Aufgabenblatt 09

PS Einführung in die Kryptographie

Andreas Schlager

23. Mai 2025

Inhaltsverzeichnis

Aufgabe 31. Recherchieren sie ein weiteres Kriterium zur Bestimmung einer Primitivwurzel (zu dem auf Slide 150 der VO) und implementieren sie Experimente, in denen sie, für wachsende Modulgrösse, den Zeitbedarf beider Kriterien bestimmen. Hinweis: z.B. in Mathematica sind diverse hilfreiche zahlentheoretische Funktionen - wie z.B. Faktorisierung - implementiert.

Eine Primitivwurzel q modulo p ist eine Zahl im Restklassenring von p, die durch modulares potenzieren alle Zahlen im Restklassenring erzeugt. In der Vorlesung wurde ein einfaches Kriterium für die Bestimmung einer Primitivwurzel vorgestellt. Sei $q < p$ und

$$\forall b \quad 1 \leq b \leq p-1 \quad \exists a : q^a \equiv b \mod p,$$

so ist q eine Primitivwurzel von p. Also kann durch einfaches Ausprobieren aller Exponenten modulo p festgestellt werden, ob q eine Primitivwurzel ist.

Brute-Force Algorithmus Es wird angenommen, dass die gegebene Zahl p wirklich eine Primzahl ist. Für die Überprüfung könnte ein probabilistischen Primzahlentest, wie etwa der Miller-Rabin-Test, verwendet werden. Für jede Zahl $q \in [2; p-1]$ und $i \in [1; p-1]$ wird die modulare Exponentiation durchgeführt und das Ergebnis in einem Set gespeichert. Ist das Ergebnis bereits vorhanden, bzw. kommt doppelt vor, so kann q keine Primitivwurzel sein, da jeder Exponent zu einer einzigartigen Zahl führen muss.

```rust
pub fn primitive_root_brute_force(p: u128) -> Option<u128> {
    let phi = p - 1;
    for q in 2..phi {
        let mut unique_numbers = HashSet::new();
        let unique_numbers = (1..=phi)
            .take_while(|x| unique_numbers.insert(mod_exp(q, *x, p)))
            .count();

        if unique_numbers == phi as usize {
            return Some(q);
        }
    }
    None
}
```

Ordnungskriterium Ein weiteres Kriterium für eine Primitivwurzel verwendet die multiplikative Ordnung einer Zahl und die eulersche φ-Funktion.

Korollar 1 (Primitivwurzel)**.** Sei $q < p$ und $p \in \mathbb{P}$. q ist genau dann eine Primitivwurzel modulo p, wenn

$$\mathrm{ord}_p(q) = \varphi(p) = p - 1.$$

Die Ordnung $\text{ord}_p(q)$ ist der kleinste Exponent $i \in [1; p-1]$ für den

$$q^i \equiv 1 \mod p$$

Der Satz von Lagrange besagt, dass die Ordnung eines Elements einer endlichen zyklischen Gruppe die Ordnung der Gruppe selbst, hier $\varphi(p)$, teilt. Man nutzt nun diese Eigenschaft aus und probiert anstelle aller Exponenten nur die Teiler von $\varphi(p)$ (alle möglichen Ordnungen der Gruppe). Ist der kleinste Exponent i für den $q^i \equiv 1 \mod p$ gilt, gleich $p-1$, so ist q eine Primitivwurzel.

Diese Vorgehen kann noch weiter verbessert werden, indem nur die Primfaktoren von $\varphi(p)$ berechnet werden und

$$q^{p-1/g} \not\equiv 1 \mod p \qquad \forall \text{ Primfaktoren } g \mid p-1 \tag{1}$$

überprüft wird[1].

Lagrange-Primfaktoren-Algorithmus Zuerst werden die Primfaktoren von $p-1$ bestimmt. Anschließend wird, wie beim Brute-Force Algorithmus, über alle $q \in [2; p-1]$ iteriert und das Kriterium aus 1 überprüft. Falls keine Berechnung 1 ergibt, liefert die Methode als Ergebnis das aktuelle q. In den Zeilen 5-8 ist zu sehen, wie die Primfaktoren auf die modulare Exponentiation gemapped und anschließend mittels `all` auf die Ungleichheit zu 1 überprüft werden.

```
1  pub fn primitive_root_lagrange(p: u128) -> Option<u128> {
2      let phi = p - 1;
3      let prime_factors = Factorization::run(phi).prime_factor_repr();
4      for q in 2..phi {
5          let is_primitive_root = prime_factors
6              .iter()
7              .map(|(factor, _)| mod_exp(q, phi / factor, p))
8              .all(|x| x != 1);
9
10         if is_primitive_root {
11             return Some(q);
12         }
13     }
14     None
15 }
```

[1] Otto Forster, Einführung in die Zahlentheorie, [22.05.2025], Korollar 9.5

Benchmarks Zum Vergleich der zwei Algorithmen wurde die `criterion` Benchmarking-Bibliothek verwendet. Jede Funktion wurde mit einer größer werdenden Folge von Primzahlen aufgerufen und der Zeitbedarf gemessen. Die verwendete Primzahlen Liste ist überschaubar, da die Laufzeiten, vorallem von der Brute-Force Variante, schnell gestiegen sind.

$$\text{primes} = \{223, 1033, 2239, 5179, 7919, 22447, 73907, 111799, 153913\}$$

Wie in Abb. 1 zu sehen ist, steigt die Laufzeit der Brute-Force Variante besonders schnell

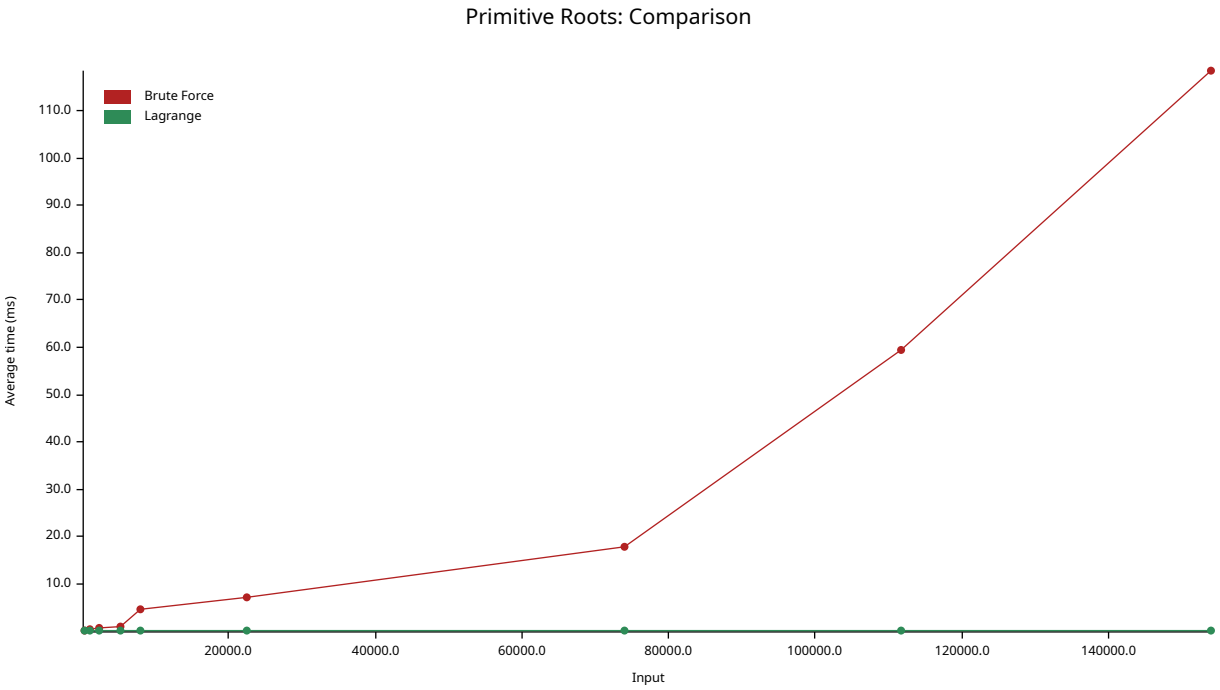

Abbildung 1: Vergleich der Laufzeiten bei ansteigenden Primzahlen

im Vergleich zum Lagrange-Primfaktoren-Algorithmus (grüne Linie, kaum zu sehen). Der verbesserte Algorithmus bewegt sich durchwegs im μs Bereich, wobei der langsame schnell Millisekunden bzw. Sekunden benötigt. In Abb. 2 ist nochmal deutlich zu sehen, dass die Laufzeit bei der Lagrange-Variante nur langsam ansteigt. In Abb. 3 ist hingegen ein

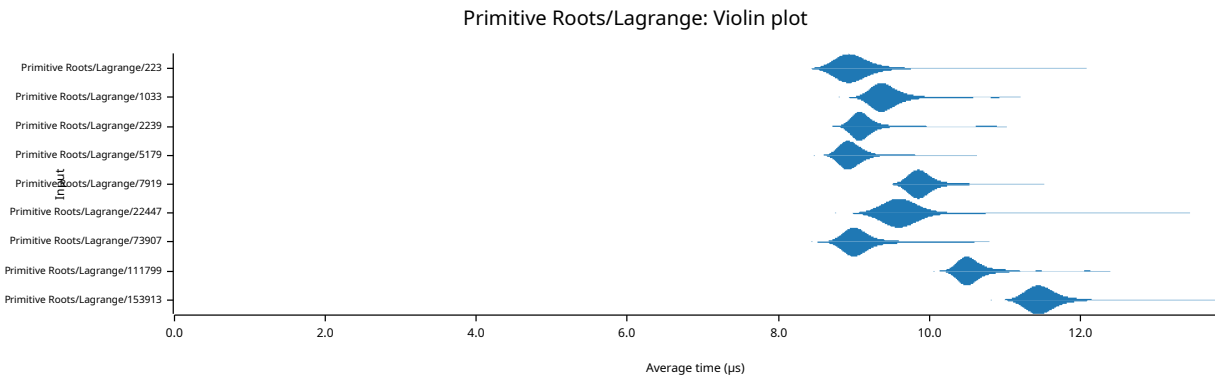

Abbildung 2: Violinen-Plot der Laufzeit der Lagrange-Primfaktoren Variante

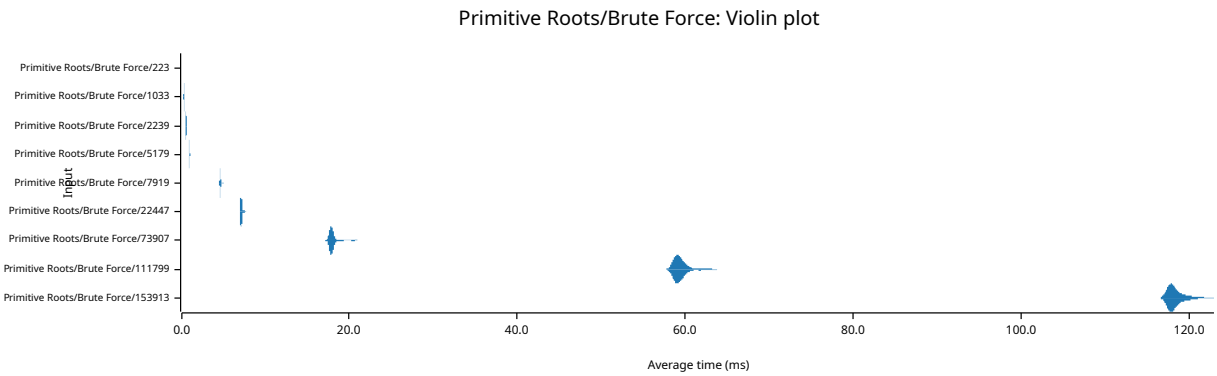

Abbildung 3: Violinen-Plot der Laufzeit der Brute-Force Variante

deutlicher Anstieg der Laufzeit zu erkennen.

Aufgabe 32. Beweisen sie die Korrektheit der RSA Ver- und Entschlüsselungsformel für $(m_i, n) = 1$ und $(m_i, n) = 1$.

Die Formeln sind korrekt, wenn in beiden Fällen eine Entschlüsselung nach der Verschlüsselung wieder zum ursprünglichen Klartext führt. Seien p und q zwei zufällige, große Primzahlen und $n = pq$. Eine Nachricht $m < n$ kann durch

$$c = m^e \mod n \tag{2}$$

verschlüsselt werden, um den Ciphertext c zu erhalten. Die Entschlüsselung erfolgt dann über

$$m = c^d \mod n. \tag{3}$$

e und d sind so gewählt, dass $ed \equiv 1 \mod \varphi(n)$.

Fall 1 $(m, n) = 1$, d.h. m und n sind teilerfremd. Die eulersche Verallgemeinerung des kleinen Fermats besagt, wenn m und n teilfremd sind, dann ist

$$m^{\varphi(n)} \equiv 1 \mod n. \tag{4}$$

Wenden wir die Entschlüsselungsformel 3 an, so erhalten wir

$$\begin{gathered} m = c^d \mod n = m^{e^d} \mod n = m^{ed} \mod n \\ m \equiv m^{ed} \mod n \end{gathered}$$

wobei e, d so gewählt wurden, dass

$$ed \equiv 1 \mod \varphi(n) \iff ed = 1 + k\varphi(n)$$

Wir setzen also für ed ein

$$\begin{aligned} m^{ed} &\equiv m^{1+k\varphi(n)} \mod n \\ &\equiv m \cdot \left(m^{\varphi(n)}\right)^k \mod n \\ &\equiv m \cdot (1)^k \mod n \\ &\equiv m \mod n \end{aligned}$$

Fall 2 $(m, n) \neq 1$, d.h. m und n sind **nicht** teilerfremd. Somit ist m ein Vielfaches von p oder von q. Wir können o.B.d.A. annehmen, dass m ein Vielfaches von p ist. Somit gilt

$$m \equiv 0 \equiv m^{ed} \mod p \Longrightarrow m \equiv m^{ed} \mod p$$

Da m nur ein Vielfaches von entweder p oder q sein kann und q eine Primzahl ist, sind q und m teilerfremd. Der kleine Satz von Fermat besagt, dass für eine Primzahl und einer teilerfremden Zahl, die kein Vielfaches ist

$$m^{q-1} \equiv 1 \mod q$$

gilt.

$$\begin{aligned}
m^{ed} &\equiv m^{1+k\varphi(n)} \mod q \\
&\equiv m^{1+k(p-1)(q-1)} \mod q && \mid k' := k(p-1) \\
&\equiv m^{1+k'(q-1)} \mod q \\
&\equiv m \cdot \left(m^{(q-1)}\right)^{k'} \mod q && \mid \text{kleiner Fermat} \\
&\equiv m \cdot (1)^{k'} \mod q \\
&\equiv m \mod q
\end{aligned}$$

Wir wissen also, dass

$$m^{ed} \equiv m \mod p \qquad \text{und} \qquad m^{ed} \equiv m \mod q.$$

Somit ist auch

$$m^{eq} \equiv m \mod pq \iff m^{eq} \equiv m \mod n.$$

Aufgabe 33. Warum ist RSA in der bisherigen Beschreibung (sog. Textbook RSA) nicht IND-CPA? Wie wird mit RSA typischerweise diese Sicherheitsstufe erreicht?

Reguläres (Textbook RSA) ist nicht IND-CPA sicher, weil es deterministisch ist. Das heißt, der gleiche Plaintext wird immer gleich verschlüsselt. Falls der Angreifer in der Lage ist, zwei Ciphertexte mit einer Wahrscheinlichkeit $> 1/2$ zu unterscheiden, ist die Verschlüsselung nicht IND-CPA sicher.

Angenommen der Angreifer ist in der Lage den Klartextraum einzuschränken, weil er etwa weiß, dass nur 'ja' oder 'nein' als Nachrichten verschickt werden. Ausgehend von dem Orakel-Modell, könnte er dem Orakel beide Texte zum Verschlüsseln geben. Das Orakel wählt zufällig einen der Texte aus, verschlüsselt ihn mit dem öffentlichen RSA-Schlüssel und schickt ihn an den Angreifer zurück. Der Angreifer kann nun selbst beide Nachrichten mit dem bekannten öffentlichen Schlüssel verschlüssen und einen einfachen Vergleich der Ciphertexte durchführen, um den ausgewählten Klartext herauszufinden.

Um RSA IND-CPA sicher zu machen, muss der Plaintext mit zufälligen Bits ausgepolstert werden. Dadurch wird sich der Ciphertext, den der Angreifer generiert, vom Ciphertext des Orakels unterscheiden. In der Praxis wird das z.B durch OAEP (Optimal Asymmetric Encryption Padding) realisiert.