4.2 Schlüsselsicherheit

Eine praktische Anwendung der Berechenbarkeitstheorie ist die Entschlüsselung von verschlüsselten Daten. Historisch betrachtet gibt es verschiedene Vorgehen beim Verschlüsseln und Entschlüsseln von Daten.

Symmetrische Verschlüsselung

Bei symmetrischen Verschlüsselungsverfahren benutzt man den gleichen Schlüssel, um sowohl Verschlüsselung als auch Entschlüsselung durchzuführen. Das berühmteste Beispiel ist die Caesar Chiffre.

Die Verschlüsselung und Entschlüsselung geht hier mit dem gleichen Verfahren, nur dass einmal der Buchstabe nach hinten verschoben wird (Verschlüsselung) und einmal nach vorne verschoben wird (Entschlüsselung).
Dieses System war zu Cäsars Zeit vielleicht ausreichend, da das Verfahren unbekannt war. Heute würde ein Computer die statistische Verteilung der Buchstaben in der Verschlüsselung und einem Klartext überprüfen und daher sofort wissen, welcher Buchstabe das „E“ sein soll (der häufigste Buchstabe des deutschen Alphabets) und damit die Verschiebung erkennen[1] würde.

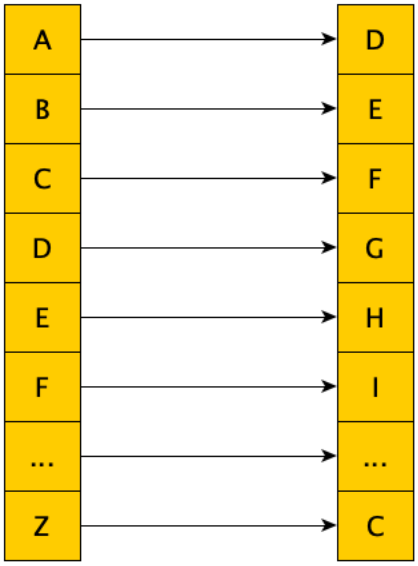

Abbildung 1: Caesar Chiffre

Asymmetrische Verschlüsselung

Bei asymmetrischen Verfahren sind die Schlüssel für das Verschlüsseln und Entschlüsseln unterschiedlich. Außerdem bestehen diese Verfahren auf mathematischen Berechnungen, die durch die Berechnung das Vorkommen gleicher Zeichen verschleiern. Ein Standardalgorithmus, der auch heute noch als sicher gilt ist der RSA Verschlüsselungs-Algorithmus.

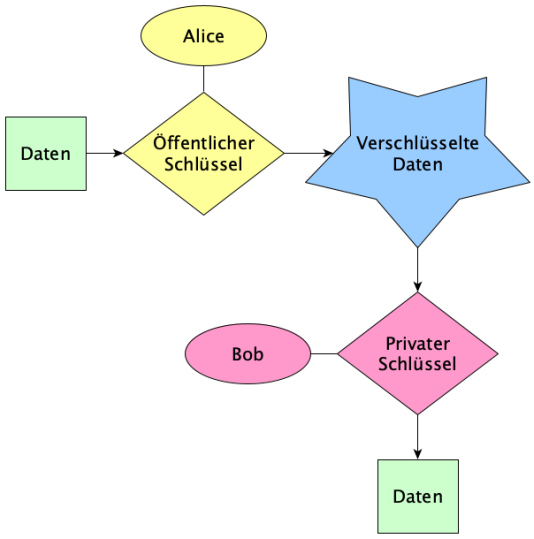

Abbildung 2: Asymmetrisches Verschlüsselungsverfahren

[1] Das gilt übrigens auch für „zufällige“ Austauschalgorithmen z.B. A -> D; B -> Z; … da auch hier der Computer über die statistische Verteilung die Buchstaben schnell erkennt.

Der RSA Algorithmus:

Allgemein	**Beispiel**
1. Wähle zwei Primzahlen p und q die verschieden sind	p = 5 und q = 11
2. Berechne $N = pq$	N = 55
3. Berechne $\varphi(N) = (p-1)(q-1)$	$\varphi(N) = 40$
4. Suche e als teilerfremde Zahl zu $\varphi(N)$ mit $e < \varphi(N)$	e = 7
5. Suche d so, dass gilt: $d \cdot e = 1 \, mod \, \varphi(N)$ (Rest 1 bei Teilen durch $\varphi(N)$) (Achtung d muss positiv sein, also im negativen Fall einfach $\varphi(N)$ addieren.)	Euklidischer Algorithmus: $d \cdot e = 1 \, mod \, (p-1)(q-1)$ $d \cdot 7 = 1 \, mod \, 40$ I: $40 = 5 \cdot 7 + 5$ II: $7 = 1 \cdot 5 + 2$ III: $5 = 2 \cdot 2 + 1$ d.h: III: $1 = 5 - 2 \cdot 2$ II: $2 = 7 - 1 \cdot 5$ I: $5 = 40 - 5 \cdot 7$ III*: II in III: $1 = 5 - 2 \cdot (7 - 1 \cdot 5)$ $1 = 5 - 2 \cdot 7 + 2 \cdot 5$ $1 = 3 \cdot 5 - 2 \cdot 7$ I in III*: $1 = 3 \cdot (40 - 5 \cdot 7) - 2 \cdot 7$ $1 = 3 \cdot 40 - 17 \cdot 7$ Nun d = -17 negativ -> -17+40 = 23 d = 23
6. Verschlüsseln der Zahl m indem man berechnet: $c = m^e mod \, N$	Verschlüsseln der Zahl 8: $c = 8^7 mod \, 55$ $c = 2097152 \, mod \, 55 = 2$
7. Entschlüsseln der Chiffre c = 2 $m = c^d mod \, N$	$m = 2^{23} mod \, 55$ $m = 8388608 \, mod \, 55 = 8$

Beim RSA werden nun p,q und d geheim gehalten, e und N sind öffentlich. Damit kann jeder verschlüsseln, aber nur der Partner weiß die nötigen Informationen für das Entschlüsseln.

Brechen der Verschlüsselungsverfahren

Um beispielsweise das RSA Verfahren zu brechen, muss man die Primzahlen p und q erraten. Normalerweise benutzt man im RSA Verfahren nicht wie wir kleine Primzahlen, sondern möglichst große Primzahlen. Alleine das Berechnen dieser potentiell verwendeten Primzahlen dauert so lange, dass es sich nicht rentiert auf diese Weise den Algorithmus anzugehen. Daher versucht man heutzutage nicht mehr die Verschlüsselten Daten zu entschlüsseln, sondern einfach das Passwort für die verschlüsselten Daten zu erraten.

Der Rechenaufwand für das Erraten eines Passwortes hängt nur von zwei Faktoren ab:

1. Der Länge des Passwortes (L)
2. Der Anzahl der Buchstaben im verwendeten Alphabet für das Passwort (B)

Die Komplexität berechnet sich damit mit der Formel

$$K = B^L$$

Die folgende Tabelle gibt einen Überblick, wie lange ein Computer mit einem handelsüblichen Core-i7[2] mit 3,4 Ghz Kernen braucht, um alle möglichen Passwörter zu erfassen:

Alphabet	Länge	Nötige Berechnungen	Zeit (ca.)
Zahlen (0-9)	10	10^{10}	0,029 Sekunden
	15	10^{15}	49 Minuten
Kleinbuchstaben	10	$26^{10} \approx 1{,}4 \cdot 10^{14}$	7 Minuten
	15	$26^{15} \approx 1{,}6 \cdot 10^{21}$	158,3 Jahre
Klein- und Großbuchstaben	10	$52^{10} \approx 1{,}4 \cdot 10^{17}$	5 Tage
	15	$52^{15} \approx 5{,}5 \cdot 10^{25}$	5 200 000 Jahre
Klein-, Großbuchstaben und Zahlen	10	$62^{10} \approx 8{,}4 \cdot 10^{17}$	29 Tage
	15	$62^{15} \approx 7{,}7 \cdot 10^{26}$	72 600 000 Jahre
Klein-, Großbuchstaben, Zahlen und Sonderzeichen	10	$82^{10} \approx 1{,}4 \cdot 10^{19}$	1,3 Jahre
	15	$82^{15} \approx 5{,}1 \cdot 10^{28}$	4,9 Mrd. Jahre

Man sieht in der Tabelle, dass die Verlängerung des Passwortes die Komplexität des Erratens des Passwortes stark beeinflusst (logischerweise, da es sich um eine Exponentielle Funktion handelt). Wir dürfen gespannt sein, ob sich die Theorie bestätigt, dass die Erfindung der Quantencomputer diese Berechnungen so schnell machen, dass wir uns neue Wege zur Verschlüsselung suchen müssen.

[2] Der Beispielprozessor stammt aus dem Jahr 2014 und kann 336 Mrd. Berechnungen pro Sekunde durchführen. Diese Zahl berechnet sich nicht ganz simpel aus den Ghz sondern muss auch die zugrundeliegende Prozessor-Architektur berücksichtigen.

Aufgaben:

a) Verschlüsseln Sie anhand der Cäsar Chiffre einen kurzen Text und lassen Sie einen Partner den Text entschlüsseln (und Sie umgekehrt entschlüsseln Seinen/Ihren Text). Geben Sie ihm/ihr dabei zuerst nicht den Schlüssel, sondern schauen Sie wie lange ihr Code standhält.
b) Versuchen Sie mit Ihrem Partner eine Zahl mit dem RSA Algorithmus zu Verschlüsseln und zu entschlüsseln.
c) Kopieren Sie sich die Vorlage TRESOR aus dem Klassenordner.
d) Machen Sie sich mit der Funktionsweise des TRESORs und seiner Methoden vertraut.
e) Erweitern Sie die Klasse BRUTEFORCE in der Methode Knacken() so, dass der Algorithmus versucht, das Passwort des Tresors zu knacken. Geben Sie am Ende das Passwort aus. Verändern Sie dabei nicht die ersten und letzten Zahlen der Vorlage, da diese Ihnen speichern, wie lange der Algorithmus braucht.
f) Erweitern Sie Ihren Knacken Algorithmus so, dass Sie unterschiedliche Alphabete und Längen der Passwörter erkennen können. (Hinweis: Sie dürfen immer davon ausgehen, dass Sie wissen wie lange das Passwort ist.)
g) Überlegen Sie, wie das Programm anzupassen wäre, um Passwörter mit unbekannter Länge zu knacken. Passen Sie Ihr Programm dementsprechend an.