Java gibt (daher könnte man sie hier auch noch einbauen). Aber ich wollte erst einmal versuchen und verstehen, wie eine Verschlüsselungsmethode tatsächlich selber programmiert werden kann.

Die Tierverschlüsselung ist aus meiner Sicht eine gute Weiterentwicklung der Caesar-Verschlüsselung, auch weil sie sicherer ist und durch die Anschaulichkeit der Tiere sehr einfach zu benutzen ist.

Was mir wichtig war, ist, dass das Programm insgesamt gut und einfach zu bedienen ist, ohne dass man weitere Programmierkenntnisse haben muss. Das habe ich schon mit einigen Mitschülern ausprobiert, und ich denke, das Programm ist auch für jüngere Nutzer zu gebrauchen.

Ich werde auch zukünftig versuchen, noch weitere Verschlüsselungsmethoden einzufügen, um das Programm zu erweitern.

4. Quellen

Als Verweis auf die Enigma im deutschen Museum
https://www.deutsches-museum.de/sammlungen/meisterwerke/meisterwerke-ii/enigma/

Als Quelle für zahlreiche Java-Tutorials :
https://www.youtube.com/

Als Quelle zur anschaulichen Einführung in die Kryptographie:
YouTube Video: Brickscience TV - Kryptographie ("Fast Forward Science 2017").
(https://youtu.be/QhwcD4XHHJ8)

Als HEX-Editor: https://hexed.it/

Source-Code des Programms (noch nicht vollständig):
https://decryptor.page.link/github

5. Zusatz

5.1 Eine Erklärung der API

Der Decryptor bietet natürlich auch für Software Entwickler viele Möglichkeiten. Mit der eigenen Java API können jegliche Verschlüsselungen genutzt werden. Die API steht zum Download bereit (siehe Quellen). Im Kapitel 5.2.1 wird erklärt, wie man die API sinnvoll und richtig verwenden kann. Doch wofür steht die Abkürzung API eigentlich? API ist die Abkürzung für das englische Wort „Application programming Interface“ (zu Deutch „Programmierschnittstelle“). Eine API kann man sich wie eine Bibliothek mit vielen dicken Büchern vorstellen, in denen die Lösungen für Probleme bzw. die vereinfachte Darstellung von Lösungswegen stehen. Diese Bücher können genutzt werden um ein vorhandenes Problem auf clevere Art und Weise einfach zu lösen. Das besondere an APIs ist, dass sie in **jeder** Programmiersprache vorhanden sind. Ein Software Entwickler kann sich also aussuchen für welche Sprache er seine API schreiben möchte.

5.1.1 Die API richtig verwenden

Zunächst muss man um die API nutzen zu können sie aus dem Internet herunterladen (siehe Quellen). Danach ist es notwendig die heruntergeladene .jar Datei in das Projekt zu importieren (siehe 5.2.2). Die API besteht aus zwei Klassen: „Animal_compile.class“ und „Caeser.class“. Die „Animal_compile.class“ Klasse wird dazu verwendet, die Tier Verschlüsselung (siehe Seite 2) nutzen zu können. Die Klasse „Caeser.class“ wird dazu benutzt, die Caeser Verschlüsselung (siehe Seite 2) zu nutzen. Im Folgenden wird zu jeder Klasse ein Beispiel gemacht.

Tier Verschlüsselung:

```
package de.ihr.package;

import de.mctzock.*;

public class IhreKlasse {

    public static void main(String[] args)
    {
        String ihrEingegebenerText = "Das ist ein Text!";
        String ihrVerschlüsselterText = Animal_compile.compile(ihrEingegebenerText, "Löwe.Kappe.Turnschuhe.Blau" /**Ihr Tier**/);
        String ihrEntschlüsselterText = Animal_compile.decompile(ihrVerschlüsselterText, "Löwe.Kappe.Turnschuhe.Blau" /**Gleiches Tier**/);
        System.out.println("Verschlüsselt: " + ihrVerschlüsselterText);
        System.out.println("Entschlüsselt: " + ihrEntschlüsselterText);
    }
}
```

Die Klasse Animal_compile:

Diese Klasse beinhaltet zwei Methoden: compile und decompile! Mit der Compile-Methode kann man einen Text verschlüsseln. Mit der Decompile-Methode kann man einen Text verschlüsseln. (Wobei zu beachten ist, dass man das richtige Tier eingibt. Sonst gibt es einen Fehler!)

Caeser Verschlüsselung:

```
package de.ihr.package;

import de.mctzock.*;

public class IhreKlasse {

    public static void main(String[] args)
    {
        String ihrEingegebenerText = "Das ist ein Text!";
        String ihrVerschlüsselterText = Caeser.enc(ihrEingegebenerText);
        String ihrEntschlüsselterText = Caeser.dec(ihrVerschlüsselterText);
        System.out.println("Verschlüsselt: " + ihrVerschlüsselterText);
        System.out.println("Entschlüsselt: " + ihrEntschlüsselterText);
    }
}
```

Diese Klasse beinhaltet zwei Methoden: enc und dec! Mit der Enc-Methode kann ein Text verschlüsselt werden. Mit der Dec-Methode kann ein Text entschlüsselt werden. (Hierbei ist nichts zu beachten!)

5.1.2 Die API importieren

Eclipse:

1. Rechtsklick auf das Projekt
2. Properties/Einstellungen auswählen
3. Auf die Rubrik Java Build Path klicken
4. Hier Libraries auswählen
5. Um die Datei zu importieren bitte auf „Add External Jar“ klicken und die Dateien auswählen
6. Auf „Apply and Close“ klicken

Apache NetBeans:

1. Properties auswählen
2. Libraries anklicken
3. “Add Jar/folder” auswählen
4. .Jar Datei auswählen

Intellij:

1. Klicke auf „Datei“
2. Dann auf „Projekt Struktur/Project Structure“
3. Nun wähle „Modules“ aus
4. Und dann „Dependencies tab“
5. Klicke auf „+“ und wähle deine .Jar Datei aus