Sichere Programmierung — Praktikum 1

Thema: Python Einführung — Wintersemester 2020/2021

Prof. Dr. Christoph Karg — Hochschule Aalen

Ziel dieses Praktikums ist die Vertiefung der in der Python Einführung erlangten Kenntnisse. Zu diesem Zweck die Affine Chiffre (ein sehr einfaches Kryptosystem) sowie ein Verfahren zur Kryptoanalyse dieser Chiffre implementiert.

Mit der Affinen Chiffre kann man Klartexte über dem Alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ verschlüsseln. Der Text darf also keine Ziffern, Satzzeichen und Sonderzeichen enthalten. Dies ist in der Regel keine Einschränkung, da man Ziffern „ausschreiben“ und auf Satzzeichen verzichten kann.

Um einen Klartext zu verschlüsseln, werden die Buchstaben als Zahlen der Menge $\mathbb{Z}_{26} = \{0, 1, 2, \ldots, 25\}$ dargestellt. Es wird folgende Zuordnung festgelegt:

a	b	c	d	e	f	g	h	i	j	k	l	m	n
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
0	1	2	3	4	5	6	7	8	9	10	11	12	13

o	p	q	r	s	t	u	v	w	x	y	z
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
14	15	16	17	18	19	20	21	22	23	24	25

Um einen Klartextbuchstaben x zu verschlüsseln, wählt man einen Schlüssel (a, b) bestehend aus einer Zahl $a \in \{x \in \mathbb{Z}_{26} \mid \gcd(x, 26) = 1\}$ und $b \in \mathbb{Z}_{26}$ und berechnet den Geheimtextbuchstaben y anhand der Formel

$$y = (ax + b) \bmod 26.$$

Zur Illustration ein Beispiel.

Um einen Geheimtext zu entschlüsseln, muss die Äquivalenz

$$y \equiv ax + b \pmod{26}$$

nach x aufgelöst werden. Die entsprechende Umformung ist:

$$\begin{aligned} y &\equiv ax + b \pmod{26} \\ y - b &\equiv ax \pmod{26} \\ (y - b)a^{-1} &\equiv x \pmod{26} \end{aligned}$$

Der letzte Schritt ist der kritische Schritt bei dieser Umformung, denn an dieser Stelle wird die Existenz eines multiplikativen Inversen a^{-1} von a modulo 26 vorausgesetzt. Ein solches existiert genau dann, wenn $\gcd(a, 26) = 1$ gilt. Dies ist laut Definition der Affinen Chiffre für jeden Schlüssel der Fall. Die folgende Tabelle enthält alle erlaubten Werte für a und die zugehörigen Inversen modulo 26:

Invertierbare Elemente in $\mathbb{Z}_{26}$												
a	1	3	5	7	9	11	15	17	19	21	23	25
a^{-1}	1	9	21	15	3	19	7	23	11	5	17	25

Zur Illustration ein Beispiel: der Klartext `nachricht` soll mit der Affinen Chiffre unter Einsatz des Schlüssels `lx` (entspricht $(11, 23)$) verschlüsselt werden. Das Vorgehen ist in folgender Tabelle dargestellt:

Klartext	`n`	`a`	`c`	`h`	`r`	`i`	`c`	`h`	`t`
x	13	0	2	7	17	8	2	7	19
$y = (ax + b) \bmod 26$	10	23	19	22	2	7	19	22	24
Geheimtext	`K`	`X`	`T`	`W`	`C`	`H`	`T`	`W`	`Y`

Der erste Teil des Praktikums besteht in der Implementierung der Affinen Chiffre.

Aufgabe 1. Implementieren Sie die Python Funktion `decode(text)`, die als Eingabe einen String `text` erhält. Die Funktion soll die in `text` enthaltenen Buchstaben in Zahlen aus $\mathbb{Z}_{26}$ konvertieren und als Liste zurückgeben. Alle anderen Zeichen wie zum Beispiel Ziffern oder Leerzeichen sollen ignoriert werden.

Beispiel: Für `text = "Hallo Welt!"` liefert `decode(text)` die Liste

```
[7, 0, 11, 11, 14, 22, 4, 11, 19]
```

Dies entspricht übrigens dem Text `hallowelt`. ◊

Aufgabe 2. Implementieren Sie die Python Funktion `encode(char_list)`. Diese Funktion erhält als Eingabe eine Liste `char_list` von Zahlen aus $\mathbb{Z}_{26}$ und soll diese in einen String von kleinen Buchstaben konvertieren.

Beispiel: Auf Eingabe von

```
char_list = [13, 0, 2, 7, 17, 8, 2, 7, 19]
```

liefert die Funktion `encode(char_list)` den String `nachricht`. ◊

Aufgabe 3. Erstellen Sie das Dictionary `key_table`, das alle erlaubten Werte für den Teilschlüssel a mit den zugehörigen inversen Elementen enthält. ◊

Aufgabe 4. Implementieren Sie die Funktion `def acEncrypt(a, b, plain_text)`. Diese Funktion erhält als Eingabe zwei Zahlen `a` und `b` sowie einen String `plain_text` und soll den Text unter Einsatz der Affinen Chiffre verschlüsseln. Das Ergebnis soll als String mit Großbuchstaben ausgegeben werden. Ist der Schlüssel fehlerhaft, dann soll eine Fehlermeldung ausgegeben und der leere String zurück gegeben werden.

Beispiel: Auf Eingabe `a=11`, `b=23` und `plain_text=botschaft` liefert `def acEncrypt(a, b, plain_text)` den Geheimtext `IVYNTWXAY`.

Hinweis: Verwenden Sie `decode`, `encode` und `key_table` in Ihrer Implementierung. ♢

Aufgabe 5. Implementieren Sie die Funktion `def acDecrypt(a, b, cipher_text)`. Diese Funktion erhält als Eingabe zwei Zahlen `a` und `b` sowie einen String `cipher_text` und soll den Text unter Einsatz der Affinen Chiffre entschlüsseln. Das Ergebnis soll als String mit Kleinbuchstaben ausgegeben werden. Ist der Schlüssel fehlerhaft, dann soll eine Fehlermeldung ausgegeben und der leere String zurück gegeben werden.

Beispiel: Auf Eingabe `a=11`, `b=23` und `plain_text=IVYNTWXAY` liefert `def acDecrypt(a, b, cipher_text)` den Geheimtext `botschaft`.

Hinweis: Verwenden Sie `decode`, `encode` und `key_table` in Ihrer Implementierung. ♢

Aufgabe 6.

a) Verschlüsseln Sie den Klartext `strenggeheim` mit dem Schlüssel `db`.

b) Entschlüsseln Sie den Geheimtext `IFFYVQMJYFFDQ` mit dem Schlüssel `pi`.

♢

Aufgabe 7. Fassen Sie die Funktionen `decode`, `encode`, `acEncrypt` und `acDecrypt` sowie das Dictionary `key_table` in einem Modul mit dem Namen `aclib` zusammen. ♢

Aufgabe 8. Implementieren Sie ein ausführbares Python Skript `affinecipher.py` zur Ver- bzw. Entschlüsselung einer Datei mittels der Affinen Chiffre. Das Skript erwartet drei Übergabeparameter:

(1) Betriebsmodus: `e` $\rightsquigarrow$ verschlüsseln, `d` $\rightsquigarrow$ entschlüsseln.

(2) Schlüssel: String mit zwei Buchstaben, z.B. `ht` (entspricht $(7, 19)$).

(3) Pfad zu der zu bearbeitenden Datei.

Das Ergebnis der Chiffrieroperation soll auf der Konsole ausgegeben werden.

Hinweise:

- Greifen Sie bei Ihrer Implementierung auf das Modul `aclibs` zurück.
- Achten Sie bei der Implementierung darauf, dass auftretende Fehler wie fehlende Übergabeparameter oder falsche Schlüssel abgefangen werden.

♢

Aufgabe 9. Verschlüsseln Sie die Datei `klartext.txt` mit dem Schlüssel `pn`. ♢

Aufgabe 10. Entschlüsseln Sie die Datei `geheimtext.txt` mit dem Schlüssel `pn`. ♢

Im zweiten Teil wird ein Verfahren entwickelt, mit dem einen mit der Affinen Chiffre verschlüsselten Geheimtext knacken kann. Hierbei wird ausgenutzt, dass es sich bei der Affinen Chiffre um eine monoalphabetische Chiffre handelt. Dies bedeutet, dass jeder Buchstabe unabhängig von seiner Position im Klartext immer mit demselben Geheimtextbuchstaben verschlüsselt wird. Mittels einer statistischen Analyse kann man hieraus Rückschlüsse über häufige Buchstaben im zugrundeliegenden Klartext ziehen.

Aufgabe 11. Implementieren Sie die Funktion `computeFrequencyTable(char_list)` mit folgender Funktionalität: Auf Eingabe einer Liste `char_list` von Zahlen in $\mathbb{Z}_{26}$ berechnet die Funktion eine Häufigkeitstabelle. Hierunter versteht man ein Dictionary, in dem jede in der Liste vorkommende Zahl mit ihrer Häufigkeit gespeichert ist.

Beispiel: Auf Eingabe

```
[4, 8, 13, 11, 0, 13, 6, 4, 17, 19, 4, 23, 19, 14, 7, 13, 4, 18,
8, 13, 13]
```

soll die Funktion das Ergebnis

```
{0: 1, 4: 4, 6: 1, 7: 1, 8: 2, 11: 1, 13: 5, 14: 1, 17: 1, 18: 1,
19: 2, 23: 1}
```

liefern. ◊

Aufgabe 12. Implementieren Sie die Funktion `printFrequencyTable(freq_table)`, die eine in der vorherigen Aufgabe berechnete Häufigkeitstabelle auf der Konsole ausgibt. Die Zahlen sollen dabei in Buchstaben konvertiert werden.

Beispiel: Auf Eingabe

```
{0: 1, 4: 4, 6: 1, 7: 1, 8: 2, 11: 1, 13: 5, 14: 1, 17: 1, 18: 1,
19: 2, 23: 1}
```

soll die Funktion folgende Ausgabe liefern:

```
a : 1
e : 4
g : 1
h : 1
i : 2
l : 1
n : 5
o : 1
r : 1
s : 1
t : 2
x : 1
```

◊

Aufgabe 13. Implementieren Sie die Python Funktion

```
computeMostFrequentChars(freq_table, n).
```

Diese Funktion soll die `n` häufigsten Zahlen der Häufigkeitstabelle `freq_table` in einer Liste ausgeben.

Beispiel: Für die Eingabe

```
{0: 1, 4: 4, 6: 1, 7: 1, 8: 2, 11: 1, 13: 5, 14: 1, 17: 1, 18: 1,
19: 2, 23: 1}
```

und `n=6` soll die Funktion die Liste

```
[13, 4, 19, 8, 23, 18]
```

berechnen. ◊

Die Grundlage für das erfolgreiche Brechen eines mit der Affinen Chiffre verschlüsselten Geheimtexts ist die Buchstabenhäufigkeit in einem deutschen Text. Diese ist:

Buchstabe	*Häufigkeit*
E	17.40 %
N	9.78 %
I	7.55 %
S	7.27 %
R	7.00 %
A	6.51 %
T	6.15 %
D	5.08 %
H	4.76 %
U	4.35 %
L	3.44 %
C	3.06 %
G	3.01 %

Buchstabe	*Häufigkeit*
M	2.53 %
O	2.51 %
B	1.89 %
W	1.89 %
F	1.66 %
K	1.21 %
Z	1.13 %
P	0.79 %
V	0.67 %
J	0.27 %
Y	0.04 %
X	0.03 %
Q	0.02 %

Gemäß dieser Tabelle sind E und N die beiden häufigsten Buchstaben in einem deutschsprachigen Text. Kennt man die entsprechenden Buchstaben im Geheimtext, dann kann man den benutzten Schlüssel berechnen. Angenommen, E (4) wurde in den Buchstaben c_E verschlüsselt und N (13) in den Buchstaben c_N. Dann gilt:

$$\begin{array}{lrcll} (1) & c_E & \equiv & a \cdot 4 + b & \pmod{26} \\ (2) & c_N & \equiv & a \cdot 13 + b & \pmod{26} \end{array}$$

Durch Lösen dieses Gleichungssystems erhält man:

$$\begin{array}{rcll} a & \equiv & 3 \cdot (c_N - c_E) & \pmod{26} \\ b & \equiv & c_E - 4 \cdot a & \pmod{26} \end{array}$$

Anhand einer Häufigkeitsanalyse des Geheimtexts kann man die „wahrscheinlichsten“ Buchstabenpaare für die Zuordnung von (c_E, c_N) ermitteln. Für jedes dieser Paare wird mit obigen Formeln der vermeintliche Schlüssel (a, b) berechnet. Ist $\gcd(a, 26) = 1$, dann wird der Geheimtext entschlüsselt und kontrolliert, ob der Klartext lesbar ist. Falls ja, dann war die Kryptoanalyse erfolgreich. Ist dagegen der Klartext unlesbar oder ist $\gcd(a, 26) \neq 1$, dann war die Zuordnung falsch. In diesem Fall führt man die Analyse mit einem weiteren Buchstabenpaar fort.

Aufgabe 14. Implementieren Sie die Python Funktion

`computeKeyPairs(char_list)`,

die auf Eingabe einer Liste von Zahlen in $\mathbb{Z}_{26}$ alle möglichen Zahlenpaare (x, y), $x \neq y$, berechnet, wobei x und y Elemente dieser Liste sind.

Beispiel: Auf Eingabe der Liste

```
[13, 4, 19]
```

soll die Funktion die Liste

```
[(13, 4), (13, 19), (4, 13), (4, 19), (19, 13), (19, 4)]
```

ausgeben. ◊

Aufgabe 15. Implementieren Sie die Python Funktion

`analyzeCipherText(cipher_text, char_pairs)`.

Diese Funktion erhält als Eingabe einen Geheimtext in Form eines Strings `cipher_text` sowie eine Liste `char_pairs` von Zahlenpaaren. Für jedes Zahlenpaar (c_E, c_N) wird mit obigen Formeln ein Schlüssel (a, b) berechnet und (falls möglich) der Geheimtext entschlüsselt. Anschließend werden die ersten 50 Zeichen des Klartexts auf der Konsole ausgegeben.

Hinweis: Analysieren Sie mit Ihrer Funktion den in der Datei `ac-cipher1.txt` enthaltenen Geheimtext. (Streng vertraulicher Tipp: Die Datei wurde mit dem Schlüssel `hi` verschlüsselt.) ◊

Aufgabe 16. Fassen Sie die Funktionen `computeFrequencyTable`, `printFrequencyTable`, `computeMostFrequentChars`, `computeKeyPairs` und `analyzeCipherText` in einem Modul mit dem Namen `ablib` zusammen.

Aufgabe 17. Erstellen Sie ein Python Skript `affinebreaker.py`. Dieses Skript erhält als Übergabeparameter den Pfad zu einer Datei. Das Skript soll eine Kryptoanalyse auf dieser Datei durchführen unter der Annahme, dass die Datei einen mit Affinen Chiffre verschlüsselten Geheimtext enthält. Das Ergebnis der Analyse soll auf der Konsole ausgegeben werden. ◊