# Komplexitätstheorie

Sommersemester 2020

Mario Holldack, M. Sc.
Prof. Dr. Georg Schnitger
Hannes Seiwert, M. Sc.

GOETHE UNIVERSITÄT
FRANKFURT AM MAIN

Institut für Informatik
AG Theoretische Informatik

# Übungsblatt 5

Ausgabe: 25.05.2020
Abgabe: 02.06.2020 um 8 Uhr

Die Abgabe erfolgt per E-Mail an Mario Holldack (`holldack@em.uni-frankfurt.de`).

**Aufgabe 5.1** *Faktorisierung und Quadratwurzeln* $(3+(1+4)=8$ Punkte)

Für jedes $n \in \mathbb{N}_{>0}$ sei $\mathbb{Z}_n^*$ die Menge der *primen Restklassen* modulo $n$ und $\mathrm{QR}_n$ die Menge der *quadratischen Reste* modulo $n$, wobei $\mathbb{Z}_n^* = \{i \in \{1,\dots,n-1\} : \mathrm{ggT}(i,n) = 1\}$ und $\mathrm{QR}_n = \{i^2 \bmod n : i \in \mathbb{Z}_n^*\}$. Eine *Quadratwurzel* von $c \in \mathrm{QR}_n$ ist eine Zahl $m \in \mathbb{Z}_n^*$ mit $m^2 \equiv c \bmod n$.

In dieser Aufgabe zeigen wir, dass die Bestimmung aller vier Quadratwurzeln eines quadratischen Restes $c \in \mathrm{QR}_n$ genauso schwierig ist wie die Faktorisierung von $n$.

a) Geben Sie einen effizienten Algorithmus an, der bei Eingabe einer Primzahl $p$ mit $p \equiv 3 \bmod 4$ und eines quadratischen Restes $c \in \mathrm{QR}_p$ eine Quadratwurzel $m$ von $c$ ausgibt.

*Hinweis*: Setzen Sie $m \equiv c^d \bmod p$ für einen geeigneten Exponenten $d \in \mathbb{N}$ und nutzen Sie den „Kleinen Fermat“: $a^{p-1} \equiv 1 \bmod p$ für alle $a \in \mathbb{Z}_p^*$.

b) i) Zum Aufwärmen: Geben Sie die vier Quadratwurzeln $m_1, m_2, m_3$ und $m_4$ von 4 in $\mathbb{Z}_{15}^*$ an.

ii) Sei $n = pq$ das Produkt zweier verschiedener Primzahlen $p$ und $q$. Sei $A$ ein Algorithmus, der bei Eingabe von $n$ und $c \in \mathrm{QR}_n$ die vier Quadratwurzeln $m_1, m_2, m_3$ und $m_4$ von $c$ ausgibt. Geben Sie die Primfaktoren $p$ und $q$ in Abhängigkeit von $m_1, m_2, m_3$ oder $m_4$ an und begründen Sie die Korrektheit.

*Hinweis*: Betrachten Sie die größten gemeinsamen Teiler $\mathrm{ggT}(m_i + m_j, n)$ und $\mathrm{ggT}(m_i - m_j, n)$ für geeignete $i, j \in \{1,2,3,4\}$.

**Aufgabe 5.2** *Verkettung von One-Way-Permutationen* $(3+3=6$ Punkte)

a) Sei $k = \mathtt{poly}(n)$. Zeigen Sie: Wenn $\pi_n : \{0,1\}^n \to \{0,1\}^n$ eine One-Way-Permutation ist, dann ist

$$\pi_n^k := \underbrace{\pi_n \circ \pi_n \circ \cdots \circ \pi_n}_{k\text{-mal}}$$

eine One-Way-Permutation, wobei $\pi_n \circ \pi_n \circ \cdots \circ \pi_n(x) = \pi_n(\pi_n(\cdots \pi_n(x) \cdots))$ die $k$-fache Verkettung von $\pi_n$ mit sich selbst ist.

b) Annahme: Es gibt One-Way-Permutationen. Zeigen Sie: Für jedes genügend große $n \in \mathbb{N}_{>0}$ gibt es eine natürliche Zahl $k$ mit $k \leq n$, sodass Folgendes gilt: Es gibt eine One-Way-Funktion $f_n : \{0,1\}^n \to \{0,1\}^n$, sodass

$$f_n^k := \underbrace{f_n \circ f_n \circ \cdots \circ f_n}_{k\text{-mal}}$$

*keine* One-Way-Funktion ist.

*Hinweis*: Bestimmen Sie $f_n$ so, dass $|\mathrm{Bild}(f_n^i)| = |\{f_n^i(x) : x \in \{0,1\}^n\}| = 2^{n-i+1}$ für alle $i \in \{1,\dots,n\}$, d. h. bei jeder weiteren Verkettung von $f_n$ mit sich selbst halbiert sich die Anzahl der getroffenen Bitstrings.

**Aufgabe 5.3** *Blockchiffrierung und AES* $((1+1+2+2)+(2+2) = 10$ Punkte)

Wir verschieben die Abgabe dieser Aufgabe auf Übungsblatt 6. Die Aufgabenstellung verbleibt hier, wird aber als Aufgabe 6.1 noch einmal wiederholt.

Der *Rijndael-Algorithmus* ist der im AES (Advanced Encryption Standard) spezifizierte Verschlüsselungsalgorithmus. Es handelt sich um eine Blockchiffrierung, d. h. der zu verschlüsselnde Klartext wird in gleichgroße Blöcke von jeweils 128 Bits aufgeteilt. Anschließend werden die Blöcke einzeln verschlüsselt.

In dieser Aufgabe werden wir uns einen ersten Eindruck über dieses in der Praxis weit verbreitete Verschlüsselungsverfahren verschaffen und eine interessante Verbindung zwischen Algebra und superschnellen Berechnungen aufzeigen.

a) Lesen Sie den Abschnitt 4.5 in S. Goldwasser und M. Bellare: Lecture Notes on Cryptography.

   i) Woher stammt der Name des Rijndael-Algorithmus?

   ii) Handelt es sich um ein Private-Key- oder um ein Public-Key-Verschlüsselungsverfahren?

   iii) Beschreiben Sie kurz die Schritte des Rijndael-Algorithmus. Sie müssen nicht auf Implementierungsdetails eingehen. Konzentrieren Sie sich auf die wesentlichen Ideen.

   iv) Wie kann der Klartext wiederhergestellt werden? Auch hier genügt eine kurze Beschreibung der wesentlichen Ideen.

b) In dieser Teilaufgabe betrachten wir die Implementierung des Rijndael-Algorithmus aus mathematischer Perspektive. Jedes Byte $a = a_7a_6 \dots a_0 \in \{0,1\}^8$ assoziieren wir mit dem univariaten Polynom

$$p_a(x) := \sum_{i=0}^{7} a_i x^i \mod (x^8 + x^4 + x^3 + x + 1)$$

über dem endlichen Körper $\mathrm{GF}(2^8)$ mit $2^8$ Elementen. (Die Elemente von $\mathrm{GF}(2^8)$ sind also Polynome vom Grad höchstens 7 mit $\{0,1\}$-wertigen Koeffizienten.)

   i) Multiplizieren Sie die Bytes $a_7a_6 \dots a_0$ und $0000\,0010$, d. h. berechnen Sie

$$\left(\sum_{i=0}^{7} a_i x^i\right) \cdot x \mod (x^8 + x^4 + x^3 + x + 1).$$

   Geben Sie das Ergebnis als Byte an. Erläutern Sie kurz, wie die Multiplikation mit $0000\,0010$ durch Bitshifts und bitweises XOR implementiert werden kann.
   *Hinweis*: Unterscheiden Sie die Fälle $a_7 = 0$ und $a_7 = 1$ und zeigen Sie

$$x^8 = x^4 + x^3 + x + 1 \mod (x^8 + x^4 + x^3 + x + 1).$$

   ii) Geben Sie das Produkt der Bytes $1001\,1010$ und $0010\,1001$ an.
   *Hinweis*: Nutzen Sie Teil i) und

$$0010\,1001 = 0010\,0000 \oplus 0000\,1000 \oplus 0000\,0001.$$

**Fazit:** Die Rechenoperationen des Rijndael-Algorithmus können superschnell durch Bitshifts und bitweises XOR bzw. durch Lookup-Tabellen implementiert werden.