André Bajorat
Rasmus Diederichsen
Lisa Goerke

Sebastian Höffner
Kristin Schmidt

Lösungen zu Übungsblatt 1
Kryptographische Verfahren

Besprechung 22. Oktober 2015

Aufgabe 1.1. Rechnen mit Resten

a)

$$\begin{aligned}
[a \cdot b]_n &\overset{?}{=} [a]_n \odot_n [b]_n \\
&= [(n \cdot q + r) \cdot (n \cdot p + s)]_n \\
&= [n^2pq + nqs + npr + rs]_n \\
&= [n^2pq]_n \oplus_n [nqs]_n \oplus_n [npr]_n \oplus_n [rs]_n \\
&= [rs]_n \\
&= r \odot_n s \\
&= [a]_n \odot_n [b]_n
\end{aligned}$$

b)

$$\begin{aligned}
((3126 \oplus_{11} 21783) \odot_{11} 213894) \oplus_{11} 31942 &= ([24909]_{11} \odot_{11} 213894) \oplus_{11} 31942 \\
&= (5 \odot_{11} 213894) \oplus_{11} 31942 \\
&= [1069470]_{11} \oplus_{11} 31942 \\
&= 6 \oplus_{11} 31942 \\
&= [31948]_{11} \\
&= 4
\end{aligned}$$

c)

$$\begin{aligned}
\left[2131897^8\right]_3 &= \left[\prod_{i=1}^{8} 2131897\right]_3 \\
&= \prod_{i=1}^{8} [2131897]_3 \qquad \text{Aufgabe 1.1. (a)} \\
&= \prod_{i=1}^{8} 1 \qquad \begin{array}{l}\text{Es sind } 21 \cdot x,\ 3 \cdot x,\ 18 \cdot x \\ \text{und } 9 \cdot x \text{ durch } 3 \text{ teilbar,} \\ \text{sodass nur } 7 \text{ für Rest sorgen kann}\end{array} \\
&= 1
\end{aligned}$$

Aufgabe 1.2. Die Skytale

a)

```
cryptotext = 'PLTIRIERTFEIAIASSNOHKTSNRYEPSETIOVNANSE'
k = 5; w = len(cryptotext)
cleartext = ''.join(
    [''.join(
            [cryptotext[i] for i in range(j, w, k)])
        for j in range(0, k)]
)
print(cleartext)
```

Die Lösung ist: `PIESKYTALEISTEINTRANSPOSITIONSVERFAHREN`, also in etwa `Die Skytale ist ein Transpositionsverfahren`.

b)

Für die Berechnung der `pos` im Java Code muss noch der Parameter `k` bekannt sein.

Angenommen der Index in der `for`-loop soll eigentlich in `i` sein, ergibt sich die Formel:

$$\texttt{pos} = [\texttt{i}]_{\texttt{k}}\, w + \left\lfloor \frac{\texttt{i}}{\texttt{k}} \right\rfloor$$

Die Intuition hier ist: Gehe Spalte für Spalte durch das Array, das 0-te Element zuerst, danach das w-te, dann das $2w$-te und so weiter. Daraus folgt $\texttt{pos} = \texttt{i}w$. Damit nach Erreichen der letzten Zeile wieder die erste verwendet wird, muss `i` alle `k` Schritte zurückgesetzt werden (nur für die Berechnung, nicht als Loopindex), d.h. $\texttt{pos} = [\texttt{i}]_{\texttt{k}}\, w$. So würde jedoch immer nur die erste Spalte ausgelesen. Um zusätzlich die Spalte zu erhöhen, wenn das Ende erreicht ist, kann $\left\lfloor \frac{\texttt{i}}{\texttt{k}} \right\rfloor$ addiert werden, was den Spaltenindex für den Loopindex `i` bei Spalten mit `k` Elementen angibt. Es folgt $\texttt{pos} = [\texttt{i}]_{\texttt{k}}\, w + \left\lfloor \frac{\texttt{i}}{\texttt{k}} \right\rfloor$.

Vollständig implementiert folgt:

```
        for (int i = 0; i < klartext.length(); i++) {
            int pos = ((i % key) * w) + i / key;
            kryptotext += klartext.charAt(pos);
        }
```

`PIESKYTALEISTEINTRANSPOSITIONSVERFAHREN` wird korrekt ($\texttt{k} = 5$) zum Kryptotext aus Aufgabe 1.2. (a).

c)

Es kann passieren, dass die letzte Zeile nicht vollständig oder gar überhaupt nicht gefüllt wird. Im Beispiel aus Aufgabe 1.2. (a) ist dies kein Problem, ein Zeichen ist in Ordnung. Doch wenn in der letzten Zeile mehrere Zeichen fehlen (oder gar mehrere Zeichen in der letzten Spalte, z.B. bei $|\texttt{M}| = 41, \texttt{k} = 8$) dann gibt es ein Problem.

Genau dies ist der Fall für $|\texttt{M}| = 42, \texttt{k} = 8$. Es folgt $w = 6$. Tatsächlich ist $|\texttt{M}|$ ein Vielfaches von w: $7w = |\texttt{M}|$. Das bedeutet, es werden nur 7 Zeilen gefüllt, die unterste Zeile bleibt leer.

Das würde in der Praxis bedeuten, dass das Band mit Leerzeichen aufgefüllt wird (unter Umständen mit Ausnahme des letzten Zeichens). Natürlich könnte auch ein beliebiges anderes Zeichen gewählt werden, nennen wir dieses Zeichen $\texttt{x}$. Nun kann ein Angreifer versuchen durch geschickes anordnen gleicher Buchstaben nebeneinander ohne den Schlüssel zu kennen die Nachricht zu entschlüsseln.

Um dieses Problem zu umgehen gibt es zwei einfache Möglichkeiten:

1. Klartext-Schlüssel-Kombinationen wählen, die möglichst gut $|\texttt{M}| = \texttt{k}w$ annähern.
2. Den Klartext mit zufälligen Zeichen auffüllen, bis $|\texttt{M}| = \texttt{k}w$ annähernd gilt.

Ein weiteres nicht mit der Skytale lösbares Problem ist, dass der Angreifer durch Faktorisierungen der Zahlen $|\texttt{M}| \pm 1$ bereits seinen Raum an Entschlüsselungsmöglichkeiten stark eingrenzen kann.

Aufgabe 1.3. Cäsar[2]

a)

- Die Schlüsselpaare (k_1, k_2) und (k_2, k_1) führen zur selben Codierung, sodass die Schlüsselmenge effektiv wieder halbiert wird, da man nur eine von je zwei Möglichkeiten testen muss
- Es gilt

$$\begin{aligned} E_{k_1,k_2}(M) &= E_{k_2}(E_{k_1}(M)) \\ &= [[M + k_1] + k_2]_{26} \\ &= [M + k_1]_{26} \oplus_{26} [k_2]_{26} \\ &= [M + k_1]_{26} \oplus_{26} k_2 \\ &= [M]_{26} \oplus [k_1]_{26} \oplus_{26} k_2 \\ &= [M]_{26} \oplus k_1 \oplus_{26} k_2 \\ &= [M + (k_1 + k_2)]_{26} \end{aligned}$$

  Somit stellt sich das selbe Problem wie bei der normalen Cäsar-Verschlüsselung. Sobald ein Klartextzeichen und das zugehörige Kryptozeichen bekannt sind, kann die Summe $(k_1 + k_2)$ berechnet werden. Zwar kennt man die Einzelschlüssel nicht, kann mit deren Summe aber dennoch alle Texte entschlüsseln.

b)

Ein Gegenbeispiel für $k = 2$ ist $E_2(13) = [2 \cdot 13]_{26} = 0 = E_2(0)$. Ein Gegenbeispiel für $k = 13$ ist $E_{13}(2) = [13 \cdot 2]_{26} = 0 = E_{13}(0)$. Generell tritt das Problem dann auf, wenn der Schlüssel die Klartextalphabetgröße ganzzahlig teilt, bzw. Schlüssel und Alphabetgröße nicht teilerfremd sind. Dies ist hier nur für $k = 2$ und $k = 13$ der Fall. Aus diesem Grund sollte man als Alphabetgröße eine Primzahl wählen.