DIPLOMARBEIT

Titel der Diplomarbeit

Faktorisierungsverfahren für ganze Zahlen

Verfasser

Manuel Wolf

angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag.rer.nat)

Wien, März 2012

Studienkennzahl lt. Studienblatt:	A 405
Studienrichtung lt. Studienblatt:	Mathematik
Betreuer:	Assoz. Prof. Dr. Dietrich Burde

Danksagung

Zu allererst möchte ich mich herzlich bei meinem Betreuer Dietrich Burde bedanken, der mich über die ganze Zeit hinweg geduldig mit seinem Wissen bei der Erstellung meiner Diplomarbeit unterstützt hat. Weiters möchte ich mich bei meinen Eltern Ulli und Helmuth bedanken, die oft einen Rat für mich hatten und mich finanziell unterstützt haben. Abschließend bedanke ich mich bei meiner Freundin Margarita, die immer geduldig bei der Anfertigung der Arbeit war und mir moralisch zur Seite gestanden ist.

Inhaltsverzeichnis

Teil I
Einleitung

*Eine **Primzahl** ist eine natürliche Zahl größer als Eins,*
die nur sich selbst und Eins als Teiler besitzt.

Der Fundamentalsatz der Arithmetik (Satz 1.0.4), der von Euklid um 300 vor Christus vermutet und von Carl Friedrich Gauß in der Disquisitiones Arithmeticæ um 1800 bewiesen wurde, besagt, dass jede Zahl eine im Wesentlichen eindeutige Primfaktorzerlegung besitzt. Die daraus resultierende Problemstellung für eine beliebige ganze Zahl einen (Prim-)Teiler zu berechnen ist unter dem Namen „Faktorisierungsproblem“ bekannt. Daraus ergibt sich die einfache Frage, ist es möglich für jede beliebige zusammengesetzte ganze Zahl einen Teiler bzw. dessen komplette Primfaktorzerlegung finden? Schon Gauß erkannte die Wichtigkeit des Problems, als er sagte:

„The problem of distinguishing prime numbers from composites,
and of resolving composite numbers into their prime factors,
is one of the most important and useful in all of arithmetic.“

Für beliebige Zahlen ist die Berechnung eines Faktors im Allgemeinen nicht möglich, also stellt sich als nächstes die Frage, bis zur welchen Größe und wie können wir einen Teiler für eine beliebige zusammengesetzte Zahl finden? Diese Problemstellung klären wir in dieser Arbeit, ebenso wie die Frage, wie schnell wir aktuell einen Faktor für eine beliebige Zahl berechnen können.

Zu Beginn in Teil II – der das mathematische Basiswissen für diese Arbeit abdeckt – behandeln wir den Ring $\mathbb{Z}/n\mathbb{Z}$ (Kapitel 2) und die Gruppe $(\mathbb{Z}/n\mathbb{Z})^*$ (Kapitel 2.1), die beide in mehreren Faktorisierungsalgorithmen, wie den Algorithmen von Pollard, der Methode von Legendre und der Elliptic Curve Method, eine zentrale Rolle spielen. Weiters benötigen wir des Öfteren die Theorie der quadratischen Reste (Kapitel 4), wie etwa für den Kettenbruchalgorithmus, das quadratische Sieb, das Zahlkörpersieb und die Methode von Legendre. Für die den Namen entsprechenden Faktorisierungsalgorithmen widmen wir uns außerdem den Zahlkörpern (Kapitel 3), den Kettenbrüchen (Kapitel 5) und den elliptischen Kurven (Kapitel 6).
Zumeist geben wir für die Faktorisierungsalgorithmen die (heuristische) Laufzeit und teilweise auch eine Laufzeitanalyse an und diskutieren dafür die Grundbegriffe der Komplexitätstheorie in Kapitel 7. In Teil III verfassen wir für mehrere Algorithmen verwendbare Implementierungen für die Computeralgebra Software PARI/GP und erklären in Anhang A die wichtigsten Funktionen für diese. Der in Kapitel 8 vorgestellte RSA-Algorithmus aus der Kryptographie gehört zu den wichtigsten praktischen Vertretern des Faktorisierungsproblems. Dieses aus den 70er-Jahren stammende asymmetrische Verfahren macht sich die Schwierigkeit des Faktorisierungsproblems zu Nutzen, um Daten, vorwiegend zur Übermittlung über das Internet, zu verschlüsseln. Dadurch bekam damals die Forschung über Faktorisierungsalgorithmen neue Impulse durch diesen kommerziellen Nutzen.

In Teil III, dem Kernteil der Arbeit, behandeln wir die historisch und aktuell wichtigsten Algorithmen zum Faktorisieren ganzer Zahlen. Den Beginn machen wir bei der Probedivision in Kapitel 9, dem einfachsten und wahrscheinlich ältesten Faktorisierungsalgorithmus, der aber immer noch zum Aufspüren kleiner Faktoren bis ca. 8 Dezimalstellen verwendet werden kann. In den 70er-Jahren entwickelte John Pollard zwei Algorithmen, den ρ-Algorithmus (Kapitel 10) und den $(p-1)$-Algorithmus (Kapitel 11). Diese Verfahren werden zwar nicht zur Berechnung neuer Faktorisierungsrekorden benutzt, sind aber effizienter als die Probedivision zum Aufspüren kleiner Teiler. Die $(p-1)$-Methode ist zusätzlich bei einer gewissen Art von Zahlen sehr effizient, wo selbst modernere Verfahren deutlich längere Rechenzeiten in Anspruch nehmen. Eine Art Verallgemeinerung der $(p-1)$-Methode ist

die Elliptic Curve Methode nach Hendrik H. Lenstra Jr., den wir in Kapitel 12 behandeln. Dieser merzt die Schwächen der $(p-1)$-Methode weitgehend aus und kann damit Teiler mit bis aktuell maximal 73 Dezimalstellen von beliebigen Zahlen berechnen.
Danach behandeln wir zwei historische Algorithmen, die Faktorisierungsmethode nach Fermat (Kapitel 13) aus dem 17. Jahrhundert und die Methode nach Legendre (Kapitel 14) aus dem 18. Jahrhundert, die jeweils nach dem Entwickler benannt wurden. Diese beiden Algorithmen sind heute für Faktorisierungsprogramme weitgehend uninteressant – sie sind ineffizienter als die Probedivision –, aber trotzdem erwähnenswert, da sie die Basis der modernen und schnellsten Faktorisierungsmethoden bilden.
Der chronologisch erste Algorithmus – basierend auf Fermat und Legendre – ist der Kettenbruchalgorithmus aus Kapitel 15, der 1931 von Derrick Henry Lehmer und Ralph Ernest Powers entwickelt wurde und bis in die 80er-Jahre als schnellste Faktorisierungsmethode galt. Mit ihm wurde etwa der Faktorisierungsrekord von 1970 durch die Faktorisierung der 7. Fermat-Zahl F_7 durch Michael Morrison und John Brillhart aufgestellt (siehe [44]). Das quadratische Sieb (Kapitel 16) wurde 1982 von Carl Pomerance entwickelt, baut ebenfalls auf der Idee von Fermat und Legendre auf und löste den Kettenbruchalgorithmus als schnellste Methode ab. Bis Mitte der 90er-Jahre wurde mit dem quadratischen Sieb bzw. mit Varianten davon Faktorisierungsrekorde aufgestellt, etwa Ende der 80er-Jahre die Faktorisierung der ersten 100-stelligen Zahl. Seit damals werden mit dem Zahlkörpersieb (Kapitel 17) die größten Zahlen faktorisiert, zuletzt 2009 die 232-stellige Zahl RSA-756, also eine 756-Bit-Zahl mit etwa zwei gleich großen Primteilern, durch ein Team um Thorsten Kleinjung. Das Zahlkörpersieb ist auch der Algorithmus mit der heuristisch schnellsten Laufzeit

$$O\left(\exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\log(a))^{\frac{1}{3}}\,(\log(\log(a)))^{\frac{2}{3}}\right)\right)$$

für eine natürliche Zahl a und ist, wie auch das quadratische Sieb ein aktuelles Thema in der Forschung.

Im praxisbezogenen Teil IV stellen wir neben PARI/GP kurz weitere aktuelle Programme zur Zahlenfaktorisierung vor, unter anderem YAFU, GGNFS, Mathematica und Darios ECM, die alle wichtigen Faktorisierungsmethoden abdecken. Anhand von ausgewählten Zahlen vergleichen wir diese durch einen Geschwindigkeitstest und zeigen für welche Art von Zahlen sie aufgrund der verwendeten Methoden geeignet sind.

Abschließend (Teil V) streifen wir durch die Geschichte der Zahlenfaktorisierung: von Euklid, über Gauß, Fermat, Legendre, Lehmer, bis hin zu Morrison, Brillhart, Lenstra, Kleinjung und vielen mehr, die alle einen Beitrag zum Fortschritt der Zahlenfaktorisierung geleistet haben. Für die letzten zwei bis drei Jahrzehnte betrachten wir die Entwicklung der Faktorisierungsrekorde genauer und geben den aktuellen Stand für Faktorisierungen allgemeiner Zahlen und Zahlen in spezieller Form, unter anderem von Mersenne-Zahlen und Fermat-Zahlen. Im letzten Kapitel werfen wir einen theoretischen Blick in die Zukunft der Faktorisierungen und prophezeien, welche Größe an Zahlen wir in 40 Jahren faktorisieren können. Außerdem stellen wir den auf Quantencomputern basierenden polynomialen Shor-Algorithmus vor, die theoretisch schnellste Faktorisierungsmethode.

Teil II
Mathematisches Basiswissen

Dieser Teil der Arbeit befasst sich mit den notwendigen mathematischen Theorien für die Faktorisierungsalgorithmen aus Teil III. Der Fokus liegt dabei ausschließlich auf der Auffrischung und der Angleichung von Notationen; für Beweise verweisen wir auf entsprechende Literatur.

1 Grundlegendes

Im ersten Kapitel behandeln wir grundlegende Begriffe, Funktionen und Eigenschaften, angefangen beim größten gemeinsamen Teiler bis hin zum Fundamentalsatz der Algebra. Vorausgesetzt wird die Kenntnis der Definitionen von Gruppen, Ringen und Körpern sowie den üblichen Mengen $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ und $\mathbb{C}$. Die Menge $\mathbb{N}$ ist je nach Gebrauch in- oder exklusive der Null anzusehen, wobei wir zur Verdeutlichung die Schreibweisen $\mathbb{N}_0 = \{0, 1, 2, \dots\}$ und $\mathbb{N}^* = \{1, 2, 3, \dots\}$ verwenden. Die Menge der Primzahlen bezeichnen wir mit $\mathbb{P}$.

Definition 1.0.1. Mit „$\gcd(a, b)$“ bezeichnen wir den **größten gemeinsamen Teiler** und mit „$\operatorname{lcm}(a, b)$“ das **kleinste gemeinsame Vielfache** zweier ganzer Zahlen a und b.

Definition 1.0.2. Wir definieren die **Primzahlzählfunktion** $\pi : \mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\} \to \mathbb{N}$ durch

$$\pi(x) = \sum_{\substack{p \leq x, \\ p \in \mathbb{P}}} 1$$

und die **Euler'sche Phi-Funktion** $\varphi : \mathbb{N} \to \mathbb{N}$ durch

$$\varphi(a) = \sum_{\substack{a \geq b \in \mathbb{N}, \\ \gcd(a,b)=1}} 1.$$

Einer der ältesten Algorithmen ist der euklidische Algorithmus aus Euklid von Alexandrias Arbeit „Die Elemente“ (siehe [70]) aus dem Jahr 300 vor Christus.

Satz 1.0.3 (Euklidischer Algorithmus)**.** *Für zwei natürliche Zahlen a und b bestimmen wir folgendermaßen den größten gemeinsamen Teiler:*
Wir berechnen durch Division mit Rest

$$a = q_1 r_0 + r_1$$

mit $r_0 := b$ und in weiterer Folge

$$r_{i-1} = q_{i+1} r_i + r_{i+1}$$

für $i \geq 1$. Dann ist der letzte von Null verschiedene Rest r_{i_0} der größte gemeinsame Teiler von a und b.

Satz 1.0.4 (Fundamentalsatz der Arithmetik)**.** *Für jede natürliche Zahl a gibt es im Wesentlichen eindeutig bestimmte Primzahlen p_i und Primzahlpotenzen $e_i \in \mathbb{N}$ $(1 \leq i \leq k)$, sodass*

$$a = \prod_{i=1}^{k} p_i^{e_i}$$

gilt.

Theorem 1.0.5 (Primzahlsatz)**.** *Für die Primzahlzählfunktion π gilt*

$$\lim_{x\to\infty} \frac{\pi(x)}{\frac{x}{\log(x)}} = 1.$$

Definition 1.0.6. Für einen kommutativen Ring R ist $R[x]$ der **Polynomring in einer Variable über** R. Für ein Polynom $f(x) = c_d x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0 \in R[x]$, mit $c_d \neq 0$, ist $\deg(f) := d$ der **Grad** von f.

Satz 1.0.7 (Fundamentalsatz der Algebra)**.** *Sei $f \in \mathbb{C}[x]$ ein nicht konstantes Polynom, dann besitzt f eine Nullstelle $\alpha \in \mathbb{C}$, das heißt $f(\alpha) = 0$.*

Bemerkung 1.0.8. *Ist $f \in \mathbb{C}[x]$ ein Polynom vom Grad $d \geq 1$, so hat f genau d komplexe Nullstellen, wenn wir die Vielfachheit der Nullstellen mitzählen.*

Definition 1.0.9. Sei $f \in \mathbb{C}[x]$ vom Grad $d \geq 2$ und $\alpha = \alpha_1 \in \mathbb{C}$ eine Nullstelle von f. Sind $\alpha_2, \ldots, \alpha_d \in \mathbb{C}$ die weiteren Nullstellen von f, so nennen wir diese die **Konjugierten von** α.

Definition 1.0.10. Sei $(R, +, \cdot)$ ein kommutativer Ring mit Einselement, dann bezeichnet $(R^*, \cdot)$ (oder nur R^*) die **multiplikative Gruppe** (oder Einheitengruppe) von R, also jene Elemente in R, die das Einselement teilen.

Definition 1.0.11. (a) Sei R ein kommutativer Ring. Ein Element $r \in R \setminus \{0\}$ heißt **Nullteiler**, falls es ein $s \in R \setminus \{0\}$ gibt mit $r \cdot s = 0$.
(b) Ein kommutativer Ring mit Einselement heißt **Integritätsring**, falls er keine Nullteiler besitzt.

Definition 1.0.12. Ein Integritätsring R heißt **faktorieller Ring** (oder **ZPE-Ring**), falls jedes Element $r \in R \setminus \{0\}$ mit $r \notin R^*$ sich im Wesentlichen als eindeutiges und endliches Produkt von irreduziblen Elementen in R schreiben lässt.

2 Der Ring $\mathbb{Z}/n\mathbb{Z}$

Definition 2.0.1. (a) Sei G eine Gruppe und $g \in G$. Die **Ordnung** o_g von g ist die kleinste natürliche Zahl $n \geq 1$ mit $g^n = 1$. Existiert so eine Zahl nicht, setzen wir $o_g = \infty$. Mit $\langle g \rangle := \{g^i \mid i \in \mathbb{Z}\}$ bezeichnen wir die von g **erzeugte Untergruppe**.
(b) Eine Gruppe G heißt **zyklisch**, falls es ein Element $g \in G$ gibt, sodass $\langle g \rangle = G$ gilt. Das Element g nennen wir dann **primitives Element** der Gruppe G.

Satz 2.0.2 (Satz von Lagrange)**.** *Sei G eine endliche Gruppe und U eine Untergruppe von G. Bezeichnet $|G|$ die Anzahl der Elemente von G (auch **(Gruppen-) Ordnung** genannt), so ist $|U|$ ein Teiler von $|G|$.*

Korollar 2.0.3. *(a) In einer endlichen Gruppe teilt die Ordnung jedes Gruppenelements die Gruppenordnung.*
(b) Für jede natürliche Zahl r mit $o_g \mid r$ gilt $g^r = 1$.

Definition 2.0.4. Sei $n \in \mathbb{N}$, dann bezeichnen wir mit $\mathbb{Z}/n\mathbb{Z}$ den **Restklassenring modulo** n und mit $\overline{a}$ oder $[a]_n$ dessen Elemente.

Bemerkung 2.0.5. *Der Ring $\mathbb{Z}/n\mathbb{Z}$ ist genau dann ein Körper, wenn n eine Primzahl ist.*

Definition 2.0.6. Für eine Primzahl p und $r \in \mathbb{N}$ bezeichnen wir den **endlichen Körper** mit $q = p^r$ Elementen mit $\mathbb{F}_q$.

Theorem 2.0.7. *Sei $\mathbb{F}_q$ ein endlicher Körper und $f \in \mathbb{F}_q[x]$ vom Grad > 0, dann gilt*

$$\mathbb{F}_q[x]/\langle f(x) \rangle \text{ ist ein Körper} \iff f(x) \text{ ist irreduzibel über } \mathbb{F}_q$$

Satz 2.0.8 (Chinesischer Restsatz)**.** *Seien $m_1, m_2, \ldots, m_k \in \mathbb{N}$ paarweise teilerfremd, das heißt, es gilt $\gcd(m_i, m_j) = 1 \ \forall i \neq j$. Weiters seien $b_1, b_2, \ldots, b_k \in \mathbb{Z}$ und das System von simultanen Kongruenzen*

$$x \equiv b_1 \bmod m_1, \quad x \equiv b_2 \bmod m_2, \ldots, \ x \equiv b_k \bmod m_k \tag{2.1}$$

gegeben. Setzen wir $M := m_1 \cdot m_2 \cdots m_k$, dann ist eine Lösung $x_0 \in \mathbb{Z}$ von (2.1) gegeben durch

$$x_0 = \sum_{i=1}^{k} b_i \frac{M}{m_i} \left(\left[\frac{M}{m_i} \right]_{m_i} \right)^{-1},$$

wobei $\left(\left[\frac{M}{m_i} \right]_{m_i} \right)^{-1}$ das multiplikative Inverse von $\frac{M}{m_i}$ in $\mathbb{Z}/m_i\mathbb{Z}$ ist. Alle weiteren Lösungen x' von (2.1) erfüllen die Kongruenz

$$x' \equiv x_0 \bmod M.$$

Außerdem gilt die Isomorphie

$$\mathbb{Z}/M\mathbb{Z} \cong \mathbb{Z}/m_1\mathbb{Z} \times \mathbb{Z}/m_2\mathbb{Z} \times \cdots \times \mathbb{Z}/m_k\mathbb{Z}$$

als Ringe.

Bemerkung 2.0.9. *Setzen wir $m_i = p_i^{e_i} \ \forall i$ mit $p_i \in \mathbb{P}$ ($p_i \neq p_j \ \forall i \neq j$) und $e_i \in \mathbb{N}$, so gilt*

$$\mathbb{Z}/\left(p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}\right)\mathbb{Z} \cong \mathbb{Z}/p_1^{e_1}\mathbb{Z} \times \mathbb{Z}/p_2^{e_2}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_k^{e_k}\mathbb{Z}.$$

2.1 Die prime Restklassengruppe $(\mathbb{Z}/n\mathbb{Z})^*$

Definition 2.1.1. Für eine natürliche Zahl n heißt die multiplikative Gruppe $(\mathbb{Z}/n\mathbb{Z})^*$ von $\mathbb{Z}/n\mathbb{Z}$ **prime Restklassengruppe modulo** n.

Satz 2.1.2. *Die prime Restklassengruppe modulo n lässt sich darstellen durch $(\mathbb{Z}/n\mathbb{Z})^* = \{\overline{a} \in \mathbb{Z}/n\mathbb{Z} \mid \gcd(a,n) = 1\}$ und es gilt*

$$\mid (\mathbb{Z}/n\mathbb{Z})^* \mid = \varphi(n).$$

Satz 2.1.3 (Satz von Euler)**.** *Für eine endliche Gruppe G und ein Element $g \in G$ gilt $g^{|G|} = 1$.*

Korollar 2.1.4. *Sei $n \in \mathbb{N}$ und $a \in G = (\mathbb{Z}/n\mathbb{Z})^*$, dann gilt $a^{\varphi(n)} \equiv 1 \bmod n$.*

Ein zentrales Resultat, das wir unter anderem bei Pollards $(p-1)$-Algorithmus (Kapitel 11) und beim RSA-Verfahren (Kapitel 8) benutzen, ist Fermats kleiner Satz.

Satz 2.1.5 (Fermats kleiner Satz)**.** *Sei $a \in \mathbb{Z}$ und $p \in \mathbb{P}$ mit $p \nmid a$, dann gilt*

$$a^{p-1} \equiv 1 \bmod p.$$

Theorem 2.1.6 (Satz von Gauß)**.** *Die prime Restklassengruppe $(\mathbb{Z}/n\mathbb{Z})^*$ ist genau dann zyklisch, falls n die Form $n = 2$, $n = 4$, $n = 2p^e$ oder $n = p^e$ für eine ungerade Primzahl p und $e \in \mathbb{N}^*$ hat.*

3 Zahlkörper

Für das Zahlkörpersieb in Kapitel 17 benötigen wir die folgenden grundlegenden Ergebnisse über Zahlkörper.

Definition 3.0.1. (a) Eine komplexe Zahl α heißt **algebraisch**, falls es ein Polynom $f \in \mathbb{Z}[x]$ gibt, das nicht das Nullpolynom ist, sodass $f(\alpha) = 0$ gilt.
(b) Eine komplexe Zahl α heißt **ganzalgebraisch**, falls es ein normiertes (also mit führendem Koeffizient gleich 1) Polynom $f \in \mathbb{Z}[x]$ gibt, sodass $f(\alpha) = 0$ gilt.

Definition 3.0.2. Sei $\alpha \in \mathbb{C}$ eine algebraische Zahl. Das normierte Polynom $f \in \mathbb{Q}[x]$ mit dem kleinst möglichen Grad, sodass $f(\alpha) = 0$ gilt, heißt **Minimalpolynom** von α und $d =$deg(f) der **Grad** von α.

Definition 3.0.3. Sei $\alpha \in \mathbb{C}$ eine algebraische Zahl. Wir definieren $\mathbb{Q}(\alpha)$ als den kleinsten Körper, der $\mathbb{Q}$ und α enthält.

Satz 3.0.4. *Sei $\mathbb{K} : \mathbb{Q}$ eine Körpererweiterung. Ein Element $\alpha \in \mathbb{K}$ ist genau dann algebraisch, wenn $\mathbb{Q}(\alpha)$ eine endliche Körpererweiterung von $\mathbb{Q}$ ist.*

Definition 3.0.5. Ein **(algebraischer) Zahlkörper** $\mathbb{K}$ ist eine endliche Körpererweiterung von $\mathbb{Q}$ und $[\mathbb{K} : \mathbb{Q}] = d$ bezeichnet den **Grad** von $\mathbb{K}$.

Theorem 3.0.6. *Für jeden Zahlkörper $\mathbb{K}$ gibt es eine algebraische Zahl $\alpha \in \mathbb{K}$ mit*

$$\mathbb{K} = \mathbb{Q}(\alpha).$$

Ist α vom Grad d und $f \in \mathbb{Q}[x]$ das Minimalpolynom von α, so gilt

$$\mathbb{Q}(\alpha) \cong \mathbb{Q}[x]/\langle f(x)\rangle.$$

Weiters bildet die Menge $\{1, \alpha, \ldots, \alpha^{d-1}\}$ eine Basis von

$$\mathbb{Q}(\alpha) = \langle 1, \alpha, \ldots, \alpha^{d-1}\rangle = \left\{ \sum_{i=0}^{d-1} r_i \alpha^i \mid r_i \in \mathbb{Q} \right\}.$$

Definition 3.0.7. Für einen Zahlkörper $\mathbb{Q}(\alpha)$ vom Grad d definieren wir den **Ring ganzalgebraischer Zahlen in** $\mathbb{Q}(\alpha)$ durch

$$\mathcal{O}_\alpha := \{r \in \mathbb{Q}(\alpha) \mid r \text{ ist ganzalgebraisch}\}.$$

Weiters definieren wir $\mathbb{Z}[\alpha]$ als den kleinsten Ring, der $\mathbb{Z}$ und α enthält. Dieser lässt sich schreiben als

$$\mathbb{Z}[\alpha] := \{r_{d-1}\alpha^{d-1} + \cdots + r_1\alpha + r_0 \mid r_i \in \mathbb{Z}\}.$$

Bemerkung 3.0.8. *Für eine quadratfreie ganze Zahl $d \neq 0, 1$ ist der Ring ganzalgebraischer Zahlen $\mathcal{O}_{\sqrt{d}}$ gegeben durch*

$$\mathcal{O}_{\sqrt{d}} = \begin{cases} \mathbb{Z} + \mathbb{Z} \cdot \sqrt{d}, & \text{falls } d \equiv 2, 3 \bmod 4 \\ \mathbb{Z} + \mathbb{Z} \cdot \left(\frac{1+\sqrt{d}}{2}\right), & \text{falls } d \equiv 1 \bmod 4. \end{cases}$$

Bemerkung 3.0.9. *Es gelten die Inklusionen $\mathbb{Z}[\alpha] \subseteq \mathcal{O}_\alpha \subseteq \mathbb{Q}(\alpha)$ und im Allgemeinen keine Gleichheit. Beispielsweise gilt $\mathbb{Z}[\sqrt{d}] \neq \mathcal{O}_{\sqrt{d}} = \mathbb{Z} + \mathbb{Z} \cdot \left(\frac{1+\sqrt{d}}{2}\right)$ für ein quadratfreies $d \neq 0, 1$ mit $d \equiv 1 \bmod 4$.*

Bemerkung 3.0.10. *Der Ring $\mathbb{Z}[\alpha]$ ist im Allgemeinen nicht faktoriell, zum Beispiel bildet $\mathbb{Z}[\sqrt{-5}]$ keinen faktoriell Ring.*

3.1 Idealtheorie

Definition 3.1.1. Sei R ein kommutativer Ring mit Einselement und $x_i \in R$ für $1 \leq i \leq d$, dann heißt

$$\mathcal{I} = \{r_1x_1 + r_2x_2 + \cdots + r_dx_d \mid r_j \in R\}$$

das von $x_1, x_2, \ldots, x_d$ **erzeugte Ideal**, und wir schreiben

$$\mathcal{I} = \langle x_1, x_2, \ldots, x_d \rangle.$$

Gilt $\mathcal{I} = \langle x \rangle$ für ein $x \in \mathcal{I}$, dann heißt $\mathcal{I}$ das von x erzeugte **Hauptideal**.

Definition 3.1.2. Ein Integritätsring heißt **Hauptidealring**, falls jedes Ideal ein Hauptideal ist.

Beispiel 3.1.3. (a) Jeder faktorieller Ring ist ein Hauptidealring, insbesondere die Ringe $\mathbb{Z}$ und $\mathbb{Z}/n\mathbb{Z}$.
(b) Der Ring $\mathbb{Z}[\alpha]$ ist im Allgemeinen kein Hauptidealring.

Definition 3.1.4. (a) Sei R ein kommutativer Ring und $\mathcal{I}$, $\mathcal{J}$ zwei Ideale in R, dann heißt

$$\mathcal{I}\mathcal{J} = \langle x \cdot y \mid x \in \mathcal{I},\ y \in \mathcal{J} \rangle$$

das **Produkt der Ideale** $\mathcal{I}$ und $\mathcal{J}$.
(b) Ein Ideal $\mathcal{I}$ teilt ein Ideal $\mathcal{K}$ und wir schreiben $\mathcal{I} \mid \mathcal{K}$, falls es ein Ideal $\mathcal{J}$ gibt, sodass $\mathcal{I}\mathcal{J} = \mathcal{K}$ gilt.

Definition 3.1.5. Sei R ein kommutativer Ring und $\mathcal{P} \neq \langle 1 \rangle$ ein Ideal in R. Wir nennen $\mathcal{P}$ ein **Primideal in** R, falls für alle Ideale $\mathcal{I}$, $\mathcal{J}$ in R mit $\mathcal{I}\mathcal{J} \subseteq \mathcal{P}$ folgt, dass $\mathcal{I} \subseteq \mathcal{P}$ oder $\mathcal{J} \subseteq \mathcal{P}$ gilt.

Definition 3.1.6. Sei $\mathcal{O}_\alpha$ ein Ring ganzalgebraischer Zahlen und $\mathcal{I} \neq \langle 0 \rangle$ ein Ideal in $\mathcal{O}_\alpha$, dann definieren wir die **Norm** $\mathcal{N}$ vom Ideal $\mathcal{I}$ durch

$$\mathcal{N}(\mathcal{I}) = |\mathcal{O}_\alpha/\mathcal{I}|.$$

Satz 3.1.7. *Die Norm $\mathcal{N}$ ist für alle nichtleeren Ideale endlich.*

Satz 3.1.8. *Die Normfunktion $\mathcal{N}$ ist multiplikativ, das heißt, für zwei nichtleere Ideale $\mathcal{I}, \mathcal{J}$ in $\mathcal{O}_\alpha$ gilt*

$$\mathcal{N}(\mathcal{I}\mathcal{J}) = \mathcal{N}(\mathcal{I})\mathcal{N}(\mathcal{J}).$$

Definition 3.1.9. Ein Integritätsring R heißt **Dedekind Ring**, falls sich jedes echte Ideal $\mathcal{I} \subsetneq R$ im Wesentlichen als eindeutiges Produkt von Primidealen schreiben lässt.

Theorem 3.1.10. *Jeder Ring ganzalgebraischer Zahlen ist ein Dedekind Ring.*

Korollar 3.1.11. *Jedes echte Ideal $\mathcal{I} \subsetneq \mathcal{O}_\alpha$ lässt sich als Produkt von Primidealen schreiben, das heißt*

$$\mathcal{I} = \prod_i \mathcal{P}_i^{f_i}$$

für Primideale $\mathcal{P}_i$ aus $\mathcal{O}_\alpha$ und $f_i \in \mathbb{N}$.

4 Quadratische Reste und das Legendre-Symbol

Ab Kapitel 14, der Faktorisierungsmethode nach Legendre, arbeiten wir eingehend mit der Kongruenz

$$x^2 \equiv a \bmod m, \tag{4.1}$$

wobei m eine natürliche und a eine zu m teilerfremde ganze Zahl ist. Im Allgemeinen hat Gleichung (4.1) keine Lösung x_0, beispielsweise für $a = 3$ und $m = 7$. In diesem Fall nennen wir a einen **quadratischen Nichtrest modulo** m. Andernfalls nennen wir a einen **quadratischen Rest modulo** m, etwa für $a = 2$ und $m = 7$.

Definition 4.0.1. Für $a \in \mathbb{Z}$ und eine ungerade Primzahl p definieren wir das **Legendre-Symbol** $\left(\frac{a}{p}\right)$ durch

$$\left(\frac{a}{p}\right) := \begin{cases} 0, & \text{wenn } p \mid a \text{ gilt} \\ 1, & \text{wenn } a \text{ ein quadratischer Rest modulo } p \text{ ist} \\ -1, & \text{wenn } a \text{ ein quadratischer Nichtrest modulo } p \text{ ist.} \end{cases}$$

Bemerkung 4.0.2. *Für $p = 2$ sind alle Zahlen entweder Vielfache von 2 oder quadratische Reste modulo 2.*

Zum Berechnen des Legendre-Symbols $\left(\frac{a}{p}\right)$ verwenden wir das sogenannte Euler'sche Kriterium.

Satz 4.0.3 (Euler'sches Kriterium)**.** *Sei p eine ungerade Primzahl und $a \in \mathbb{Z}$, dann gilt*

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \bmod p.$$

Korollar 4.0.4. *Für eine ungerade Primzahl p und $a, b \in \mathbb{Z}$ gilt*
(a) $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$, *falls* $a \equiv b \bmod p$ *gilt*
(b) $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right)$
(c) $\left(\frac{ab^2}{p}\right) = \left(\frac{a}{p}\right)$.

Theorem 4.0.5 (Quadratisches Reziprozitätsgesetz)**.** *Seien p und q zwei verschiedene ungerade Primzahlen, dann gilt*

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{(p-1)(q-1)}{4}},$$

oder anders ausgedrückt

$$\left(\frac{p}{q}\right) = (-1)^{\frac{(p-1)(q-1)}{4}}\left(\frac{q}{p}\right).$$

Beweis. Siehe [59], Seiten 253-259 □

Beispiel 4.0.6. Es gilt

$$\left(\frac{13}{67}\right) = (-1)^{198}\left(\frac{67}{13}\right) = \left(\frac{2}{13}\right) \equiv 2^{\frac{13-1}{2}} \equiv -1 \bmod 13,$$

also ist 13 ein quadratischer Nichtrest modulo 67.

Wir wollen im Folgenden mit Hilfe des Legendre-Symbols feststellen, wann a ein quadratischer Rest oder Nichtrest modulo einer beliebigen Zahl m ist.

Lemma 4.0.7. *Sei $a \in \mathbb{Z}$, $m \in \mathbb{N}$ und $m = \prod_{i=1}^{k} p_i^{e_i}$ die Primfaktorzerlegung von m, dann gilt*

$$a \text{ ist ein quadratischer Rest modulo } m \iff a \text{ ist ein quadratischer Rest modulo } p_i^{e_i} \quad \forall i.$$

Lemma 4.0.7 reduziert Gleichung (4.1) von beliebigen Zahlen m auf den Fall der Primzahlpotenzen p^e. Als nächsten Schritt reduzieren wir diesen Fall auf Primzahlen p und unterscheiden dafür zwischen geraden und ungeraden Primzahlen.

Lemma 4.0.8. *Sei $a \in \mathbb{Z}$, p eine ungerade Primzahl und $e \in \mathbb{N}^*$, dann gilt*

$$a \text{ ist ein quadratischer Rest modulo } p^e \iff a \text{ ist ein quadratischer Rest modulo } p.$$

Für $p^e = 2$ liegt genau dann ein quadratischer Rest a modulo 2 vor, wenn $a \equiv 1 \bmod 2$ gilt. Für $p^e = 2^2$ hat Gleichung (4.1) Lösungen, wenn $a \equiv 1 \bmod 4$ gilt. Für $e \geq 3$ formulieren wir folgendes Lemma.

Lemma 4.0.9. *Sei $e \geq 3$, dann sind folgende Aussagen äquivalent:*
(a) a ist ein quadratischer Rest modulo 2^e
(b) a ist ein quadratischer Rest modulo 2^3
(c) $a \equiv 1 \bmod 8$.

Damit haben wir die Lösbarkeit von Gleichung (4.1) für beliebige Zahlen m auf die Primteiler von m reduziert.

Satz 4.0.10. *Eine ganze Zahl a ist genau dann ein quadratischer Rest modulo m, wenn a ein quadratischer Rest modulo jedes ungeraden Primteilers von m ist und*
(a) $a \equiv 1 \bmod 2$ gilt, wenn der Fall $2 \mid m$ und $4 \nmid m$ eintritt
(b) $a \equiv 1 \bmod 4$ gilt, wenn der Fall $4 \mid m$ und $8 \nmid m$ eintritt
(c) $a \equiv 1 \bmod 8$ gilt, wenn der Fall $8 \mid m$ eintritt.

Beispiel 4.0.11. Sei $m = 67335 = 3 \cdot 5 \cdot 67^2$ und $a = 13$. Nach Beispiel 4.0.6 ist 13 ein quadratischer Nichtrest modulo 67 und damit nach Satz 4.0.10 ein quadratischer Nichtrest modulo 67335.

5 Kettenbrüche

In diesem Kapitel behandeln wir die wichtigsten Begriffe über Kettenbrüche für den Kettenbruchalgorithmus aus Kapitel 15.

Definition 5.0.1. Ein **Kettenbruch** ist ein endlicher oder unendlicher Bruch der Form

$$x = a_0 + \cfrac{d_1}{a_1 + \cfrac{d_2}{a_2 + \cfrac{d_3}{a_3 + \cfrac{d_4}{\dots}}}}$$

mit $a_i, d_i \in \mathbb{Z}$. Gilt $d_i = 1\ \forall i$, so heißt der Kettenbruch **regelmäßig** und wir verwenden die Schreibweise

$$x = [a_0; a_1, a_2, a_3, \dots].$$

Wir benötigen für den Kettenbruchalgorithmus ausschließlich regelmäßige Brüche. Eine reelle Zahl x entwickeln wir auf folgende Weise als regelmäßigen Kettenbruch. Wir setzen

$$a_0 := \lfloor x \rfloor$$

als die größte ganze Zahl $\leq x$. Als nächstes definieren wir

$$x_0 := x - a_0$$

als die Nachkommastellen von x, setzen induktiv für $i \geq 1$

$$a_i := \left\lfloor \frac{1}{x_{i-1}} \right\rfloor, \quad x_i := \frac{1}{x_{i-1}} - a_i$$

und erhalten die (regelmäßige) Kettenbruchentwicklung

$$x = [a_0; a_1, a_2, a_3, \dots].$$

Gilt $x_j = 0$ für einen Index j, so bricht die Entwicklung ab und x lässt sich als endlicher Kettenbruch darstellen und ist rational (⇝ Satz 5.0.3):

$$x = [a_0; a_1, a_2, a_3, \dots, a_j] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\dots + \cfrac{1}{a_j}}}}.$$

Wir können die Kettenbruchentwicklung von x auch künstlich abbrechen und erhalten so den j-ten Näherungsbruch

$$\frac{b_j}{c_j} := [a_0; a_1, a_2, a_3, \dots, a_j]$$

von x.

Beispiel 5.0.2. Sei $x = \pi = 3,14159265\dots$. Der 10-te Näherungsbruch von π lautet

$$\pi = [3; 7, 15, 1, 292, 1, 1, 1, 3] = \frac{1146408}{364913} = 3,141592653591$$

und unterscheidet sich erst in der elften Stelle von π.

Beispiel 5.0.2 deutet an, dass sich Kettenbrüche ausgezeichnet für Approximationen irrationaler Zahlen eignen. Wollten wir eine bessere rationale Approximation als den j-ten Näherungsbruch für eine reelle Zahl x, so müsste der Nenner der Approximation zumindest so groß wie c_j sein. Für rationale Zahlen bricht die Kettenbruchentwicklung nach endlich vielen Schritten ab. Doch auch die Umkehrung stimmt.

Satz 5.0.3. *Sei $x \in \mathbb{R}$, dann gilt*

$$x \in \mathbb{Q} \iff x \text{ hat eine endliche Kettenbruchdarstellung.}$$

Satz 5.0.4. *Sei $\frac{b_j}{c_j}$ der j-te Näherungsbruch einer reellen Zahl x, dann gilt*
(a) $\frac{b_0}{c_0} = \frac{a_0}{1}$, $\frac{b_1}{c_1} = \frac{a_0 a_1 + 1}{a_1}$ *und* $\frac{b_j}{c_j} = \frac{a_j b_{j-1} + b_{j-2}}{a_i c_{j-1} + c_{j-2}}$ $\forall j \geq 2$
(b) $b_j c_{j-1} - b_{j-1} c_j = (-1)^{j-1}$ $\forall j \geq 1$.

Satz 5.0.5. *Sei $x \in \mathbb{R}$, $x > 1$ und $\frac{b_j}{c_j}$ dessen j-ter Näherungsbruch, dann gilt*

$$|b_j^2 - x^2 c_j^2| < 2x \quad \forall j.$$

Als Korollar ergibt sich folgende wichtige Aussage für den Kettenbruchalgorithmus (Kapitel 15):

Korollar 5.0.6. *Sei $a \in \mathbb{N}$ kein Quadrat und $\frac{b_j}{c_j}$ der j-te Näherungsbruch von $\sqrt{a}$, dann gilt*

$$|b_j^2 \bmod a| < 2\sqrt{a} \quad \forall j,$$

wobei wir für $b_j^2 \bmod a$ die Restklasse im Intervall $]-\frac{a}{2}, \frac{a}{2}]$ wählen.

Ist $a \in \mathbb{N}$ kein Quadrat, so hat $\sqrt{a}$ eine periodische Kettenbruchentwicklung. Bevor wir den Satz formulieren, definieren wir, was „periodisch“ in diesem Sinn bedeutet.

Definition 5.0.7. Sei $x \in \mathbb{R}$, dann ist die Kettenbruchentwicklung $[a_0; a_1, a_2, a_3, \dots]$ von x **periodisch**, falls ein j_0 und ein s existieren, sodass $a_j = a_{j+s}$ $\forall j \geq j_0$ gilt. Der Index j_0 heißt die **Vorperiode** und s die **Periode** des Kettenbruchs und wir schreiben

$$x = [a_0; a_1, \dots, a_{j_0-1}, \overline{a_{j_0}, a_{j_0+1}, \dots, a_{j_0+s-1}}].$$

Satz 5.0.8. *Sei $a \in \mathbb{N}$ kein Quadrat, dann hat $\sqrt{a}$ die periodische Kettenbruchentwicklung*

$$\sqrt{a} = [a_0; \overline{a_1, \dots a_s, 2a_0}]$$

für eine Periode $s \in \mathbb{N}$.

Beweis. siehe [27], Seite 201 □

6 Elliptische Kurven

Für die Elliptic Curve Method in Kapitel 12 führen wir die notwendigsten Definitionen und Methoden des sehr weitreichenden Feldes der elliptischen Kurven ein. Dieses streckt sich über die Beweisführung von Fermats letztem Satz bis hin zu modernen Verschlüsselungsmethoden. Für tiefer gehende Informationen siehe beispielsweise [73].

6.1 Die projektive Ebene

Sei $\mathbb{K}$ vorerst ein beliebiger Körper.

Definition 6.1.1. Der 2-dimensionale **affine Raum** $\mathbb{A}^2_{\mathbb{K}}$ über $\mathbb{K}$, auch affine Ebene genannt, ist gegeben durch

$$\mathbb{A}^2_{\mathbb{K}} := \{(x, y) \mid x, y \in \mathbb{K}\}.$$

Definition 6.1.2. Seien $x, y, z \in \mathbb{K}$ nicht alle gleich Null. Der 2-dimensionale **projektive Raum** $\mathbb{P}^2_{\mathbb{K}}$ über $\mathbb{K}$, auch projektive Ebene genannt, ist gegeben durch die Menge aller Äquivalenzklassen von Tripel (x, y, z) . Zwei Tripel (x_1, y_1, z_1) und (x_2, y_2, z_2) sind genau dann äquivalent, wenn es ein $\lambda \in \mathbb{K}^* = \mathbb{K} \setminus \{0\}$ gibt, so dass

$$(x_1, y_1, z_1) = \lambda \cdot (x_2, y_2, z_2)$$

gilt. Symbolisch schreiben wir für die Äquivalenz zweier Elemente $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$ und für die Äquivalenzklasse $(x : y : z)$. Also gilt

$$\mathbb{P}^2_{\mathbb{K}} := \{(x : y : z) \mid x, y, z \in \mathbb{K}, \text{ nicht alle Null}\}.$$

Wir können die affine Ebene $\mathbb{A}^2_{\mathbb{K}}$ in natürlicher Weise in die projektive Ebene $\mathbb{P}^2_{\mathbb{K}}$ durch die Abbildung

$$\varphi : \mathbb{A}^2_{\mathbb{K}} \to \mathbb{P}^2_{\mathbb{K}}, \ (x, y) \mapsto (x : y : 1)$$

einbetten. In der projektiven Ebene $\mathbb{P}^2_{\mathbb{K}}$ sind die uns „bekannten“ Punkte, die sogenannten **endlichen Punkte**, von der Form $(x : y : 1) = (xz : yz : z)$ für $z \neq 0$. Für $z = 0$ nennen wir die Punkte $(x, y, 0)$ **unendliche Punkte**.

Definition 6.1.3. Ein Polynom $F(x, y, z)$ heißt **homogen** vom Grad n, falls für alle Terme $ax^i y^j z^k$ mit $a \in \mathbb{K}^*$ des Polynoms F die Summe der Exponenten gleich n ist, also $i + j + k = n$ gilt. Ist dies nicht der Fall, so heißt F **inhomogen**.

Ist ein homogenes Polynom $F(x, y, z)$ vom Grad n gegeben, so können wir leicht zeigen, dass $F(\lambda x, \lambda y, \lambda z) = \lambda^n F(x, y, z)$ für $\lambda \in \mathbb{K}^*$ gilt. Für ein inhomogenes Polynom lässt sich wiederum zeigen, dass dies im Allgemeinen nicht der Fall ist. Betrachten wir die Nullstellenmenge $\{(x : y : z) \in \mathbb{P}^2_{\mathbb{P}} \mid F(x, y, z) = 0\}$ eines homogenen Polynoms $F(x, y, z)$ in der projektiven Ebene $\mathbb{P}^2_{\mathbb{K}}$, so ist diese unabhängig vom Repräsentanten, den wir wählen. Es gilt also für $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$

$$F(x_1, y_1, z_1) = 0 \iff F(x_2, y_2, z_2) = 0.$$

Für inhomogene Polynome ist die Nullstellenmenge in $\mathbb{P}^2_{\mathbb{K}}$ sehr wohl vom Repräsentanten abhängig und dadurch nicht wohldefiniert. Um das zu vermeiden, können wir beliebige Polynome $f(x, y)$ homogenisieren, indem wir alle Terme von f mit passenden Potenzen von z „auffüllen“. Das homogenisierte Polynom $F(x, y, z)$ vom Grad n zu einem Polynom $f(x, y)$ ist

$$F(x, y, z) = z^n f(\frac{x}{z}, \frac{y}{z}), \text{ mit } f(x, y) = F(x, y, 1).$$

6.2 Grundlegendes über elliptische Kurven

Bezeichne $\mathbb{K}$ weiter einen beliebigen Körper. Später interessieren wir uns besonders für endliche Körper, also $\mathbb{K} = \mathbb{F}_q$ (siehe Definition 2.0.6).

Definition 6.2.1. (a) Sei $f \in \mathbb{K}[x, y]$ ein Polynom in 2 Variablen. Die Nullstellenmenge $V(f) := \{(x, y) \in \mathbb{K} \mid f(x, y) = 0\}$ von f heißt **algebraische ebene Kurve** über $\mathbb{K}$.
(b) Sei $F \in \mathbb{K}[x, y, z]$ ein homogenes Polynom. Die Nullstellenmenge $V(F) := \{(x : y : z) \in \mathbb{P}^2_{\mathbb{K}} \mid F(x, y, z) = 0\}$ von F heißt **projektive ebene Kurve** über $\mathbb{K}$.

Beispiel 6.2.2. (a) Sei $f(x, y) = x^2 + y^2 - 1$ und $\mathbb{K} = \mathbb{R}$, dann ist die algebraische ebene Kurve $V(f) = \{(x, y) \in \mathbb{R} \mid x^2 + y^2 = 1\}$ der Kreis in der Ebene.
(b) Für $f(x, y) = y^2 - x^3$ und $\mathbb{K} = \mathbb{R}$ heißt $V(f)$ Neil'sche Parabel.

Definition 6.2.3. Eine algebraische ebene Kurve $V(f)$ über $\mathbb{K}$ ist **glatt** in einem Punkt $(x_0, y_0) \in V(f)$, falls nicht beide partiellen Ableitungen von f in (x_0, y_0) gleichzeitig verschwinden, also

$$\left(\frac{\partial f}{\partial x}(x_0, y_0), \frac{\partial f}{\partial y}(x_0, y_0)\right) \neq (0, 0)$$

gilt. Die Kurve $V(f)$ ist glatt, falls sie in jedem ihrer Punkte glatt ist.

Beispiel 6.2.4. (a) Für $f(x, y) = x^2 + y^2 - 1$ und $\mathbb{K} = \mathbb{R}$ ist $V(f)$ glatt, da

$$\left(\frac{\partial f}{\partial x}(x, y), \frac{\partial f}{\partial y}(x, y)\right) = (2x, 2y) = (0, 0) \iff (x, y) = (0, 0)$$

und $(0, 0) \notin V(f)$ gilt.
(b) Die Neil'sche Parabel $V(f)$ mit $f(x, y) = y^2 - x^3$ und $\mathbb{K} = \mathbb{R}$ ist nicht glatt, da

$$\left(\frac{\partial f}{\partial x}(x, y), \frac{\partial f}{\partial y}(x, y)\right) = (3x^2, 2y) = (0, 0) \iff (x, y) = (0, 0)$$

und $(0, 0) \in V(f)$ gilt.

Definition 6.2.5. Seien $a_i \in \mathbb{K}$ $(i = 1, 2, 3, 4, 6)$ und eine glatte, algebraische ebene Kurve der Form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{6.1}$$

gegeben. Eine Gleichung dieser Art heißt **allgemeine Weierstrass-Gleichung** und die dazugehörige glatte, algebraische ebene Kurve zusammen mit einem unendlich fernen Punkt ∞ (dazu später noch mehr) heißt **elliptische Kurve** $E(\mathbb{K})$ über $\mathbb{K}$ und ist gegeben durch

$$E(\mathbb{K}) = \{(x, y) \in \mathbb{K} \times \mathbb{K} \mid y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\infty\}.$$

Die oben angeführte Gleichung (6.1) für eine elliptische Kurve ist die allgemeinste ihrer Art, kann aber mit einer Voraussetzung durch Transformationen vereinfacht werden.

Definition 6.2.6. Sei $\mathbb{K}$ ein Körper der Charakteristik $\neq 2, 3$. Dann lässt sich (6.1) schreiben als

$$y^2 = x^3 + a_4x + a_6 \tag{6.2}$$

mit $a_4, a_6 \in \mathbb{K}$ und heißt **Weierstrass-Gleichung**. Die elliptische Kurve $E(\mathbb{K})$ ist die dazugehörige glatte, algebraische ebene Kurve zusammen mit dem Punkt ∞ und gegeben durch

$$E(\mathbb{K}) = \{(x, y) \in \mathbb{K} \times \mathbb{K} \mid y^2 = x^3 + a_4x + a_6\} \cup \{\infty\}.$$

Sei ab jetzt $\mathbb{K}$ ein Körper der Charakteristik $\neq 2, 3$ und elliptische Kurven durch die Nullstellenmenge der Weierstrass-Gleichung (6.2) gegeben, wie auch bei der Elliptic Curve Method.

Definition 6.2.7. Sei $V(F)$ mit $F(x, y) = y^2 - x^3 - a_4x - a_6$ eine algebraische ebene Kurve, dann heißt

$$\Delta := 4a_4^3 + 27a_6^2$$

die **Diskriminante** von F.

Um die Glattheit einer Kurve zu überprüfen, verwenden wir folgendes Kriterium:

Satz 6.2.8. *Eine algebraische ebene Kurve $V(F)$ mit $F(x, y) = y^2 - x^3 - a_4x - a_6$ ist genau dann glatt (und damit eine elliptische Kurve), wenn $\Delta \neq 0$ gilt.*

6.3 Die Gruppe $E(\mathbb{K})$

Sei $\mathbb{K}$ weiterhin ein Körper der Charakteristik $\neq 2, 3$.

Definition 6.3.1. Sei $E(\mathbb{K}) : \ y^2 = x^3 + a_4x + a_6$ eine elliptische Kurve über $\mathbb{K}$ und $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2) \in E(\mathbb{K})$ mit $P_1, P_2 \neq \infty$. Für $P_1 + P_2 = P_3 = (x_3, y_3)$ definieren wir folgendermaßen eine Gruppenoperation „+“ auf $E(\mathbb{K})$, auch Addition genannt:

1. Ist $x_1 \neq x_2$, dann gilt

   $$(x_3, y_3) = (m^2 - x_1 - x_2, m(x_1 - x_3) - y_1)$$

   mit $m = \frac{y_2 - y_1}{x_2 - x_1}$.

2. Ist $x_1 = x_2$ und $y_1 = -y_2$, dann gilt $P_3 = \infty$.

3. Ist $P_1 = P_2$ und $y_1 \neq 0$, dann gilt

   $$(x_3, y_3) = (m^2 - 2x_1, m(x_1 - x_3) - y_1)$$

   mit $m = \frac{3x_1^2 + a_4}{2y_1}$.

Gilt $P_i = \infty$ für $i = 1$ und/oder 2, so ist

$$P_3 = P_{1/2} + \infty = P_{1/2}.$$

Nach Definition 6.3.1 ist der Punkt ∞ das neutrale Element der Gruppe $E(\mathbb{K})$. Zur Verdeutlichung der Gruppenaddition betrachten wir diese geometrisch am Beispiel $E(\mathbb{R}) : \ y^2 = x^3 - x$. Wie wir in Abbildung 6.3.2 sehen können, ist die Addition $P_1 + P_2 = P_3$ auf elliptischen Kurven im Allgemeinen (Punkt 1 in Definition 6.3.1) der gespiegelte dritte Schnittpunkt zwischen der elliptischen Kurve und der Verbindungsgerade von P_1 und P_2. Die entsprechenden Formeln aus Definition 6.3.1 lassen sich so herleiten. Die anderen Fälle der Definition stellen wir geometrisch analog dar, beispielsweise Punkt 2 in Abbildung 6.3.3. Beachte, dass in Punkt 3 und in Punkt 2 für $y_1 = y_2 = 0$ die „Verbindungsgerade“ von $P_1 = P_2$ zur Tangente wird.

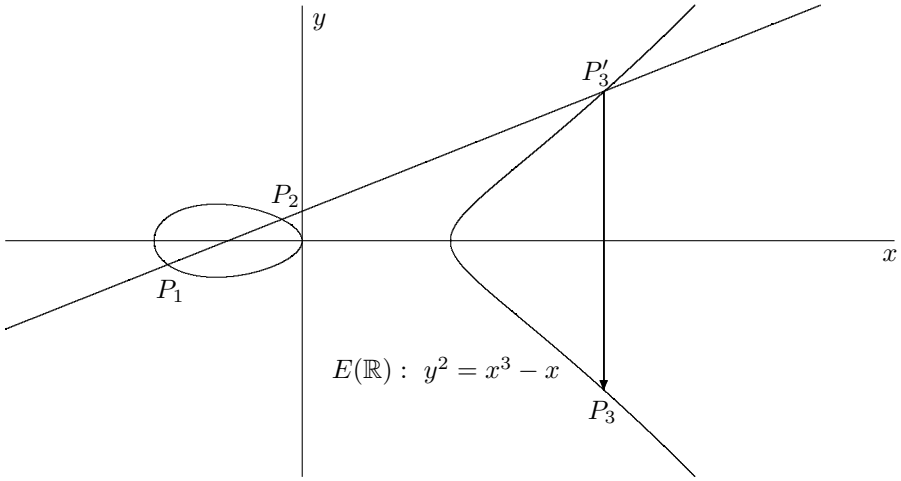

Abbildung 6.3.2: Gruppenverknüpfung im allgemeinen Fall, Punkt 1

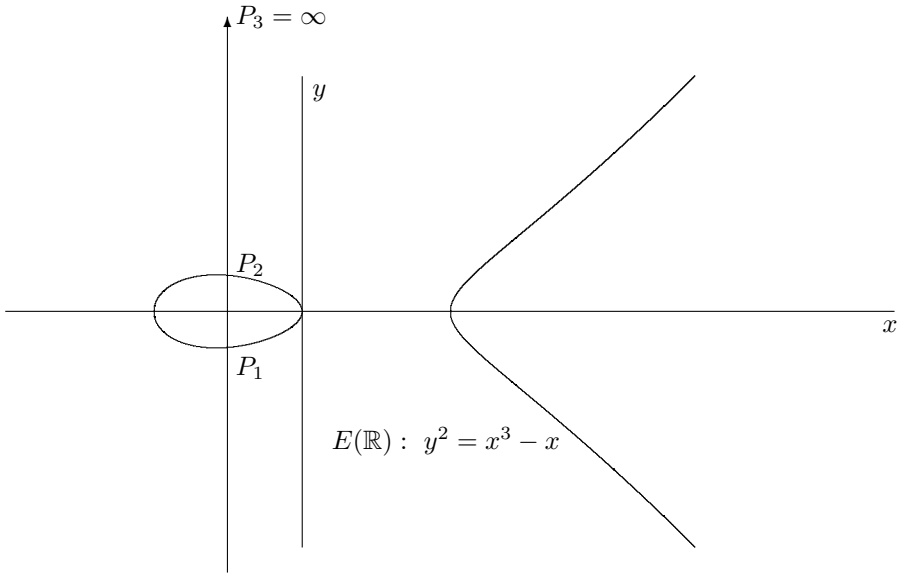

Abbildung 6.3.3: Gruppenverknüpfung, Punkt 2

Satz 6.3.4. *Elliptische Kurven sind kommutative Gruppen mit Einselement ∞.*

Beweis. Siehe [73], Seite 15. □

Bemerkung 6.3.5. *Die Gruppe $E(\mathbb{K})$ ist im Allgemeinen nicht zyklisch. Ist $\mathbb{K} = \mathbb{F}_q$ endlich, so ist $E(\mathbb{F}_q)$ zyklisch oder von der Form*

$$E(\mathbb{F}_q) \cong \mathbb{Z}/d_1\mathbb{Z} \times \mathbb{Z}/d_2\mathbb{Z} \quad (d_1, d_2 \in \mathbb{N})$$

mit $d_1 \mid d_2$ und $d_1 \mid q - 1$.

Kommen wir zurück zum Einselement ∞ der Gruppe. Wir betrachten nach Kapitel 6.1 die elliptische Kurve $E(\mathbb{K})$ als projektive ebene Kurve. Dazu sei $E(\mathbb{K}):\ y^2 = x^3 + a_4x + a_6$ eine elliptische Kurve und $e(x, y) = y^2 - x^3 - a_4x - a_6$ das dazugehörige Polynom. Das homogenisierte Polynom E von e ist

$$E(x, y, z) = y^2z - x^3 - a_4xz^2 - a_6z^3.$$

Wir setzen $z = 0$ (die unendlichen Punkte sind genau von der Form $(x : y : 0)$), also folgt $x = 0$. Da nicht alle Koordinaten in $\mathbb{P}^2_{\mathbb{K}}$ gleich Null sein dürfen, gilt $y \neq 0$. Damit liegt der Punkt $(0 : y : 0) = (0 : 1 : 0) = (0 : -1 : 0)$ auf der elliptischen Kurve $E(\mathbb{K})$. Geometrisch ist das genau der Punkt im Unendlichen der y-Achse, wie in Abbildung 6.3.3 ersichtlich.

6.4 Elliptische Kurven über endlichen Körpern

Sei $\mathbb{K} = \mathbb{F}_q$ ein endlicher Körper der Charakteristik $\neq 2, 3$, $E(\mathbb{F}_q)$ eine elliptische Kurve und $N = |E(\mathbb{F}_q)|$ die Ordnung der elliptischen Kurve. Wie viele Punkte besitzt die Gruppe $E(\mathbb{F}_q)$? Trivialerweise ist die Ordnung aufgrund der Endlichkeit von $\mathbb{F}_q$ endlich. Eine weitere Beobachtung zeigt, dass $E(\mathbb{F}_q)$ auf jeden Fall nicht mehr als $2q + 1$ Elemente besitzt. Warum? Für jeden der q x-Werte gibt es maximal 2 dazu passende y-Werte, da die Wurzel maximal 2 Lösungen in $\mathbb{F}_q$ hat (siehe Satz 14.0.1), also besitzt $E(\mathbb{F}_q)$ höchstens $2q$ Punkte. Der noch fehlende Punkt ist das neutrale Element $\infty \in E(\mathbb{F}_q)$, damit gilt $N \leq 2q + 1$.

Beispiel 6.4.1. Sei $E(\mathbb{F}_5):\ y^2 = x^3 + x + 1$ eine elliptische Kurven, also gilt $N \leq 2 \cdot 5 + 1 = 11$. Rechnen wir nach:

x-Wert	$(x^3 + x + 1)$-Wert	y-Wert	Punkt
0	1	± 1	$(0, 1)$, $(0, 4)$
1	3	–	–
2	1	± 1	$(2, 1)$, $(2, 4)$
3	1	± 1	$(3, 1)$, $(3, 4)$
4	4	± 2	$(4, 2)$, $(4, 3)$
∞	–	∞	∞

Also gilt $N = |E(\mathbb{F}_5)| = 9 \leq 11$.

Ein Resultat für eine schärfere Abschätzung ist das Theorem von Hasse. Dieser Satz verschafft uns nicht nur eine bessere Ober-, sondern auch eine Untergrenze für N.

Theorem 6.4.2 (Theorem von Hasse)**.** *Sei $E(\mathbb{F}_q)$ eine elliptische Kurve und $N = |E(\mathbb{F}_q)|$, dann gilt*

$$| q + 1 - N | \leq 2\sqrt{q}.$$

Beweis. Siehe [73], Seite 91 □

Für Beispiel 6.4.1 erhalten wir nach dem Theorem von Hasse die Abschätzung $2 \leq N \leq 10$. Eine Methode die Ordnung genau zu bestimmen, bietet der Algorithmus von Schoof. Dessen Beschreibung würde aber den Rahmen der Arbeit sprengen. Interessierte können in [73] oder [34] nachlesen.

7 Komplexitätstheorie

In Kapitel III über die Faktorisierungsalgorithmen führen wir Laufzeitanalysen, soweit es im Rahmen der Arbeit möglich ist, durch. Abgesehen davon beziehen wir uns immer wieder auf den Begriff des Algorithmus', den wir an dieser Stelle nicht exakt definieren (müssen). Wir begnügen uns mit der intuitiven Charakterisierung, dass ein Algorithmus eine festgelegte endliche Abfolge von ausführbaren Schritten ist.

Definition 7.0.1. (a) Ein Algorithmus A heißt **deterministisch**, falls jeder der Schritte von A eindeutig festgelegt ist.
(b) Ein Algorithmus A heißt **probabilistisch**, falls A zufällige Wahlen in seinem Ablauf trifft und damit bei gleicher Eingabe unterschiedliche Ausgaben erzeugen kann.

Beispiel 7.0.2. (a) Der euklidische Algorithmus (Satz 1.0.3) und der Probedivisionsalgorithmus (Kapitel 9) sind deterministische Algorithmen.
(b) Pollards ρ-Algorithmus (Kapitel 10), das quadratische Sieb (Kapitel 16) und das Zahlkörpersieb (Kapitel 17) sind probabilistische Algorithmen.

Für alle Algorithmen (abgesehen vom Legendre-Algorithmus) aus Teil III geben wir die (heuristische) Laufzeit an. Doch wie bemessen wir die Laufzeit eines Algorithmus'? Diese soll lediglich vom Eingabewert – genauer gesagt dessen Bit-Anzahl – des Algorithmus' abhängen. Andere Parameter, die auf die reale Rechenzeit Einfluss besitzen, wie die Wahl des verwendeten Computers, die sehr unterschiedliche, sich schnell verändernde Leistungen aufweisen, sowie das verwendete Rechenmodell (beispielsweise der Touringmaschine oder der Registermaschine), die nach der These von Church (siehe [74], Seite 22) alle bis auf einen konstanten Faktor die gleiche Rechenzeit haben und die verwendete Implementierung, sollen keine Rolle für die (theoretische) Laufzeit spielen.

Definition 7.0.3. Sei $A \subseteq \mathbb{R}$ und $f : A \to \mathbb{R}$, $g : A \to \mathbb{R}_{>0} = \{x \in \mathbb{R} \mid x > 0\}$ zwei Funktionen.
(a) Wir sagen „f ist ein groß Oh von g" und schreiben

$$f(x) = O(g(x)),$$

falls es ein $d \in \mathbb{R}_{>0}$ gibt, sodass

$$|f(x)| \leq d \cdot g(x)$$

für ausreichend große $x \in A$ gilt.
(b) Wir sagen „f ist ein klein Oh von g" und schreiben

$$f(x) = o(g(x)),$$

falls

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$$

gilt.

Bemerkung 7.0.4. *In Definition 7.0.3 müsste es exakter $f(x) \in O(g(x))$ (bzw. für $f(x) \in o(g(x))$ analog), statt $f(x) = O(g(x))$ lauten, da $O(g(x)) = \{h : A \to \mathbb{R} \mid \exists\, d \in \mathbb{R}_{>0}$, so dass $|h(x)| \leq d \cdot g(x)$ für ausreichend große $x \in A$ gilt$\}$ eine Menge von Funktionen ist. Der Einfachheit halber bleiben wir bei der Schreibweise von Definition 7.0.3.*

Bemerkung 7.0.5. *Für zwei Funktionen $f, g : A \to \mathbb{R}_{>0}$ gelten die Rechenregeln*
(a) $O(f + g) = O(max\,(f, g))$
(b) $O(f) \cdot O(g) = O(f \cdot g)$

Beispiel 7.0.6. (a) Gilt $f(x) = O(1)$, so ist f beschränkt.
(b) Für $f(x) = x^2 + 5x + 1$ gilt $f(x) = O(x^2 + 5x + 1) = O(\max(x^2, 5x, 1)) = O(x^2)$
(c) Für alle Polynome $f(x)$ gilt $f(x) = O(e^x)$.

Die Laufzeit bzw. Komplexität eines Algorithmus', die wie anfangs erwähnt von der Bit-Anzahl der Eingabe abhängt, definieren wir als die Anzahl der auszuführenden Bit-Operationen im Algorithmus. Als eine Bit-Operation verstehen wir etwa die Addition, die Multiplikation oder Shifts von einzelnen Bits. Beispielsweise benötigt der Additionsalgorithmus zweier natürlicher Zahlen $a, b \in \mathbb{N}$ $(a \geq b)$ $O(n_a)$ Bit-Operationen, wobei n_a die Bit-Anzahl von a ist. Um allgemein die Anzahl der Bits n_a einer natürlichen Zahl a zu berechnen, stellen wir folgende Überlegung an:

$$2^{n_a-1} \leq a < 2^{n_a} \iff \log_2(n_a - 1) \leq \log_2(a) < \log_2(n_a)$$

$$\Rightarrow n_a = O(\log_2(a)),$$

wobei „$\log_2$" den Zweierlogarithmus bezeichnet und „log" den natürlichen Logarithmus zur Basis e. Es gilt weiters

$$\log_2(a) = n_a \iff 2^{n_a} = a \iff n_a \cdot \log(2) = \log(a) \iff n_a = \frac{\log(a)}{\log(2)}$$

$$\Rightarrow n_a = \log_2(a) = O(\log(a)).$$

Die Anzahl der Bit-Operationen der Addition $a + b$ $(a > b)$ beträgt also $O(\log(a))$. Da es bei der Bezeichnung der Bit-Anzahl (n_a vs. $\log(a)$) in der Literatur öfters zu Vermischungen kommt, verwenden wir zur Verbesserung der Übersichtlichkeit ab jetzt ausschließlich n_a für die Bit-Anzahl einer natürlichen Zahl a.

Lemma 7.0.7. *Seien $a, b \in \mathbb{Z}$ und $a > b$. Die Anzahl der Bit-Operationen und damit die Laufzeit*
(a) der Addition und Subtraktion von a und b beträgt $O(n_a)$.
(b) der Multiplikation und Division von a und b beträgt $O(n_a^2)$.
(c) des Vergleichens von a und b beträgt $O(n_a)$.
(d) des euklidischen Algorithmus' (Satz 1.0.3) von a und b beträgt $O(n_a \cdot n_b)$.

Bemerkung 7.0.8. *Der schnellste bekannte Multiplikations- bzw. Divisionsalgorithmus benötigt $O(n_a \log(n_a) \log(\log(n_a)))$ Bit-Operationen.*

In Lemma 7.0.7 haben wir für alle Algorithmen die sogenannte „Worst Case"-Laufzeit angegeben, also die Anzahl an Bit-Operationen, die der Algorithmus maximal benötigt. Da bei gewissen Eingabewerten (z.B. $b = 2$ bei der Multiplikation) eine deutlich geringere Laufzeit möglich ist, kann auch das sogenannte „Average Case"-Szenario von Interesse sein, also die durchschnittliche Laufzeit aller Eingabewerte, die aber in vielen Fällen schwierig zum angeben ist.

Definition 7.0.9. Sei a eine natürliche Zahl mit n_a Bits. Ein Algorithmus A heißt **polynomial** (oder hat polynomiale Laufzeit), falls A eine Komplexität von

$$O\left(n_a^r\right)$$

für ein $r \in \mathbb{R}_{>0}$ hat.

Alle Algorithmen, die eine polynomiale Laufzeit besitzen, darunter sind auch alle Algorithmen aus Lemma 7.0.7, gelten als **effizient**, da die Rechenzeit bei großen Eingabewerten vergleichsweise weniger stark anwächst (siehe Abbildung 7.0.16).

Beispiel 7.0.10. (a) Derzeit gibt es keinen bekannten polynomialen Faktorisierungsalgorithmus.
(b) Seit dem Jahr 2002 gibt es einen effizienten Primzahltest, den sogenannten AKS-Test (siehe [3]).

Definition 7.0.11. Sei a eine natürliche Zahl mit n_a Bits. Ein Algorithmus A heißt **exponentiell** (oder hat exponentielle Laufzeit), falls A eine Komplexität von

$$O\left(r^{f(n_a)}\right)$$

für ein $r \in \mathbb{R}_{>0}$ und ein Polynom $f(n_a)$ hat.

Bei Algorithmen dieser Art ist die Rechenzeit bei Eingabe großer Zahlen sehr lange, weswegen sie als **ineffizient** gelten (siehe Abbildung 7.0.16).

Beispiel 7.0.12. Pollards $(p-1)$-Algorithmus hat eine Laufzeit von $O\left(2^{\frac{n_a}{3}}\right)$ (siehe Kapitel 11.4) und ist damit exponentiell. Ebenso hat der Probedivisionsalgorithmus mit $O(2^{\frac{n_a}{2}} \cdot n_a)$ Bit-Operationen exponentielle Laufzeit.

Definition 7.0.13. Sei a eine natürliche Zahl mit n_a Bits. Ein Algorithmus A heißt **subexponentiell** (oder hat subexponentielle Laufzeit), falls A eine Komplexität von

$$L_a[s;r] := O\left(e^{(r+o(1))n_a^s(\log(n_a))^{1-s}}\right)$$

für ein $r \in \mathbb{R}_{>0}$ und ein reelles $0 < s < 1$ hat.

Bemerkung 7.0.14. *Für $s = 1$ in Definition 7.0.13 ist ein Algorithmus exponentiell, da*

$$L_a[1;r] = O\left(e^{(r+o(1))n_a}\right)$$

gilt. Für $s = 0$ in Definition 7.0.13 ist ein Algorithmus polynomial, da

$$L_a[0;r] = O\left(e^{(r+o(1))\log(n_a)}\right) = O(n_a)$$

gilt.

Subexponentielle Algorithmen sind also eine Art Zwischenstufe von polynomialen und exponentiellen Algorithmen, gelten aber trotz der Geschwindigkeitsvorteile gegenüber exponentiellen Algorithmen als ineffizient.

Beispiel 7.0.15. Zwei der aktuell heuristisch schnellsten Faktorisierungsalgorithmen, das quadratische Sieb mit Laufzeit

$$L_a[\frac{1}{2};1] = O\left(\exp\left((1+o(1))\sqrt{n_a \log(n_a)}\right)\right)$$

und das Zahlkörpersieb mit Laufzeit

$$L_a[\frac{1}{3};\sqrt[3]{\frac{64}{9}}] = O\left(\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right) n_a^{\frac{1}{3}} (\log(n_a))^{\frac{2}{3}}\right)\right),$$

sind subexponentielle Algorithmen und somit ineffizient.

Betrachten wir Abbildung 7.0.16, so sehen wir, warum nur polynomiale Algorithmen als effizient gelten. Bis etwa 57 Bits ist ein Algorithmus mit Laufzeit $O\left(n_a^{10}\right)$ zwar der langsamste der drei Beispielalgorithmen, doch danach wachsen der subexponentielle Algorithmus mit Komplexität $O\left(10 \cdot e^{2,5 \cdot \sqrt{n_a \cdot (\log(n_a))}}\right)$ und der exponentielle Algorithmus mit Komplexität $O\left(2^{n_a}\right)$ innerhalb weniger Bits so stark an, dass der polynomiale Algorithmus im Vergleich nur noch verschwindend kleine Rechenzeit hat. Bei der Zahlenfaktorisierung werden aktuell Zahlen mit über 700 Bits mit den verfügbaren (subexponentiellen) Methoden und großem Aufwand zerlegt, wo ein theoretisches polynomiales Faktorisierungsverfahren nur einen Bruchteil der Zeit benötigen würde. Diese Quantität an Rechenersparnis könnte mit der Verbesserung von Computern in absehbarer Zeit nicht erreicht werden.

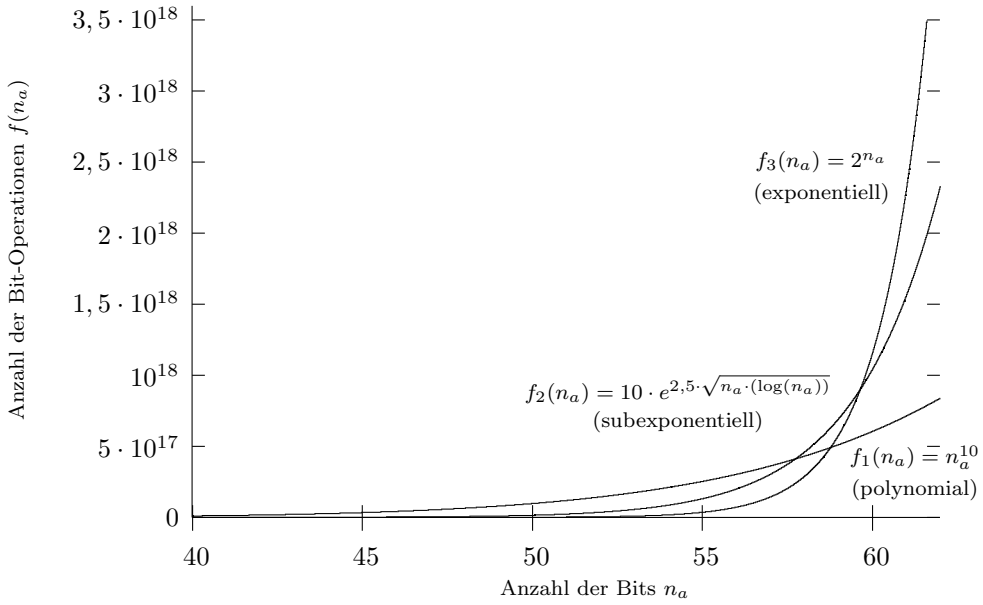

Abbildung 7.0.16: Polynomial vs. exponentiell vs. subexponentiell

7.1 $\mathcal{P}$ vs. $\mathcal{NP}$

Wir betrachten nun nicht mehr einzelne Algorithmen eines Problems, wie dem Faktorisierungsproblem, sondern alle möglichen Lösungsalgorithmen und teilen so die Probleme in Komplexitätsklassen ein. Die beiden bekanntesten Komplexitätsklassen $\mathcal{P}$ und $\mathcal{NP}$ finden sich in einem der Milleniumsprobleme wieder, dem $\mathcal{P} - \mathcal{NP}$-Problem (siehe [29]). Dabei stellt sich folgende Frage:

- Sind die Klassen $\mathcal{P}$ und $\mathcal{NP}$ gleich oder nicht?

Definition 7.1.1. Ein beliebiges Problem A ist genau dann in der Klasse $\mathcal{P}$, wenn es einen deterministischen Algorithmus gibt, der das Problem in polynomialer Zeit löst.

Bemerkung 7.1.2. *Damit für ein Problem $A \in \mathcal{P}$ gilt, reicht es nicht, dass es einen probabilistischen Algorithmus gibt, der A in polynomialer Zeit löst.*

Beispiel 7.1.3. Das Problem des Multiplizierens und Dividierens ist in der Klasse $\mathcal{P}$, weil es nach Lemma 7.0.7 einen deterministischen Lösungsalgorithmus mit Laufzeit $O(n_a^2)$ gibt. Der AKS-Test (siehe [3]) ist ein polynomialer, deterministischer Algorithmus, der feststellt, ob eine Zahl prim ist und damit ist auch das Primzahlproblem in der Klasse $\mathcal{P}$.

Um die Klasse $\mathcal{NP}$ zu definieren, betrachten wir zuvor folgendes Beispiel zur Veranschaulichung:

Beispiel 7.1.4. Wie wir in Teil III sehen, gibt es aktuell keinen deterministischen Algorithmus, der in polynomialer Laufzeit einen Faktor einer ganzen Zahl berechnen kann. Da man nicht alle möglichen Faktorisierungsalgorithmen kennt, kann es trotzdem sein, dass es einen entsprechend effizienten Algorithmus gibt, sodass das Faktorisierungsproblem in der Klasse $\mathcal{P}$ liegt. Wenn wir aber einen Faktor (also eine Lösung des Problems) gegeben haben, ist es ein Leichtes, effizient zu überprüfen, ob es sich wirklich um einen Faktor handelt: Wir führen eine simple Division durch, wenden also einen deterministischen polynomialen Algorithmus an. Genau diese Eigenschaft beschreibt die Klasse $\mathcal{NP}$.

Definition 7.1.5. Ein beliebiges Problem A ist genau dann in der Klasse $\mathcal{NP}$, wenn es für jede Lösung von A einen deterministischen Algorithmus gibt, der in polynomialer Zeit die Korrektheit der Lösung verifiziert.

Bemerkung 7.1.6. *Die Buchstaben „NP“ stehen nicht, wie manchmal irrtümlich angenommen, für „not polynomial“, sondern für „non-deterministic polynomial“.*

Satz 7.1.7. *Es gilt $\mathcal{P} \subseteq \mathcal{NP}$.*

Wie oben kurz erläutert, stellen wir uns die berechtigte Frage: Gilt $\mathcal{NP} = \mathcal{P}$? Die 1 Million Dollar Antwort dieses Milleniumproblems steht seit vielen Jahren aus. Aber was würde sie bedeuten? Für jedes auch noch so komplizierte Problem aus der Klasse $\mathcal{NP}$ gäbe einen polynomialen deterministischen Lösungsalgorithmus, den es vorher erst zu finden gilt. Insbesondere wäre Faktorisieren effizient zu bewältigen und damit beispielsweise viele Methoden der Kryptographie, wie das RSA-Verfahren (Kapitel 8), unbrauchbar. Man vermutet allerdings, dass es recht unwahrscheinlich ist, dass tatsächlich die Gleichheit $\mathcal{NP} = \mathcal{P}$ gilt.

8 Der RSA-Algorithmus

Ein wichtiges Anwendungsgebiet des Faktorisierungsproblems ist die Kryptographie. Weltweit beruhen Millionen wichtiger Datenübertragungen über das Internet auf dem Prinzip des Faktorisierungsproblems, seien es Online-Banking, E-Mail-Verkehr oder Vertragsübermittlungen. Ein wichtiger Vertreter der Verschlüsselungstheorie ist das RSA-Verfahren, das 1978 von Ron Rivest, Adi Shamir und Leonard Adleman ($\rightsquigarrow$ RSA, siehe [61]) entwickelt und patentiert wurde. Das RSA-Verfahren ist ein sogenanntes asymmetrisches Verfahren (oder auch Public-Key-Verfahren), das heißt, ein Benutzer von RSA besitzt einen öffentlichen und einen geheimen, privaten Schlüssel. Mit dem öffentlichen Schlüssel werden Nachrichten codiert und über unsichere Leitungen an den Empfänger gesendet, und nur wer den passenden privaten Schlüssel besitzt, kann die Nachricht lesen. Asymmetrische Verfahren sind sehr nützliche Hilfen in der Kryptographie. Diese haben gegenüber symmetrischen Verfahren – also Verfahren, die zum Ver- und Entschlüsseln den selben Schlüssel verwenden und seit Julius Caesar bekannt sind – den Vorteil, dass Benutzer vor der Verschlüsselung nicht persönlich Schlüssel austauschen müssen und daher problemlos distanzunabhängig codierte Daten übermitteln können.
Mit folgendem Algorithmus sendet Alice mittels RSA eine Nachricht m mit $0 \leq m < n$ in numerischer Form an Bob (üblicherweise werden die Protagonisten in der Kryptographie Alice und Bob genannt):

Algorithmus 8.0.1. (RSA-Algorithmus)

1. Bob wählt zwei verschiedene, große Primzahlen p und q aus, bildet $n = p \cdot q$ und $\varphi(n) = (p-1)(q-1)$.
2. Bob wählt eine zu $\varphi(n)$ teilerfremde Zahl e aus und berechnet die Lösung d der Kongruenz $ed \equiv 1 \bmod \varphi(n)$, also das multiplikative Inverse d zu e in $\mathbb{Z}/\varphi(n)\mathbb{Z}$.
3. Der öffentliche Schlüssel ist das Paar (n, e), das Paar (n, d) der private.
4. Alice besorgt sich den öffentlichen Schlüssel (n, e) von Bob und bildet $c \equiv m^e \bmod n$.
5. Den Geheimtext c schickt Alice über die unsichere Leitung an Bob.
6. Bob bildet $m \equiv c^d \bmod n$ und kann die Nachricht m von Alice lesen.

Satz 8.0.2. *Der RSA-Algorithmus ist korrekt.*

Beweis. Zu zeigen ist $c^d = m^{ed} \equiv m \bmod n$ für $0 \leq m < n$. Aus $ed \equiv 1 \bmod \varphi(n)$ folgt, dass es ein $l \in \mathbb{Z}$ gibt, sodass $ed = 1 + l \cdot \varphi(n)$ gilt. Daraus folgt

$$m^{ed} = m^{1+l\cdot\varphi(n)} = m(m^{p-1})^{l(q-1)} \overset{(1)}{\equiv} m \bmod p.$$

Um Kongruenz (1) zu zeigen, unterscheiden wir die folgenden beiden Fälle:

1. $\gcd(m, p) = 1 \overset{\underset{2.1.5}{Satz}}{\Longrightarrow} m^{p-1} \equiv 1 \bmod p$ ✓
2. $\gcd(m, p) > 1 \Longrightarrow p \mid m$ ✓

Analog gilt für die Primzahl q die Kongruenz $m^{ed} \equiv m \bmod q$. Da $\gcd(p, q) = 1$ gilt, folgt $m^{ed} \equiv m \bmod n$. □

Zur Illustration wollen wir folgendes unrealistisches, aber einfaches Beispiel betrachten.

Beispiel 8.0.3. Alice möchte Bob die Nachricht $m = 1819$ zukommen lassen. Bob wählt $p = 47$ und $q = 59$ und berechnet $n = 47 \cdot 59 = 2773$ und $\varphi(n) = (p-1)(q-1) = 46 \cdot 58 = 2668$. Weiters wählt Bob $e = 17$, sodass $\gcd(e, \varphi(n)) = 1$ gilt und berechnet aus $17d \equiv 1 \bmod 2668$ die Lösung $d = 157$. Das heißt der öffentliche Schlüssel ist $(n, e) = (2773, 17)$ und der private $(n, d) = (2773, 157)$.
Alice besorgt sich den öffentlichen Schlüssel, kann die Geheimnachricht $c \equiv m^e = 1819^{17} \equiv 818 \bmod 2773$ bilden und über eine unsichere Leitung an Bob schicken. Dieser kann mit dem geheimen Schlüssel die Nachricht $m \equiv c^d = 818^{157} \equiv 1819 \bmod 2773$ berechnen.

Ein Angreifer des RSA-Verfahrens hat zwar Zugang zum öffentlichen Schlüssel (n, e) und zum Geheimtext c, kann aber daraus keine direkten Rückschlüsse auf m aufgrund der fehlenden Kenntnis von d ziehen. Das Inverse d vom bekannten e lässt sich wiederum nicht aufgrund der fehlenden Kenntnis von $\varphi(n)$ berechnen. Es wird vermutet, dass die Sicherheit vom RSA-Verfahren äquivalent zum Faktorisierungsproblem ist. Hat man nämlich die Faktorisierung von $n = p \cdot q$ gegeben, so ist es möglich, sich $\varphi(n) = (p-1)(q-1)$, das Inverse d von e modulo $\varphi(n)$ und damit den privaten Schlüssel (n, d) von Bob zu berechnen. Möchten wir etwa als alternativen Ansatz das Inverse d ohne Kenntnis der Faktorisierung von n berechnen, so lässt sich zeigen, dass dies die selbe Komplexität wie das Faktorisierungsproblem hat. Es ist aber möglich, dass es noch einen anderen Zugang zum Brechen des RSA-Algorithmus' gibt, und dies damit nicht äquivalent zum Faktorisierungsproblem ist. Allerdings ist bis dato kein anderer Zugang bekannt, weshalb seit der Einführung des RSA-Verfahrens die Forschung im Bereich der Faktorisierungsalgorithmen forciert wird, um entsprechende Verschlüsselungen zu decodieren.
Bei der Wahl der Parameter p, q und e müssen in der Praxis einige Kriterien für eine sichere Anwendung beachtet werden, da sonst die Nachricht mit relativ einfachen Methoden entschlüsselt werden kann.

- Der Modul n sollte mindestens in der Größenordnung von 1024 Bits gewählt werden, da der aktuelle Faktorisierungsrekord allgemeiner Zahlen bei 756 Bits liegt (Kapitel 20) und für kleinere Moduln eine Faktorisierung möglich ist. Das deutsche Bundesamt für Sicherheit in der Informationstechnik (BSI) empfiehlt seit 2005 sogar eine Größe von 2048 Bits (siehe [14]).
- Die Primzahlen p und q sollten beide etwa in der Größenordnung von 512 Bits gewählt werden, aber nicht zu nah beieinander liegen, da sonst eine Faktorisierung durch die Methode von Fermat (Kapitel 13) möglich ist.
- Die Primzahlen p und q sollten so gewählt sein, dass $p \pm 1$ und $q \pm 1$ jeweils zumindest einen großen Primfaktor besitzen, da sonst mit Pollards $(p-1)$-Verfahren (Kapitel 11) bzw. Williams $(p+1)$-Methode (siehe [27], Kapitel 18) eine Faktorisierung möglich ist.
- Die Zahl e wird üblicherweise aus der Menge $\{3, 17, 65537\}$ gewählt, da dies relativ kleine Fermat-Zahlen sind, also Zahlen der Form $F_n = 2^{2^n} + 1$, und somit effizient modulare Exponentiationen (siehe Anhang C) durchgeführt werden können.
- Die Wahl $e = \frac{1}{2}\varphi(n) + 1$ sollte besonders vermieden werden, da sonst der Geheimtext nach dem Beweis von Satz 8.0.2 die ungünstige Form $c = m^{\frac{1}{2}\varphi(n)+1} = m\left(m^{p-1}\right)^{\frac{1}{2}(q-1)} \equiv m \bmod n$ besitzt.
- Die Nachricht m und die Zahl e sollten nicht gleichzeitig klein gewählt werden, da sonst m^e deutlich kleiner als n ist und die Nachricht durch Wurzelziehen berechnet werden kann.

Werden all diese Punkte bei der Verwendung des RSA-Verfahrens beachtet, so kann man zur Zeit mit ausreichender Sicherheit Daten über das Internet verschicken. Das 1978 entwickelte RSA-Verfahren findet trotz für Computer hohe Alter in der Praxis immer noch Verwendung, wie etwa beim E-Mail-Verschlüsselungsprogramm Pretty Good Privacy (PGP) oder auf Chips von deutschen Reisepässen.

Teil III
Faktorisierungsalgorithmen

Wir kommen nun zum Kernteil der Arbeit, den Faktorisierungsalgorithmen. Unser Ziel ist es bei allen Algorithmen für eine o.B.d.A. natürliche (bei ganzen Zahlen wechseln wir das Vorzeichen) zusammengesetzte Zahl a einen echten Teiler p – also $p \neq 1$ und $p \neq a$ – zu finden. An einer vollständigen Faktorisierung sind wir primär nicht interessiert.
Die Faktorisierungsalgorithmen sind nach den unterschiedlichen Grundideen und Einsatzbereichen gegliedert. Wir haben einerseits den Probedivisionsalgorithmus, die Algorithmen von John Pollard und deren Weiterentwicklungen und andererseits die Algorithmen von Fermat und Legendre und deren Weiterentwicklungen. Zu Beginn behandeln wir die Probedivision (Kapitel 9), die relativ einfach aufgebaut ist und wahrscheinlich die erste Methode zur Zahlenfaktorisierung war. Weiters widmen wir uns den Algorithmen von John Pollard, das sind Pollards ρ-Methode (Kapitel 10) und Pollards $(p-1)$-Methode (Kapitel 11), die beide in den 70er-Jahren entwickelt wurden. Dann lernen wir die Elliptic Curve Method von Hendrik W. Lenstra kennen, die auf der Idee von Pollards $(p-1)$-Methode basiert, aber wesentlich effizienter und Teil der aktuellen Forschung ist. Die Laufzeit all dieser Algorithmen hängt weniger von der zu faktorisierenden Zahl ab als von der Größe der Teiler. Diese sind daher besonders effizient, „kleine“ Teiler von großen Zahlen aufzuspüren.
Anschließend befassen wir uns mit der zweiten großen Kategorie von Algorithmen, die auf der Idee von Fermat und Legendre basieren. Diese beiden Methoden aus dem 17. bzw. Ende des 18. Jahrhunderts – eine Abhandlung findet sich in den Kapiteln 13 und 14 – sind für die Praxis aber weniger interessant. Der chronologisch erste auf Fermat und Legendre aufbauende Algorithmus ist der Kettenbruchalgorithmus (Kapitel 15), der in den 70er-Jahren „state of the art“ war, und mit dem die 7. Fermat-Zahl faktorisiert wurde. Der nächst aktuellere Algorithmus ist das quadratische Sieb (Kapitel 16), das Anfang der 80er-Jahre entwickelt wurde und den Kettenbruchalgorithmus als schnellste Methode ablöste. Abschließend widmen wir uns dem aktuell effizientesten Algorithmus, dem Zahlkörpersieb (Kapitel 17), der ebenfalls auf die Idee von Fermat und Legendre zurückzuführen ist und mit dem bisher eine Vielzahl von Faktorisierungsrekorde aufgestellt wurden. Die letzten beiden Algorithmen markieren in dieser Kategorie den aktuellen Forschungsbereich (mehr dazu in Teil V). Bei dieser Kategorie von Algorithmen ist weniger die Größe der Teiler, sondern die Größe der zu faktorisierenden Zahl für die Rechenzeit entscheidend. Deshalb werden das quadratische Sieb und das Zahlkörpersieb bei Faktorisierungsprogrammen immer zusammen mit Algorithmen zum Berechnen kleinerer Teiler verwendet.
In den Kapiteln 15 bis 17 verfassen wir passend zu den Methoden Pseudoalgorithmen, während wir in den Kapiteln 9 bis 13 auf einen Pseudoalgorithmus verzichten und den Algorithmus als funktionstüchtigen PARI-Code angeben, der selbst ausprobiert werden kann. Die Codes sind nicht dafür optimiert, große Zahlen besonders schnell zu faktorisieren, sondern sind einfach gehalten, um die Ideen der Methoden im Code wieder zu geben. In den Kapiteln über Eigenschaften und Laufzeiten testen wir anhand von Zahlenbeispielen die programmierten Algorithmen auf ihre Geschwindigkeit und filtern heraus, für welche Art von Zahlen die Methoden geeignet sind[1] (für die gesammelten Ergebnisse siehe Anhang E). Das Zeitlimit für die aus Tabelle 9.0.0 entnommenen Zahlen liegt dafür bei 10 Minuten.
Wegen des Umfangs der zahlreichen Primzahltests ist hier nur ein kurzer Anriss möglich. Sie haben den Vorteil, dass sie wesentlich schneller als die heute bekannten Faktorisierungsalgorithmen feststellen können, ob eine natürliche Zahl echte Teiler besitzt oder nicht. Die bedeutendsten Primzahltests sind unter anderem der Sieb des Eratosthenes (siehe [23], Seite 122), der Primzahltest mit elliptischen Kurven (siehe [19], Seite 459), der Primzahltest von Miller-Rabin (siehe [45], Seite 90) und der AKS-Test (siehe [3] oder [23], Seite 200).

[1] mittels eines Intel® Core™ i3-350M 2,26 GHz Prozessors mit 4096 MB Arbeitsspeicher

Zahl	Stellen	Primfaktorzerlegung $p \times q$	Stellen von p	Primfaktorzerlegung von $p-1$
$M_{67} = 2^{67} - 1$	21	$193707721 \times 761838257287$	9	$2^3 \times 3^3 \times 5 \times 67 \times 2677$
$M_{101} = 2^{101} - 1$	31	$7432339208719 \times 341117531003194129$	13	$2 \times 3 \times 101 \times 44029 \times 278557$
$M_{103} = 2^{103} - 1$	32	$2550183799 \times 3976656429941438590393$	10	$2 \times 3 \times 83 \times 103 \times 599$
$M_{109} = 2^{109} - 1$	33	$745988807 \times 870035986098720987332873$	9	$2 \times 107 \times 109 \times 31981$
$M_{137} = 2^{137} - 1$	42	$32032215596496435569 \times$ $543904218360020429 0159$	20	$2 \times 3^2 \times 5 \times 7 \times 47 \times 1099380767$
$M_{139} = 2^{139} - 1$	42	$5625767248687 \times$ $1238761322052083357622784236 01$	13	$2 \times 5^2 \times 13 \times 37 \times 53 \times$ $139 \times 193 \times 457$
$M_{149} = 2^{149} - 1$	45	$866562685662821831 51 \times$ $823510933669084672398 6161$	21	$2 \times 5^2 \times 149 \times 307 \times$ 37888318897441
$M_{1039} = 2^{1039} - 1$	313	$5080711 \times (q_{80} \times q_{227})$	7	$2 \times 3 \times 5 \times 163 \times 1039$
$F_6 = 2^{2^6} + 1$	20	$274177 \times 67280421310721$	6	$2^8 \times 3^2 \times 7 \times 17$
$F_7 = 2^{2^7} + 1$	39	$59649589127497217 \times 5704689200685129054721$	17	$2^9 \times 116503103764643$
$F_8 = 2^{2^8} + 1$	78	$1238926361552897 \times$ $93461639715357977769163558199606 89$ $658405123754163818858028 0321$	16	$2^{11} \times 157 \times 3853149761$
$a_1 = \frac{7^{101}+2}{3 \cdot 13 \cdot 185723 \cdot 4944889067}$	69	$200740742557443271 \times$ $31490507858969420239796238939277$ 45650556299283761521	18	$2 \times 3^2 \times 5 \times 7 \times 13 \times 83 \times$ $131 \times 881 \times 1033 \times 2477$

Tabelle 9.0.0: Testzahlen der Algorithmen

9 Probedivision

Im Englischen wird die Probedivision mit „Trial Division“ bezeichnet. Ist eine Zahl $a \in \mathbb{N}$ gegeben, so arbeitet diese Methode durch stures ausprobieren der Division a durch die Zahlen $2, 3, \ldots, a-1$. Ist eine Division ohne Rest möglich, so ist ein nichttrivialer Teiler gefunden. Eine Verbesserung erzielen wir, indem wir lediglich Zahlen von 2 bis $\lfloor\sqrt{a}\rfloor$ für die Division auswählen, denn ist bis $\sqrt{a}$ kein Teiler gefunden, so tritt auch danach keiner mehr auf. Eine weitere Verbesserung der Probedivision lässt sich durch das Weglassen aller zusammengesetzten Zahlen erreichen, das heißt, wir dividieren nur durch alle Primzahlen von 2 bis $\lfloor\sqrt{a}\rfloor$.

9.1 Der Algorithmus

Wir programmieren die Probedivision mit den oben genannten Vereinfachungen in PARI, führen also nur Divisionen mit Primzahlen bis zur Schranke $\lfloor\sqrt{a}\rfloor$ für eine natürliche Zahl a durch.

Algorithmus 9.1.1. (Probedivision)

```
pd(a)=
{local(p,sr);
 p=2;
 sr=sqrt(a);
 while(p<=sr,
        if(a%p==0,
                print(p," ist ein Teiler von ",a);
                break(5)
                );
        p=nextprime(p++)
        );
 print(a," ist eine Primzahl")
}
```

9.2 Varianten der Probedivision

Es gibt noch weitere alternative Ansätze zu Algorithmus 9.1.1, die den Algorithmus zwar nicht in seiner Geschwindigkeit verbessern, aber in der Praxis sinnvolle Vorteile liefern. Zum einen können wir die Divisionen bei einer sehr großen Zahl a nicht zur Gänze bis zu $\sqrt{a}$ laufen lassen, sondern schon früher den Algorithmus abbrechen, da ab einer gewissen Größe der Zeitaufwand zu groß wird. Hier müssten wir auf effektivere Verfahren zurückgreifen, was den Vorteil hat, dass wir bereits wissen, dass a keine „kleinen“ Primfaktoren besitzt.
Außerdem ist es üblich, dass bei den Divisionen nicht nur Primzahlen zur Probedivision getestet werden. Statt dessen verwenden wir neben der Zahl 2 alle ungeraden Zahlen für die Probedivision, um nicht vorab alle Primzahlen bis $\sqrt{a}$ speichern zu müssen. Eine effizientere Methode ist es an dieser Stelle neben den Primzahlen 2 und 3 alle Zahlen der Form $p = 6k \pm 1$, $k \in \mathbb{N}$ (also $5, 7, 11, 13, \ldots$) zur Division heranzuziehen.

9.3 Laufzeit und Eigenschaften

Ist die Probedivision ein günstiges Verfahren zum Faktorisieren? Wie schnell lässt sich damit ein nichttrivialer Faktor einer Zahl a finden?
Die Probedivision ist ein deterministischer Algorithmus (siehe Definition 7.0.1), der bei gleicher Eingabe a immer den gleichen Teiler (falls es einen gibt) berechnet. Die Laufzeit der Probedivision

ist – wie wir gleich sehen – exponentiell und damit für eine Faktorisierung großer Zahlen nicht geeignet. Ganz außer Acht sollten wir die Probedivision nicht lassen, weil sich kleine Faktoren einer auch großen Zahl a relativ schnell ermitteln lassen und das Verfahren deswegen auch zum Einsatz kommt.
Betrachten wir nun die Laufzeit der Probedivision in Abhängigkeit der Eingabe a. Dividieren wir wie in Algorithmus 9.1.1 mit allen Primzahlen bis $\sqrt{a}$, so benötigen wir hierfür nach Theorem 1.0.5 etwa $\pi(\sqrt{a}) = \frac{\sqrt{a}}{\log(\sqrt{a})}$ Divisionen. Ist n_a die Bit-Länge von a, so gilt $a = O(2^{n_a})$ und nach Lemma 7.0.7 hat eine Division mit Dividend a eine Laufzeit von $O(n_a^2)$. Insgesamt folgt, dass die Laufzeit von Algorithmus 9.1.1

$$\frac{\sqrt{O(2^{n_a})}}{\log(\sqrt{O(2^{n_a})})} \cdot O(n_a^2) = \frac{O(2^{\frac{n_a}{2}})}{O(\frac{n_a}{2}\log 2)} \cdot O(n_a^2) = O(2^{\frac{n_a}{2}}) \cdot \frac{O(n_a^2)}{O(n_a)} = O(2^{\frac{n_a}{2}}) \cdot O(n_a) = O(2^{\frac{n_a}{2}} \cdot n_a)$$

beträgt.
Verwenden wir für die Divisionen neben 2 nur ungerade Zahlen, so kommen wir mit etwa $\frac{\sqrt{a}}{2}$ Divisionen aus und erhalten dementsprechend eine etwas schlechtere Laufzeit. Die Version mit Divisor $p = 2, 3$ und $p = 6k \pm 1$, $k \in \mathbb{N}$ benötigt etwa $\frac{\sqrt{a}}{3}$ Divisionen.
In Tabelle 9.3.1 betrachten wir bei welcher Größe von a der Algorithmus noch Sinn macht und wann die Probedivision keinen Teiler mehr berechnen wird. Am Beispiel M_{1039} erkennen wir, dass hauptsächlich die Größe des kleinsten Primteilers dafür entscheidend ist, wie lange der Algorithmus braucht und nicht die Größe von a selbst.

Zahl	Berechnungszeit
M_{67}	1m26s784ms
M_{101}	×
M_{103}	×
M_{109}	5m31s096ms
M_{137}	×
M_{139}	×

Zahl	Berechnungszeit
M_{149}	×
M_{1039}	2s231ms
F_6	140ms
F_7	×
F_8	×
a_1	×

Tabelle 9.3.1: Zeitaufwand für eine Faktorberechnung mittels Algorithmus 9.1.1

Hat der kleinste Primteiler p von a eine Größe von 6 Dezimalstellen, wie bei F_6, so kann die Probedivision in kürzester Zeit (< 1 Sekunde) einen Teiler berechnen. Bei Zahlen mit einem kleinen 9-stelligen Faktor, wie bei M_{67}, benötigt Algorithmus 9.1.1 bereits 1,5 Minuten und bei Zahlen mit einem großen 9-stelligen Faktor, wie bei M_{109}, steigt die Rechenzeit sprunghaft auf 5,5 Minuten an. In diesem Bereich kommt die Probedivision an ihre Grenzen und kann für Zahlen mit Teilern mit mehr als 9 Dezimalstellen, etwa M_{103}, keinen Faktor in 10 Minuten berechnen.

10 Pollards ρ-Methode

Pollards ρ-Methode – auch unter Monte Carlo-Methode bekannt – wurde nach seinem Entdecker John Pollard, der diesen Algorithmus im Jahre 1975 entwickelte, benannt. Das „ρ“ im Namen verdankt, wie wir noch sehen, die Methode seiner Funktionsweise. Abgesehen von der Probedivision und der Faktorisierung nach Fermat (Kapitel 13), ist Pollards ρ-Methode einer der weniger effektiven Algorithmen.

10.1 Der Algorithmus

Gegeben sei eine ungerade natürliche Zahl a, die einen echten Teiler $p \in \mathbb{N}$ besitzt, den wir nicht kennen. Für den Algorithmus wählen wir eine zufällige Funktion $f : \mathbb{Z}/a\mathbb{Z} \to \mathbb{Z}/a\mathbb{Z}$, also etwa $f \in \mathbb{Z}/a\mathbb{Z}[x]$. Außerdem wählen wir einen ebenfalls zufälligen Startpunkt $x_0 \in \mathbb{Z}/a\mathbb{Z}$ und definieren die Folge

$$x_{i+1} = f(x_i) \text{ für } i \geq 0.$$

Aus der Endlichkeit von $\mathbb{Z}/a\mathbb{Z}$ folgt, dass es zwei Indizes i_0 und j_0 geben muss, sodass $x_{i_0} \equiv x_{j_0} \bmod a$ gilt. Daraus folgt wiederum

$$x_{i_0+k} \equiv x_{j_0+k} \bmod a \quad \forall k \in \mathbb{N}.$$

Wir nehmen weiters an, dass i_0 und j_0 die kleinsten Indizes mit dieser Eigenschaft sind, also die Folge $(x_i)_{i\in\mathbb{N}} \in \mathbb{Z}/a\mathbb{Z}$ periodisch mit der Vorperiode i_0 und der Periode $j_0 - i_0$ ($j_0 > i_0$) ist. Betrachten wir die Folge $(x_i)_{i\in\mathbb{N}}$ nun modulo p (wobei wir den Teiler p noch nicht kennen), dann folgt $\mathbb{Z}/p\mathbb{Z} \subsetneq \mathbb{Z}/a\mathbb{Z}$ aus $p < a$, also hat die Folge $(x_i)_{i\in\mathbb{N}}$ in $\mathbb{Z}/p\mathbb{Z}$ mit großer Wahrscheinlichkeit eine kürzere Periode als in $\mathbb{Z}/a\mathbb{Z}$. Finden wir zwei Folgenglieder $x_i, x_j \in \mathbb{Z}/a\mathbb{Z}$, sodass

$$x_i \equiv x_j \bmod p \text{ und } x_i \not\equiv x_j \bmod a$$

gilt, so können wir den (nichttrivialen) Teiler

$$p = \gcd(x_i - x_j, a)$$

berechnen. Da wir p nicht kennen, berechnen wir nach jedem neuen Folgenglied x_k den größten gemeinsamen Teiler $p = \gcd(x_i - x_j, a)\ \forall\ 0 \leq i < j \leq k$ und hoffen auf einen nichttrivialen Teiler p zu stoßen.

Beispiel 10.1.1. Sei $a = 1633$ und wähle $f(x) = x^4 + x^2 + 1$ und $x_0 = 1$. Dann gilt $x_1 = 3$, $x_2 = 91$, $x_3 = 509$ und $x_4 = 1356$, und wir berechnen $p = \gcd(x_1 - x_2, a) = 1$ und den echten Teiler $p = \gcd(x_1 - x_3, a) = 23$.

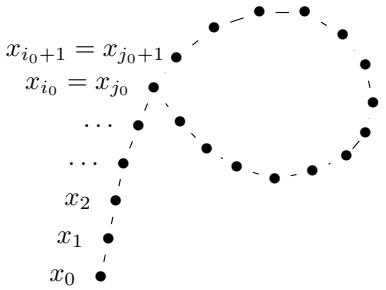

Abbildung 10.1.2: Pollards ρ-Algorithmus

Wenn wir uns die Folge bildlich als Punkte in der Ebene vorstellen, erhalten wir eine Vorperiode als aufsteigenden Ast und direkt im Anschluss die Periode in Kreisform. Zusammen ergibt dies den griechischen Buchstaben ρ, wie in Abbildung 10.1.2 zu sehen ist. Bevor wir den Algorithmus als PARI-Code verfassen, betrachten wir eine Variante der vorgestellten Version.

10.2 Varianten der Pollard ρ-Methode

Bei sehr großen Werten von a wird die Chance auf eine Übereinstimmung zweier Folgenglieder immer geringer, das heißt, um einen Treffer $x_i \equiv x_j \bmod p$ zu landen, müssen wir unter Umständen sehr viele Folgenglieder berechnen. Dies führt wiederum zu einem sehr hohem Speicheraufwand der Folge $(x_k)_{k\in\mathbb{N}}$ und einem hohen Rechenaufwand durch die Berechnungen $p = \gcd(x_i - x_j, a)\ \forall\ 0 \leq i < j \leq k$. Durch die Verwendung von Zykel-Algorithmen können wir diesen Umstand etwas eindämmen.

10.2.1 Floyds Zykel-Algorithmus

Bei Floyds Zykel-Algorithmus – auch bekannt unter dem Namen Hase-Igel-Methode – verwenden wir neben der ersten Folge $(x_i)_{i\in\mathbb{N}}$ zusätzlich eine zweite Folge $(y_i)_{i\in\mathbb{N}}$, definiert durch $y_i := x_{2i}$. Damit läuft die Folge $(y_i)_{i\in\mathbb{N}}$ doppelt so schnell wie die Folge $(x_i)_{i\in\mathbb{N}}$ und wir berechnen $p = \gcd(x_i - y_i, a)$ lediglich für den aktuellen Index i. Durch Floyds Zykel-Algorithmus ersparen wir uns die Speicherung der vorhergehenden Folgenglieder und auch die meisten Berechnungen des größten gemeinsamen Teilers.

Algorithmus 10.2.1. (Pollards ρ-Algorithmus)[2]

```
pollrho(a,pol)=
{local(p,x,y);
 p=1;
 x=y=random(a);
 while(p==1||p==a,
         x=poleval(pol,x)%a;
         y=poleval(pol,y)%a;
         y=poleval(pol,y)%a;
         p=gcd(x-y,a);
         if(p>1&&p<a,
                 print(p," ist ein Teiler von ",a);
                 break(5)
                 )
         )
}
```

Beispiel 10.2.2. Sei $a = 5513$, $f(x) = x^2 + 1$ und $x_0 = 2088$. Dann gilt $x_1 = 4475$, $y_1 = x_2 = 2410$, $y_2 = x_4 = 751$, $x_3 = 2912$, $y_3 = 2860$ und $y_4 = 3748$. Daraus folgt $p = \gcd(x_i - y_i, a) = 1$ für $i \leq 3$ und $p = \gcd(x_4 - y_4, a) = 37$. Also gilt $a = 37 \cdot 149$.

Wir können noch weitere gcd-Berechnungen einsparen, wenn wir nicht bei jedem Schritt $p = \gcd(x_i - y_i, a)$ berechnen, sondern die ersten k Folgengliederdifferenzen multiplizieren und danach erst mit a den größten gemeinsamen Teiler berechnen, also $p = \gcd((x_1 - y_1) \cdot (x_2 - y_2) \cdots (x_k - y_k), a)$. Das Ergebnis wird, außer wir haben Pech und im Produkt stecken alle Teiler von a, wieder einen echten Teiler p liefern.
Eine weitere Verbesserung bringt ein Ansatz von Richard P. Brent in seiner Arbeit „An Improved Monte Carlo Factorization Algorithm“ (siehe [7]). Er nimmt statt des Zykel-Algorithmus' von Floyd

[2] Der PARI-Code für die Funktion „poleval()“ befindet sich in Anhang B.5

den nach sich selbst benannten Zykel-Algorithmus von Brent, der einen Geschwindigkeitsvorteil (etwa 25% schneller) gegenüber der ursprünglichen Variante bringen soll.

10.3 Laufzeit und Eigenschaften

Für die Herleitung der Laufzeit halten wir fest, dass wir durchschnittlich $O(\sqrt{p})$ mal aus einer p-elementigen Menge zwei beliebige Elemente ziehen müssen, bis wir eine Übereinstimmung erhalten. Nehmen wir $p \approx \sqrt{a}$ an, so müssen wir $O(\sqrt{p}) = O(a^{\frac{1}{4}})$ Folgenglieder x_i und y_i berechnen, bis die Kongruenz $x_i \equiv y_i \bmod p$ erfüllt wird. Nach jedem Schritt berechnen wir in Algorithmus 10.2.1 den größten gemeinsamen Teiler mit Laufzeit $O(n_a^2)$ (siehe Lemma 7.0.7 (d)). Somit ist Pollards ρ-Methode ein probabilistischer Algorithmus mit exponentieller Laufzeit

$$O\left(2^{\frac{n_a}{4}} \cdot n_a^2\right).$$

Wichtige Parameter für Algorithmus 10.2.1 sind neben der Größe des Primteilers p von a die Wahl des Polynoms $f(x)$, das die Folgen $(x_i)_{i\in\mathbb{N}}$ und $(y_i)_{i\in\mathbb{N}}$ definiert. Lineare Polynome wären aufgrund ihrer Einfachheit die erste Wahl, eignen sich aber nicht, da sie besonders lange Perioden für die Folge $(x_i)_{i\in\mathbb{N}}$ erzeugen. Beispielsweise konnte $f(x) = 2x + 1$ keinen einzigen Teiler für Zahlen aus Tabelle 9.0.0 berechnen. Optimaler sind quadratische Polynome $f(x) = x^2 + b$ mit $b \neq 0, -2$, also beispielsweise $f(x) = x^2 + 1$. Zum Vergleich sehen wir die Unterschiede in der Berechnungszeit für unterschiedliche Polynome in Tabelle 10.3.1.

	Berechnungszeit mit dem Polynom		
Zahl	$f(x) = x^3 + x^2 + 1$	$f(x) = x^2$	$f(x) = x^2 + 1$
M_{67}	256ms	499ms	265ms
M_{101}	47s315ms	×	30s248ms
M_{103}	1s014ms	×	484ms
M_{109}	609ms	17s706ms	327ms
M_{137}	×	×	×
M_{139}	31s357ms	×	29s593ms
M_{149}	×	×	×
M_{1039}	421ms	3s650ms	78ms
F_6	31ms	0ms	31ms
F_7	×	×	×
F_8	5m7s493ms	×	3m6s687ms
a_1	×	×	×

Tabelle 10.3.1: Zeitaufwand für eine Faktorberechnung mittels Algorithmus 10.2.1

Tabelle 10.3.1 zeigt, dass die Rechenzeit von Pollards ρ-Methode wie bei der Probedivision von der Größe des kleinsten Primfaktors p von a abhängt. Beispielsweise wird mit dem Polynom $f(x) = x^2+1$ für die 313-stellige Zahl M_{1039} ähnlich schnell der 7-stellige Teiler berechnet (78ms) wie für die 20-stellige Zahl F_6 der 6-stellige Teiler (31ms). Unter einer Minute werden mit dem Algorithmus bis zu 13-stellige Faktoren gefunden, wie etwa bei M_{101}. Bei guter Wahl des Polynoms werden auch 16-stellige Teiler berechnet, bei F_8 etwa in 3 Minuten. Für größere Teiler, wie dem 17-stelligen Teiler von F_7, ist Algorithmus 10.2.1 aber zu ineffizient und gibt in 10 Minuten kein Ergebnis mehr aus.

11 Pollards $(p-1)$-Algorithmus

Pollards $(p-1)$-Methode – ein Jahr vor dem ρ-Algorithmus 1974 von John Pollard entwickelt – besitzt eine ähnliche Laufzeit wie sein Bruder. Interessanter macht ihn die Tatsache, dass er als Basis für einen der aktuell schnelleren Algorithmen dient, der Elliptic Curve Method (Kapitel 12), und dass er für eine ganz spezielle Art von Zahlen sehr wirksam ist.

Definition 11.0.1. Sei B eine natürliche Zahl.
(a) Eine natürliche Zahl n heißt B**-glatt**, falls alle Primfaktoren von n nicht größer als B sind. Das heißt, aus $n = \prod_{i=1}^{k} p_i^{e_i}$ folgt $p_i \leq B \ \forall i$.
(b) Eine natürliche Zahl n heißt B**-potenzglatt**, falls alle Primfaktorpotenzen von n nicht größer als B sind. Das heißt, aus $n = \prod_{i=1}^{k} p_i^{e_i}$ folgt $p_i^{e_i} \leq B \ \forall i$.

11.1 Der Algorithmus

Gegeben sei eine natürliche Zahl a und es sei $a = \prod_{i=1}^{k} p_i^{e_i}$ dessen Primfaktorzerlegung, die wir nicht kennen. Wir wählen eine natürliche Zahl B und eine zufällige Zahl $b \in (\mathbb{Z}/a\mathbb{Z})^*$ und berechnen das kleinste gemeinsame Vielfache $V = \operatorname{lcm}(1, 2, 3, \ldots, B)$ aller Zahlen $\leq B$. Nach dem kleinen Satz von Fermat (Satz 2.1.5) gilt für alle Primteiler p_i von a

$$b^{p_i - 1} \equiv 1 \bmod p_i.$$

Ist für eine der Primzahlen p_i von a die Zahl $p_i - 1$ B-potenzglatt, so folgt die Kongruenz $b^V \equiv 1 \bmod p_i$, da V ein Vielfaches von $p_i - 1$ ist. Daraus folgt $p_i \mid b^V - 1$ und damit $p_i \mid \gcd(b^V - 1, a)$, womit ein Teiler von a gefunden wäre, falls $\gcd(b^V - 1, a) \neq a$ gilt.
Da wir nicht wissen, welche der Primteiler von a B-potenzglatt sind, berechnen wir

$$p = \gcd(b^V - 1, a)$$

und hoffen, dass wir auf einen echten Teiler p stoßen. Ein wichtiger Parameter ist dabei die Schranke B, denn ist B zu klein gewählt, so ist die Wahrscheinlichkeit, dass einer der um eins verringerten Teiler von a B-potenzglatt ist, zu gering. Ist B zu groß gewählt, so wird der Wert V zu groß und der Rechenaufwand steigt rasant an, beispielsweise gilt $V = 2520$ für $B = 10$, aber bereits $V = 69720375229712477164533808935312303556800$ für $B = 100$ und $V \approx 7 \cdot 10^{432}$ für $B = 1000$. Hat a außerdem eine ungünstige Gestalt bei einem zu großen Wert B und sind alle Werte $p_i - 1$ B-potenzglatt, so gilt $\gcd(b^V - 1, a) = a$ (siehe Satz 11.2.1). Eine Verbesserung bietet eine der Varianten von Pollards $(p-1)$-Methode aus Kapitel 11.2.

Beispiel 11.1.1. Sei $a = 4223$, $B = 5$ und $b = 2$ $(\gcd(a, b) = 1)$ und es gilt

$$V = \operatorname{lcm}(2, 3, 4, 5) = 60 \Rightarrow b^V = 2^{60} - 1 \equiv 820 \bmod 4223 \Rightarrow \gcd(820, 4223) = 41$$

und wir haben die Zerlegung $a = 41 \cdot 103$ gefunden.

Die „Basis-Version“ des $(p-1)$-Algorithmus' ist als PARI-Code in Anhang B.1 zu finden.

11.2 Varianten des Pollard $(p-1)$-Algorithmus'

Bisher haben wir zuerst das kleinste gemeinsame Vielfache V berechnet und dann direkt $b^V - 1$. Das kann bei großem V zum einen sehr rechenintensiv sein und zum anderen könnte bei schlechter Wahl von B und ungünstiger Gestalt von a $\gcd(b^V - 1, a) = a$ gelten.

Satz 11.2.1. *Seien $a, b, B \in \mathbb{N}$ mit $\gcd(a, b) = 1$ und $V = lcm(2, 3, \ldots, B)$. Angenommen $a = p_1 \cdot p_2 \cdots p_k$ ist die Primfaktorzerlegung von a mit $p_i \neq p_j \ \ \forall i \neq j$ und es seien alle $p_i - 1$ B-potenzglatt, dann gilt*

$$\gcd(b^V - 1, a) = a$$

Beweis. Sei p_i ein beliebiger Primteiler von a, dann ist $p_i - 1$ B-potenzglatt und $p_i - 1$ teilt V. Aus dem kleinen Satz von Fermat (Satz 2.1.5) folgt $b^V = 1 \bmod p_i$, also gilt $p_i \mid b^V - 1$ für alle Primteiler p_i von a. Da wir $\gcd(p_i, p_j) = 1$ vorausgesetzt haben, folgt $b^V = 1 \bmod a$ und $\gcd(b^V - 1, a) = a$. □

Beispiel 11.2.2. Für $a = 77$, $b = 2$ und $B = 10$ gilt $\gcd(b^V - 1, a) = 77$. Wir haben keinen echten Teiler von a gefunden, weil $a = 7 \cdot 11$ quadratfrei ist und $7 - 1 = 2 \cdot 3$ und $11 - 1 = 2 \cdot 5$ beide 10-potenzglatt sind, also muss nach Satz 11.2.1 $\gcd(b^V - 1, a) = 77$ gelten.

Bemerkung 11.2.3. *Ist a nicht quadratfrei, so lässt sich im Allgemeinen bei jeglicher Größe von B keine Aussage über das Ergebnis von $p = \gcd(b^V - 1, a)$ treffen.*

Alternativ betrachten wir die Primfaktorzerlegung von $V = \prod_{j=1}^{l} q_j^{f_j}$ (o.B.d.A. $q_j < q_{j+1}$), deren Primteiler q_j genau die Primzahlen aus dem Intervall $[1, B]$ und deren Potenzen f_j maximal sind, sodass $q_j^{f_j} \leq B$ gilt ($\Rightarrow f_j = \log_{q_j}(B)$, der q_j-Logarithmus von B). Berechnen wir nicht unmittelbar b^V, sondern iterativ $b \equiv b^{q_j^{f_j}} \bmod a$ für $j \geq 1$ und nach jedem (bzw. jedem j-ten) Schritt des Potenzierens $p = \gcd(b - 1, a)$ (Bemerkung: b hat nach $m < l$ Schritten die Form $b^{\prod_{j=1}^{m} q_j^{f_j}}$), so kann der Algorithmus einerseits mit großer Wahrscheinlichkeit früher terminieren, andererseits wird das Ergebnis $\gcd(b - 1, a) = a$ unwahrscheinlicher.

Beispiel 11.2.4. Sei wie in Beispiel 11.2.2 $a = 77$, $b = 2$ und $B = 10$. Benutzen wir die angesprochene Variante, so potenzieren wir zuerst mit der ersten Primzahlpotenz $2^3 \leq 10$, also $b \equiv b^{2^3} \equiv 25 \bmod 77$ und berechnen $p = \gcd(b - 1, a) = \gcd(24, 77) = 1$. Die nächste Primzahlpotenz ist $3^2 \leq 10$ und wir erhalten $b \equiv b^{3^2} \equiv 25^{3^2} \equiv 15 \bmod 77$ und es folgt $p = \gcd(b-1, a) = \gcd(14, 77) = 7$.

11.2.1 Variante mit großer Primzahl

Angenommen es gilt $\gcd(b^V - 1, a) = 1$, so definieren wir neben der Schranke $B_1 := B$ eine zweite Schranke $B_2 > B_1$, die es den Primteilern p_i von a erlaubt, dass $p_i - 1$ nicht mehr „ganz“ B_1-potenzglatt ist. Nicht ganz potenzglatt bedeutet, dass einer der Primfaktoren q von $p_i - 1$ zwischen B_1 und B_2 liegen darf und wir $p_i - 1 = r \cdot q$ mit r B_1-potenzglatt und $q \in]B_1, B_2] \cap \mathbb{P}$ schreiben können. Diese Methode wird Variante mit großer Primzahl oder auch Big Prime Variation genannt. Nachdem wir nicht wissen, welche Gestalt die Primteiler von a haben, gehen wir folgendermaßen vor. Am Ende der Basis-Version des $(p-1)$-Algorithmus' erhalten wir das Ergebnis $b_0 := b^V \bmod a$. Daraus berechnen wir $b \equiv b_0^{q_1} \bmod a$ für die erste Primzahl q_1 aus dem Intervall $]B_1, B_2]$ und für alle weiteren Primzahlen $q_i \in]B_1, B_2]$ (o.B.d.A. $q_i < q_{i+1}$, $i \geq 1$) die Differenzen $d_{i+1} := q_{i+1} - q_i$. Jetzt ermitteln wir iterativ für $i \geq 2$

$$b \equiv b \cdot b_0^{d_i} \equiv b_0^{q_1 + d_2 + d_3 + \cdots + d_{i-1}} \cdot b_0^{d_i} \equiv b_0^{q_1 + d_2 + d_3 + \cdots + d_i} \equiv b_0^{q_i} \bmod a$$

und $\gcd(b - 1, a)$. Gilt für einen Primteiler p von a wie oben $p - 1 = r \cdot q_i$, so erhalten wir im i-ten Schritt den Teiler p.

Beispiel 11.2.5. Sei $a = 4399$, $b = 2$ und $B = B_1 = 10 \Rightarrow b^V \equiv 2928 \bmod 4399$ und $p = \gcd(b^V - 1, a) = 1$. Wir setzen $b_0 = b^V = 2928$ und wählen $B_2 = 20$. Daraus folgt

$$q_1 = 11,\ q_2 = 13,\ q_3 = 17,\ q_4 = 19 \text{ und } d_2 = q_2 - q_1 = 2,\ d_3 = q_3 - q_2 = 4,\ d_4 = q_4 - q_3 = 2.$$

Wir berechnen $b \equiv b_0^{q_1} \equiv 2928^{11} \equiv 16 \bmod 4399$ und $p = \gcd(b - 1, a) = \gcd(15, 4399) = 1$. Weiters berechnen wir $b \equiv b \cdot b_0^{d_2} \equiv 16 \cdot 2928^2 \equiv 1326 \bmod 4399$ und $p = \gcd(b-1, a) = \gcd(1325, 4399) = 53$. Tatsächlich gilt $p - 1 = 53 - 1 = 52 = 2^2 \cdot 13 = r \cdot q$ mit $r = 2^2$ 10-potenzglatt und $q = 13 \in]10, 20]$.

Bemerkung 11.2.6. *Die Big Prime Variation ist nur erfolgreich, wenn genau ein Primteiler q von $p - 1$ nicht B_1-potenzglatt ist. Sind zwei oder mehr Primzahlen oder auch eine Primzahlpotenz von $p - 1$ nicht B_1-potenzglatt, so wird dieses Verfahren fehlschlagen.*

Algorithmus 11.2.7. (Erweiterter Pollard $(p-1)$-Algorithmus)[3]

```
pollp1var(a,B1,B2,j)=                          /*j = Frequenz der gcd-Berechnungen*/
{local(p,b,k,q,q1,q2,d);
 b=1;
 while(b==0||b==1,
        b=random(a)
        );
 p=gcd(b,a);
 k=1;
 q=2;
 if(p>1,
        print(p," ist ein Teiler von ",a);
        break(3)
        );
 while(q<=B1,
        b=lift(Mod(b,a)^(q^floor(logp(q,B1))));
        if(k%j==0,
                p=gcd(b-1,a);
                if(p>1&&p<a,
                        print(p," ist ein Teiler von ",a);
                        break(5)
                        )
                );
        q=nextprime(q++);
        k++
        );
 p=gcd(b-1,a);
 if(p>1&&p<a,
        print(p," ist ein Teiler von ",a)
        );
 if(p==a,
        print("B1 = ",B1," ist zu gross fuer ",a);
        break(3)
        );
 if(p==1,
        q1=q;
        q2=nextprime(q++);
        b0=b;
        b=(b0^q1)%a;
        p=gcd(b-1,a);
        if(p>1&&p<a,
                print(p," ist ein Teiler von ",a)
                );
        while(q2<=B2,
                d=q2-q1;
                b=b*b0^d%a;
                p=gcd(b-1,a);
                if(p>1&&p<a,
                        print(p," ist ein Teiler von ",a);
```

[3]Der PARI-Code für die Funktion „logp()“ befindet sich in Anhang B.4

```
                        break(5)
                        );
                q1=q2
                q2=nextprime(q2++)
                );
        );
 if(p==1,
        print("B1 = ",B1," und B2 = ",B2," sind zu klein fuer ",a)
        );
 if(p==a,
        print("B2 = ",B2," ist zu gross fuer ",a)
        )
}
```

11.3 Die Wahl von b

Ist die Wahl von $b \in (\mathbb{Z}/a\mathbb{Z})^*$ mitentscheidend für den Ausgang des $(p-1)$-Algorithmus'?
Wählen wir beispielsweise $a = 2047 = 23 \cdot 89$, so findet Algorithmus 11.2.7 für $b = 3$, $B_1 = 12$, $B_2 = 12$ keinen Teiler von a. Wählen wir aber $b = 12$, so berechnet der Algorithmus den Teiler $p = 89$. Der Grund liegt in den unterschiedlichen Ordnungen o_b von b in $(\mathbb{Z}/23\mathbb{Z})^*$ bzw. $(\mathbb{Z}/89\mathbb{Z})^*$. Nach Tabelle 11.3.1 hat $b = 3$ sowohl in $(\mathbb{Z}/23\mathbb{Z})^*$ als auch in $(\mathbb{Z}/89\mathbb{Z})^*$ eine Ordnung, die 12-glatt

b	o_b in $(\mathbb{Z}/23\mathbb{Z})^*$	o_b in $(\mathbb{Z}/89\mathbb{Z})^*$
3	11	88
12	11	8

Tabelle 11.3.1: Ordnungen von b in $(\mathbb{Z}/23\mathbb{Z})^*$ bzw. in $(\mathbb{Z}/89\mathbb{Z})^*$

ist und in deren Primfaktorzerlegung der größte Primteiler 11 vorkommt. Für $b = 12$ sind auch beide Ordnungen 12-glatt, doch in $(\mathbb{Z}/89\mathbb{Z})^*$ hat 12 eine Ordnung, in der nicht der Primteiler 11 auftritt, weswegen Algorithmus 11.2.7 hier ein Ergebnis liefert.
Die Wahl von b kann also mitentscheidend für den Ausgang des Algorithmus' sein. Das Problem ist aber, dass wir keinen der Teiler von a kennen und damit die Wahl von b nicht günstig beeinflussen können.

11.4 Laufzeit und Eigenschaften

Wie die beiden vorangegangenen Algorithmen ist auch Pollards $(p-1)$-Methode ein exponentieller Algorithmus und hat eine heuristische Laufzeit von

$$O\left(2^{\frac{n_a}{3}}\right)$$

mit Bit-Länge n_a von a. Für die Rechenzeittabelle 11.4.1 für Algorithmus 11.2.7 wählen wir als Parameter $B_1 = 10^5$, $B_2 = 10^{11}$ und $j = 200$ (nach jeder 200-sten Potenzierung wird der größte gemeinsame Teiler berechnet). Die Größe von a selbst ist für die Rechenzeit der $(p-1)$-Methode kaum entscheidend, wie wir am Beispiel M_{1039} sehen können. Im Gegensatz zur ρ-Methode und zur Probedivision ist die Größe des kleinsten Primteilers nicht unmittelbar ausschlaggebend über Erfolg oder Misserfolg von Algorithmus 11.2.7. Wie erwähnt müssen wir besonders die Primfaktorzerlegung von $p-1$ des kleinsten Primfaktors p von a, noch spezieller den größten Primteiler von $p-1$ ins Auge fassen. So hat M_{137} als kleinsten Primteiler $p_{M_{137}} = 32032215596496435569$ mit 20 Stellen und wird in gut 9 Minuten faktorisiert, da der größte Primteiler von $p_{M_{137}} - 1$ „nur“ 10 Stellen

hat. Im Gegensatz dazu kann F_7 nicht faktorisiert werden, obwohl der kleinste Primteiler $p_{F_7} = 116503103764643$ nur 17 Stellen hat, weil der größte Primteiler von $p_{F_7} - 1$ 15-stellig ist.
Kleine 10-stellige Primteiler von $p - 1$ bilden die Grenze für den $(p-1)$-Algorithmus. Die Zahl $p_{F_8} - 1$ hat beispielsweise einen etwa 3-mal so großen Primteiler wie $p_{M_{137}} - 1$, F_8 kann aber nicht mehr im Zeitlimit von 10 Minuten (auch nicht nach 30 Minuten) faktorisiert werden. Problemlos sind dagegen Zahlen wie M_{67} und M_{101}, wo die Primteiler von $p_{M_{67}/M_{101}} - 1$ maximal 6 Stellen besitzen und eine Faktorisierung in unter einer Sekunde möglich ist. Ein interessantes Beispiel ist a_1, dessen 16-stelliger Primteiler $p_{a_1} = 200740742557443271$ eine so günstige Gestalt hat, dass wir in 32ms ein Ergebnis erhalten und der Faktorisierungsalgorithmus „factor(a_1)“ von PARI/GP über 2,5 Minuten benötigt, weil er nicht mit der $(p-1)$-Methode arbeitet.

Zahl	Berechnungszeit
M_{67}	31ms
M_{101}	3s947ms
M_{103}	0ms
M_{109}	125ms
M_{137}	9m15s395ms
M_{139}	0ms

Zahl	Berechnungszeit
M_{149}	×
M_{1039}	63ms
F_6	16ms
F_7	×
F_8	×
a_1	31ms

Tabelle 11.4.1: Zeitaufwand für eine Faktorberechnung mittels Algorithmus 11.2.7

Bei Pollards $(p-1)$-Algorithmus stellt sich zusammenfassend das Problem (bei manchen Zahlen auch den Segen), dass wir immer auf die Primfaktorzerlegung von $p-1$ angewiesen sind. Ist diese für alle Primteiler p von a unbrauchbar – sind also die Primteiler von $p-1$ so groß, dass sie nicht B-potenzglatt sind –, so finden wir mit dieser Methode keinen Teiler. Der Grund liegt darin, dass wir mit der Gruppenordnung von $(\mathbb{Z}/p\mathbb{Z})^*$ arbeiten müssen, die immer gleich $p-1$ ist. Um dem etwas auszuweichen kann man Williams $(p+1)$-Methode verwenden (siehe [27], Kapitel 18), in der eine Untergruppe U von $\mathbb{F}_{p^2}^*$ mit $|U| = p+1$ verwendet wird, wobei p ein Primteiler der zu faktorisierenden Zahl a ist. Analog zur $(p-1)$-Methode hofft man, dass $p+1$ eine Primfaktorzerlegung mit kleinen Primteilern besitzt. Für eine flexiblere Wahl der Gruppenordnung lernen wir im nächsten Kapitel die Elliptic Curve Method kennen.

12 Elliptic Curve Method

Wir kommen zu einem der wichtigsten und effizientesten Vertreter der Faktorisierungsalgorithmen, dem Faktorisierungsalgorithmus mit elliptischen Kurven. Im Jahre 1985 veröffentlichte Hendrik H. Lenstra Jr. in der Arbeit „Factoring integers with elliptic curves“ in den Annals of Mathematics (siehe [40]) die Elliptic Curve Method (ECM); oder auch Lenstras Algorithmus genannt.
Sei a eine zusammengesetzte natürliche Zahl mit $\gcd(a, 6) = 1$, die keine echte Potenz ist und p einer der Primteiler von a. Wie wir bereits in Kapitel 11 erwähnt haben, ist Pollards $(p-1)$-Algorithmus die Grundidee für die Faktorisierungsmethode mittels elliptischer Kurven. Bei der $(p-1)$-Methode stellt sich das Problem, dass wir auf die Primfaktorzerlegung von $p-1$, der Gruppenordnung von $(\mathbb{Z}/p\mathbb{Z})^*$ angewiesen sind. Ist diese ungünstig, so ist es mit dem Algorithmus schwierig einen Teiler zu berechnen. Bei den elliptischen Kurven genießen wir den Vorteil, dass wir für einen Teiler p von a viele verschiedene elliptische Kurven mit verschiedenen Gruppenordnungen aus dem Intervall $]p+1-2\sqrt{p}; p+1+2\sqrt{p}[$ erzeugen können und damit mehr Spielraum für eine „geeignete“ Primfaktorzerlegung einer Ordnung zu haben. Falls nämlich eine Gruppenordnung des Intervalls keine „geeignete“ Form hat, nehmen wir eine andere elliptische Kurve mit einer neuen, möglicherweise besseren Ordnung, bis wir Erfolg haben.

12.1 Der Algorithmus

Wir wählen eine elliptische Kurve $E(\mathbb{K}): \; y^2 = x^3 + a_4x + a_6$ über einem endlichen Körper $\mathbb{F}_q$ der Charakteristik $\neq 2, 3$. Zur Bestimmung eines geeigneten Grundkörpers $\mathbb{F}_q$ betrachten wir die folgenden beiden Punkte:

1. Für einen beliebigen Primteiler p von a können wir den – nicht bekannten – Körper $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ wählen. Dieser würde nach der Voraussetzung $\gcd(a, 6) = 1$ die Eigenschaft $\mathrm{char}(\mathbb{Z}/p\mathbb{Z}) \neq 2, 3$ erfüllen.

2. Die zweite Option ist der „Körper“ $\mathbb{Z}/a\mathbb{Z}$ selbst, der aber nach Bemerkung 2.0.5 kein Körper, sondern nur ein kommutativer Ring mit Einselement und damit $E(\mathbb{Z}/a\mathbb{Z})$ nach Definition 6.2.6 keine elliptische Kurve ist. Solche Kurven, also elliptische Kurven über einem Ring, heißen **pseudoelliptische Kurven**.

Aus der Not der obigen beiden Punkte machen wir eine Tugend und die Grundidee für den Algorithmus. Wir operieren auf der pseudoelliptischen Kurven $E(\mathbb{Z}/a\mathbb{Z})$ so, als ob sie eine elliptische Kurve wäre, und führen die Gruppenaddition auf ihr durch. Diese Additionen entsprechen genau den Additionen auf $E(\mathbb{Z}/p\mathbb{Z})$ – wenn wir sie durchführen können. Nach Definition 6.3.1 ist dies auf $E(\mathbb{Z}/a\mathbb{Z})$ solange möglich, solange Inverse zu $d := x_2 - x_1$ und $d := 2y_1$ existieren, die es in $\mathbb{Z}/a\mathbb{Z}$ nicht für alle Elemente gibt. Nicht mehr ausführbar ist die Addition, wenn

$$d \notin (\mathbb{Z}/a\mathbb{Z})^* \iff \gcd(d, a) > 1 \iff \exists \text{ Primzahl } p, \text{ sodass } p \mid a \text{ und } p \mid d$$

gilt, also wir in $\mathbb{Z}/p\mathbb{Z}$ ein Nullelement berechnet haben. Bestimmen wir dann $\gcd(d, a)$, so erhalten wir einen nichttrivialen Teiler von a.
Betrachten wir den Ablauf des Algorithmus' genauer. Wir wählen Elemente $x_0, y_0, a_4 \in \mathbb{Z}/a\mathbb{Z}$, setzen $a_6 \equiv y_0^2 - x_0^3 - a_4x_0 \bmod a$ und erhalten die pseudoelliptische Kurve

$$E(\mathbb{Z}/a\mathbb{Z}): \; y^2 = x^3 + a_4x + a_6$$

mit $Q = Q_0 = (x_0, y_0) \in E(\mathbb{Z}/a\mathbb{Z})$. Um $\mathrm{char}(\mathbb{Z}/a\mathbb{Z}) \neq 2, 3$ zu garantieren, setzen wir o.B.d.A. $\gcd(6, a) = 1$ voraus. Weiters muss die Kurve $E(\mathbb{Z}/a\mathbb{Z})$ glatt sein und nach Satz 6.2.8

$$\gcd(\Delta = 4a_4^3 + 27a_6^2, a) = 1$$

erfüllen; andernfalls erhalten wir einen Teiler von a. Gilt $\gcd(\Delta, a) = a$, so wählen wir neue Startwerte.
Sind die Startvoraussetzungen erfüllt, wählen wir, wie bei Pollards $(p-1)$-Methode, eine Schranke $B \in \mathbb{N}$ (vergleiche Kapitel 11.1) und bestimmen die Primzahlpotenzen $q_j^{f_j}$ (o.B.d.A. $q_j < q_{j+1}$) mit Primzahlen $q_j \leq B$ und maximalen f_j, sodass $q_j^{f_j} \leq B$ gilt ($\Rightarrow f_j = \log_{q_j}(B)$, vergleiche Kapitel 11.2). Weiters bilden wir iterativ die Vielfachen $Q = q_j^{f_j} \cdot Q = \underbrace{Q + \cdots + Q}_{q_j^{f_j}\text{–Mal}}$ für $j \geq 1$ auf $E(\mathbb{Z}/a\mathbb{Z})$.
Bevor wir das Vielfache mit der nächsten Primzahlpotenz $q_{j+1}^{f_{j+1}}$ bilden, überprüfen wir, ob wir die oben angesprochenen Inversen bilden können oder vielleicht bereits einen Teiler gefunden haben. Bei der Berechnung von $p = \gcd(d, a)$ treten folgende 3 Fälle ein:

1. Gilt $p \neq 1$ und $\neq a$, so haben wir einen echten Teiler p von a gefunden.
2. Ist $p = a$, so haben wir alle Primteiler auf einmal gefunden und müssen mit neuen Anfangswerten und einer neuen pseudoelliptischen Kurve starten.
3. Gilt $p = 1$, so können wir die Inversen berechnen und das Vielfache $Q = q_{j+1}^{f_{j+1}} \cdot Q$ bilden.

Punkt 1 tritt – analog zur $(p-1)$-Methode – nach [48], Satz 5.3 im Algorithmus genau dann auf, wenn die Ordnung o_{Q_0} von Q_0 in $E(\mathbb{Z}/p\mathbb{Z})$ B-potenzglatt ist und folglich $k \cdot Q_0 = \infty$ in $E(\mathbb{Z}/p\mathbb{Z})$ für B-potenzglatte Zahlen k mit $o_{Q_0} \mid k$ gilt. Andernfalls erhalten wir keinen Teiler und erzeugen mit neuen Startwerten $x_0, y_0, a_4 \in \mathbb{Z}/a\mathbb{Z}$ eine neue pseudoelliptische Kurve und ändern so die Ordnung von Q_0 oder vergrößern die Schranke B.

Beispiel 12.1.1. Sei $a = 10981$ ($\Rightarrow \gcd(a, 6) = 1$) und wähle $B = 25$. Als Startwerte der pseudoelliptischen Kurve wählen wir $Q = (x_0, y_0) = (2, 1)$ und $a_4 = 1$. Daraus folgt $a_6 = -9$, $\Delta = 4a_4^3 + 27a_6^2 = 2191$ und $\gcd(\Delta, a) = 1$, und die pseudoelliptische Kurve lautet $E(\mathbb{Z}/10981\mathbb{Z})$: $y^2 = x^3 + x - 9$ mit $Q \in E(\mathbb{Z}/10981\mathbb{Z})$. Für q_j und f_j ergeben sich folgende Werte:

$$q_1 = 2,\ f_1 = 4,\ q_2 = 3,\ f_2 = 2,\ q_3 = 5,\ f_3 = 2,\ q_4 = 7,\ f_4 = 1,\ q_5 = 11,\ f_5 = 1$$

$$q_6 = 13,\ f_6 = 1,\ q_7 = 17,\ f_7 = 1,\ q_8 = 19,\ f_8 = 1,\ q_9 = 23,\ f_9 = 1.$$

Mittels der PARI-Funktionen aus Anhang B.7 berechnen wir als erstes das Vielfache

$$Q = (x, y) = q_1^{f_1} \cdot Q = 2^4 \cdot (2, 1) = (8560, 6639).$$

Als nächstes berechnen wir $Q = q_2^{f_2} \cdot Q = 3^2 \cdot (8560, 6639)$ auf $E(\mathbb{Z}/10981\mathbb{Z})$ und zuvor kontrollieren wir, ob wir das Inverse von $d = x$ bilden können:[4]

$$\gcd(x, a) = \gcd(8560, 10981) = 1.$$

Weiters berechnen wir

$$Q = q_2^{f_2} \cdot Q = 3^2 \cdot (8560, 6639) = (935, 9929),$$

$$\gcd(935, 10981) = 1 \text{ und } Q = q_3^{f_3} \cdot Q = 5^2 \cdot (10161, 6040) = (79, 139)$$

und den Teiler $p = \gcd(79, 10981) = 79$. Das Ergebnis erhalten wir an dieser Stelle, weil Q_0 die Ordnung $o_{Q_0} = 25$ in $E(\mathbb{Z}/79\mathbb{Z})$ besitzt.

[4] Nach Konstruktion der PARI-Algorithmen aus Anhang B.7 genügt es für $d = x$ den größten gemeinsamen Teiler zu berechnen.

Algorithmus 12.1.2. (Elliptic Curve Algorithm (ECM))[5]

```
ecm(a,B)=
{local(p,Q,E,a4,a6,q);
 p=gcd(6,a);
 if(p!=1,
        print(p," ist ein Teiler von ",a)
        );
 l=1;
 while(p==1||p==a,
        Q=[random(a),random(a)];
        a4=random(a);
        a6=Q[2]^2-Q[1]^3-Q[1]*a4;
        p=gcd(4*a4^3+27*a6^2,a);
        if(p!=1&&p!=a,
                print(p," ist ein Teiler von ",a);
                break(5)
                );
        E=[0,0,0,a4,a6];
        q=2;
        while(q<=B,
                p=gcd(Q[1],a);
                if(p!=1&&p!=a,
                        print(p," ist ein Teiler von ",a);
                        break(5)
                        );
                if(p==a,
                        break(2)
                        );
                Q=ellpowmod(E,Q,q^(floor(logp(q,B))),a);
                q=nextprime(q++)
                )
        );
 if(p==1||p==a,
        print("Es wurde kein Teiler gefunden")
        )
}
```

Die Suchschranke B können wir je nach Größe des kleinsten Teilers p von a nach Tabelle 12.1.3 anpassen, wobei nach Erfahrungswerten für den „einfacheren“ Algorithmus 12.1.2 die Schranke B etwas größer gewählt werden sollte.

Dezimalstellen von p	< 15	< 20	< 25	< 30
Größe von B nach [1]	2000	11000	50000	250000

Tabelle 12.1.3: Größe der Suchschranke B in Abhängigkeit des kleinsten Teilers p von a

[5]Die PARI-Codes für die Funktionen „ellpowmod()“ und „elladdmod()“ befinden sich in Anhang B.7 und der für „logp()“ in Anhang B.4

12.2 Varianten der Elliptic Curve Method

12.2.1 Variante mit großer Primzahl

Diese Variante von ECM basiert auf der Idee der Variante des Pollard-$(p-1)$-Algorithmus (siehe Kapitel 11.2).
Wir definieren zu der Schranke $B_1 := B$ eine zweite größere Schranke $B_2 > B_1$. Hat die Suche nach einem Faktor mit B_1 in Algorithmus 12.1.2 keinen Teiler gebracht, so ist der Punkt

$$Q_0 := \prod_j q_j^{f_j} \cdot Q_0$$

mit $q_j \leq B_1$ und $f_j = \log_{q_j} B_1$ am Ende gegeben. Nun bilden wir Punkte der Form

$$Q = (x, y) = q_i \cdot Q_0$$

mit $B_1 < q_i \leq B_2$, indem wir den Punkt $Q = q_1 \cdot Q_0$ und die Differenzen $d_{i+1} := q_{i+1} - q_i$ der benachbarten Primzahlen zwischen B_1 und B_2 berechnen. Für $i \geq 1$ bestimmen wir mit relativ geringem Rechenaufwand rekursiv die Ausdrücke $q_{i+1} \cdot Q$ durch

$$(q_i + d_{i+1}) \cdot Q \ (= (q_1 + d_2 + \cdots + d_{i+1}) \cdot Q_0).$$

Ist für ein $Q = (x, y)$ an einer beliebigen Stelle $p = \gcd(d, a)$ nicht trivial, so haben wir einen Teiler gefunden. Andernfalls durchlaufen wir den Algorithmus mit einer neuen Kurve oder ändern die Suchschranken. Diese Variante von ECM befindet sich als PARI-Code in Anhang B.2.

12.3 Laufzeit und Eigenschaften

Die Elliptic Curve Method ist ein probabilistischer Algorithmus mit der heuristisch subexponentieller Laufzeit von

$$L_a[\frac{1}{2}; 1] = O\left(e^{(1+o(1))\sqrt{n_a \log(n_a)}}\right),$$

wenn die Primteiler von a eine annähernd gleiche Größenordnung haben. Damit hat ECM die (heuristisch) gleiche Laufzeit wie das quadratische Sieb (vergleiche Kapitel 16.3), ist aber aufgrund der komplexeren Rechenoperationen für große Zahlen in der Realität langsamer als das quadratische Sieb. Wollen wir die Laufzeit lediglich in Abhängigkeit des kleinsten Primteilers p von a, so beträgt diese

$$L_p[\frac{1}{2}; \sqrt{2}] = O\left(e^{(\sqrt{2}+o(1))\sqrt{n_p \log(n_p)}}\right).$$

Ist ein Primteiler p wesentlich kleiner als $\sqrt{a}$, so lässt sich mit elliptischen Kurven relativ effizient ein Faktor finden, wie in Tabelle 12.3.2 zu sehen. Eine genauere Ausführung der Laufzeitanalyse ist in [23], Kapitel 7.4.1 zu finden.
Wir betrachten die Gruppenordnung der elliptischen Kurven genauer, die wir im Gegensatz zu Pollards-$(p-1)$-Algorithmus wesentlich flexibler wählen können. Die Ordnung einer elliptische Kurve $E(\mathbb{Z}/p\mathbb{Z})$ für einen Teiler p von a liegt nach Theorem 6.4.2 im Intervall

$$]p + 1 - 2\sqrt{p}; p + 1 + 2\sqrt{p}[.$$

Doch haben wir wirklich alle Gruppenordnungen zur Auswahl? Oder anders gefragt: Werden für eine beliebige Primzahl p tatsächlich alle Ordnungen im Intervall von zumindest einer Kurve $E(\mathbb{Z}/p\mathbb{Z})$ angenommen? Diese Frage beantwortet folgendes Resultat aus [26].

Zahl	Berechnungszeit
M_{67}	845ms
M_{101}	12s330ms
M_{103}	7s971ms
M_{109}	2s190ms
M_{137}	6m0s565ms
M_{139}	25s569ms

Zahl	Berechnungszeit
M_{149}	×
M_{1039}	93ms
F_6	624ms
F_7	1m32s071ms
F_8	1m36s807ms
a_1	2m58s543

Tabelle 12.3.2: Zeitaufwand für die Faktorberechnung mittels Algorithmus 12.1.2

Satz 12.3.1. *Sei p eine Primzahl und $N \in]p+1-2\sqrt{p}; p+1+2\sqrt{p}[$ beliebig. Dann gibt es ein Paar $(a_4, a_6) \in (\mathbb{Z}/p\mathbb{Z}) \times (\mathbb{Z}/p\mathbb{Z})$, sodass die elliptische Kurve $E(\mathbb{Z}/p\mathbb{Z}) : \ y^2 = x^3 + ax + a_6$ genau N Elemente besitzt.*

In Tabelle 12.3.2 lesen wir ab, dass der kleinste Primteiler p einer zu faktorisierenden Zahl a für die Berechnungszeit eines Faktors ausschlaggebend ist, wie beim Beispiel M_{1039} dargestellt. Im Gegensatz zur $(p-1)$-Methode von Pollard, wo die Primfaktorzerlegung von $p-1$ entscheidend ist, spielt die Struktur des Teilers kaum eine Rolle; hauptsächlich die Größe entscheidet über die Rechenzeit. Der Grund dafür liegt an der großen Auswahl an Ordnungen aus dem Intervall $]p+1-2\sqrt{p}; p+1+2\sqrt{p}[$ für die elliptische Kurve $E(\mathbb{Z}/p\mathbb{Z})$. Passt eine Ordnung nicht, so wählen wir eine andere Kurve mit einer eventuell besseren Ordnung.
Für Zahlen mit etwa 10-stelligen Teilern, wie etwa bei M_{103} mit ca. 8 Sekunden Rechenzeit, ist die Faktorberechnung von ECM noch vergleichsweise langsam – Pollard-ρ benötigt dafür unter 0,5 Sekunden. Für M_{101} mit dem 13-stelligen Teiler $p = 7432339208719$ benötigt ECM 25 Sekunden für ein Ergebnis, während Pollard-ρ bereits über 25 Sekunden braucht. Das heißt, mit wachsender Stellenzahl der Teiler steigt die Rechenzeit bei ECM nicht so rasant an wie bei anderen Algorithmen. Die Grenze für Algorithmus 12.1.2 liegt bei Zahlen mit 20-stelligen Teilern, die eine Berechnungszeit von knapp 10 Minuten haben, wie bei M_{137} der Fall. Für M_{149}, mit 21-stelligem Teiler, steigt die Rechenzeit zu stark an und benötigt weit über 10 Minuten.

13 Faktorisieren nach Fermat

Wir haben die Methoden nach Pollard und deren Weiterentwicklungen abgeschlossen und kommen zu den Algorithmen nach Fermat und Legendre und deren Weiterentwicklungen. Die Grundidee für die Methode nach Pierre de Fermat aus dem 17. Jahrhundert liefert folgende Aussage.

Satz 13.0.1. *Sei a eine ungerade natürliche Zahl, dann gilt*

$$a \text{ ist zusammengesetzt} \iff \text{es gibt Zahlen } x, y \in \mathbb{N} \text{ mit } |x-y|>1, \text{ sodass } a = x^2 - y^2 \text{ gilt.}$$

Beweis. ($\Leftarrow$) Seien $x, y \in \mathbb{N}$ so, dass $a = x^2 - y^2$ gilt. Dann folgt $a = x^2 - y^2 = (x-y)(x+y)$, also ist a zusammengesetzt, da $|x-y| > 1$ gilt.
($\Rightarrow$) Sei $a = p \cdot q$ zusammengesetzt ($\Rightarrow p$ und q sind ungerade, da a ungerade ist). Für $x := \frac{p+q}{2} \in \mathbb{N}$ und $y := \frac{p-q}{2} \in \mathbb{N}$ gilt $x^2 - y^2 = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2 = \frac{p^2+2pq+q^2-p^2+2pq-q^2}{4} = p \cdot q = a$, also lässt sich a als Differenz zweier Quadrate x^2, y^2 mit $|x-y| > 1$ schreiben. □

13.1 Der Algorithmus

Sei a eine ungerade zusammengesetzte natürliche Zahl. Wir können eine Faktorisierung von a angeben, wenn wir nach Satz 13.0.1 zwei Quadrate x^2, y^2 finden, sodass $a = x^2 - y^2$ gilt – so der theoretische Ansatz. Dafür setzen wir $x := \lceil \sqrt{a} \rceil$ und $y^2 := x^2 - a$ und hoffen, dass y eine natürliche Zahl ist. Ist das nicht der Fall, so erhöhen wir x um 1 und berechnen den neuen y^2-Wert $y^2 := x^2 - a$, um wieder festzustellen, ob $y \in \mathbb{N}$ gilt, solange, bis y eine natürliche Zahl ist.
In Algorithmus 13.1.1 berechnen wir $(x+1)^2$ durch Hinzuaddieren von $u := 2x+1$ zu x.

Algorithmus 13.1.1. (Faktorisieren nach Fermat)

```
fermat(a)=
{local(x,u,z);
 x=ceil(sqrt(a));
 u=2*x+1;
 z=sqr(x)-a;
 while(z!=sqr(floor(sqrt(z))),
        z=z+u;
        u=u+2;
        x++
        );
        print(floor(x+sqrt(z))," ist ein Teiler von ",a)
}
```

Beispiel 13.1.2. Sei $a = 19279$. Daraus folgt

$$x = 141,\ u = 283,\ y^2 = 152 \text{ und } y \approx 12,3288.$$

Wir berechnen

$$\begin{aligned}
y^2 &= y^2 + u = 435,\ u = u+2 = 285,\ x = x+1 = 142 \text{ und } y \approx 20,8567,\\
y^2 &= y^2 + u = 720,\ u = u+2 = 287,\ x = x+1 = 143 \text{ und } y \approx 26,8328,\\
y^2 &= y^2 + u = 1007,\ u = u+2 = 289,\ x = x+1 = 144 \text{ und } y \approx 31,7333,\\
y^2 &= y^2 + u = 1296,\ u = u+2 = 291,\ x = x+1 = 145 \text{ und } y = 36.
\end{aligned}$$

Somit ist $p = x - y = 145 - 36 = 109$ (bzw. $p = x + y = 145 + 36 = 181$) ein Teiler von a.

13.2 Varianten von Fermat

Liegen die Teiler von a zu weit von $\lceil\sqrt{a}\rceil$ entfernt, so benötigt die *while*-Schleife in Algorithmus 13.1.1 zu viele Iterationen. Tritt das angesprochene Problem ein, so können wir alternativ den Algorithmus auf ein Vielfaches $k \cdot a$ von a ($k \in \mathbb{N}$) anwenden und hoffen, dass die Teiler hier günstiger liegen und wir mit weniger Iterationen einen Teiler p' von $k \cdot a$ finden und so $p = \gcd(p', a)$ ein echter Teiler von a ist. A priori haben wir aber keine Möglichkeit herauszufinden, welches Vielfache von a am besten dafür geeignet ist.

Beispiel 13.2.1. Sei, wie in Beispiel 13.1.2 $a = 19279$ und wähle $k = 8$, also gilt $8 \cdot 19279 = 157832$. Daraus folgt

$$x = 398,\ u = 797,\ y^2 = 572 \text{ und } y \approx 23,9165 \text{ und}$$

$$y^2 = y^2 + u = 1369,\ u = u + 2 = 799,\ x = x + 1 = 399 \text{ und } y = 37,$$

also haben wir bereits im zweiten Schritt ein Quadrat gefunden. Wir berechnen $p' = x + y = 436$ als Teiler von $k \cdot a$ und $p = \gcd(p', a) = 109$ als Teiler von a.

Mit der sogenannten Faktorisierungsmethode von Lehman mit Laufzeit

$$O(2^{\frac{n_a}{3}})$$

ist es möglich ein geeignetes Vielfaches von a methodisch aufzuspüren. Für ausführlichere Informationen dieser Methode siehe [23], Seite 227 und für den PARI-Code der Lehman-Methode siehe Anhang B.3.

13.3 Laufzeit und Eigenschaften

Haben wir genügend Zeit zur Verfügung, so würde uns Algorithmus 13.1.1 jedes Mal einen Faktor von a berechnen – theoretisch. Der deterministische Faktorisierungsalgorithmus nach Fermat hat die exponentielle Laufzeit

$$O(2^{n_a})$$

und ist damit asymptotisch schlechter als die Probedivision. Am ehesten eignet sich Fermats Methode für Zahlen mit zwei Teilern, die nahe beieinander liegen, da wir mit x-Werten bei $\sqrt{a}$ und y-Werten nahe bei 0 starten. Damit liegen die Teiler $(x - y)$ und $(x + y)$ bei kurzer Rechenzeit eng zusammen, doch je weiter die Teiler auseinander liegen, desto länger braucht der Algorithmus. Algorithmus 13.1.1 konnte keine einzige der Zahlen aus Tabelle 9.0.0 faktorisieren, da deren Primteiler durchgehend zu weit auseinander liegen, weswegen wir auf eine entsprechende Tabelle für die Berechnungszeiten verzichten.
Doch wozu haben wir die Methode von Fermat kennen gelernt? Die Idee die Zahl a als Differenz von 2 Quadraten darzustellen liefert die Grundidee für den Kettenbruchalgorithmus (Kapitel 15), das quadratische Sieb (Kapitel 16) und das Zahlkörpersieb (Kapitel 17), die aktuell zu den schnellsten Faktorisierungsalgorithmen gehören.

14 Legendres Faktorisierung mit Faktorbasen

Der um 1900 entwickelte Algorithmus von Adrien-Marie Legendre basiert auf Fermats Idee (ausgearbeitet und publiziert wurde die Methode von Maurice Kraitchik [35] im Jahr 1926) die zu faktorisierende Zahl mittels der Differenz zweier Quadrate darzustellen. Für eine natürliche, zusammengesetzte Zahl a, die sich nicht als Potenz schreiben lässt, betrachten wir die quadratische Kongruenz

$$x^2 \equiv y^2 \bmod a, \tag{14.1}$$

die $a \mid x^2 - y^2 = (x-y)(x+y)$ erfüllt. Gilt zusätzlich

$$x \not\equiv \pm y \bmod a, \tag{14.2}$$

so sind $(x-y)$ und $(x+y)$ echte Teiler von a. Andernfalls würde $(x \pm y) \mid a$ gelten und damit $a \mid (x-y)$ bzw. $a \mid (x+y)$.
Die erste für uns interessante Frage ist: Wie viele Lösungen x besitzt Gleichung (14.1), wenn wir uns ein Quadrat y^2 vorgeben? Oder nach Kapitel 4 formuliert: Wie viele Wurzeln hat ein quadratischer Rest b^2 modulo a in $\mathbb{Z}/a\mathbb{Z}$? Sei also $b^2 \in \mathbb{Z}/a\mathbb{Z}$ ein quadratischer Rest modulo a (siehe Definition 4.0.1).
Ist $a \neq 2$ eine Primzahl, so gibt es genau die 2 Wurzeln $\pm b$ für $b^2 \bmod a$, weil das Polynom $f(z) = z^2 - b^2$ in $\mathbb{Z}/a\mathbb{Z}$ genau 2 Nullstellen hat ($\mathbb{Z}/a\mathbb{Z}$ ist ein Körper, siehe Bemerkung 2.0.5). Diese Lösungen erfüllen Gleichung (14.2), sind also für uns unerwünscht. Da wir a als zusammengesetzt voraussetzen, tritt dieser Fall aber nicht ein. Sei also a zusammengesetzt und b^2 ein quadratischer Rest modulo a. Wie viele Wurzeln besitzt $b^2 \in \mathbb{Z}/a\mathbb{Z}$ dann? Hat a genau k verschiedene, ausschließlich ungerade Primteiler, so besitzt b^2 genau 2^k verschiedene Wurzeln. Denn sei $a = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$, $p_i \neq 2$, $p_i \neq p_j \ \forall i \neq j$, dann gilt nach dem chinesischen Restsatz (Satz 2.0.8)

$$\mathbb{Z}/a\mathbb{Z} \cong \mathbb{Z}/p_1^{e_1}\mathbb{Z} \times \mathbb{Z}/p_2^{e_2}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_k^{e_k}\mathbb{Z},$$

das heißt, wir können das Element $b^2 \in \mathbb{Z}/p_1^{e_1}\mathbb{Z} \times \mathbb{Z}/p_2^{e_2}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_k^{e_k}\mathbb{Z}$ als k-Tupel $(b_1^2, b_2^2, \ldots, b_k^2)$ schreiben. In $\mathbb{Z}/p_i^{e_i}\mathbb{Z}$ besitzt jedes der b_i^2 genau 2 Wurzeln, also können wir $b = (\pm b_1, \pm b_2, \ldots, \pm b_k) \in \mathbb{Z}/p_1^{e_1}\mathbb{Z} \times \mathbb{Z}/p_2^{e_2}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_k^{e_k}\mathbb{Z}$ schreiben. Damit hat b^2 genau 2^k Wurzeln, wovon genau 2 davon $\pm b$ darstellen.
Gilt o.B.d.A. $p_1^{e_1} = 2$, so hat b^2 genau 2^{k-1} Wurzeln $(b_1, \pm b_2, \ldots, \pm b_k)$, da nach Satz 4.0.10 $b_1^2 \equiv 1 \bmod 2$ gelten muss und b_1^2 in $\mathbb{Z}/2\mathbb{Z}$ genau 1 Lösung hat. Gilt $p_1^{e_1} = 2^2$, so hat b^2 wieder genau 2^k Wurzeln $(\pm b_1, \pm b_2, \ldots, \pm b_k)$, da $b_1^2 \equiv 1 \bmod 4$ gelten muss und b_1^2 in $\mathbb{Z}/4\mathbb{Z}$ genau die 2 Lösungen $\overline{1}$ und $\overline{3} = \overline{-1}$ besitzt. Gilt $p_1^{e_1} = 2^{e_1}$ mit $e_1 \geq 3$, so hat b^2 genau 2^{k+1} Wurzeln, da nach Lemma 4.0.9 $b_1^2 \equiv 1 \bmod 8$ gelten muss und b_1^2 in $\mathbb{Z}/2^{e_1}\mathbb{Z}$ genau 4 Lösungen hat, da b_1^2 in $\mathbb{Z}/8\mathbb{Z}$ genau 4 Lösungen hat. Wir formulieren unsere Beobachtungen in folgendem Satz.

Satz 14.0.1. *Sei a eine ungerade natürliche Zahl, $a = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$ dessen Primfaktorzerlegung und b^2 ein quadratischer Rest modulo a, dann hat b^2 genau 2^k verschiedene Wurzeln in $\mathbb{Z}/a\mathbb{Z}$. Ist a gerade, so gilt*
(a) b^2 hat genau 2^{k-1} verschiedene Wurzeln, falls $2 \mid a$ und $4 \nmid a$ gilt.
(b) b^2 hat genau 2^k verschiedene Wurzeln, falls $4 \mid a$ und $8 \nmid a$ gilt.
(c) b^2 hat genau 2^{k+1} verschiedene Wurzeln, falls $8 \mid a$ gilt.

Ist a eine ungerade zusammengesetzte Zahl, gilt also $k \geq 2$, so besitzt ein quadratischer Rest modulo a nach Satz 14.0.1 zumindest 4 Wurzeln, von denen nur 2 Eigenschaft (14.2) nicht erfüllen. Dadurch ist die Wahrscheinlichkeit für eine beliebige Lösung von (14.1), dass wir auf die beiden unerwünschten Wurzeln treffen, gleich $\frac{2}{2^k} = \frac{1}{2^{k-1}} \leq 50\%$. Erfüllen 2 Quadrate x^2 und y^2 Gleichung (14.1), so ist folglich die Wahrscheinlichkeit hoch ($\geq 50\%$), dass sie auch Gleichung (14.2) erfüllen. Ist das nicht der Fall, so suchen wir nach einem neuen Paar (x, y).

Um ein Paar (x, y), das Kongruenz (14.1) erfüllt, zu finden, brauchen wir den Begriff der Faktorbasis, die einen sehr ähnlichen Zweck wie die Definition von B-glatt erfüllt (siehe Definition 11.0.1). Zur Erinnerung: Für $B \in \mathbb{N}$ ist eine Zahl $b \in \mathbb{N}$ genau dann B-glatt, wenn ihre Primfaktorzerlegung nur aus Primzahlen $\leq B$ besteht.

Definition 14.0.2. Seien $p_1, p_2, \ldots, p_r$ die ersten r Primzahlen, dann nennen wir die Menge

$$F_{p_r} := \{-1, p_1, p_2, \ldots, p_r\}$$

eine **Faktorbasis**.

Wir sagen, dass eine ganze Zahl b F_{p_r}-glatt ist, falls

$$b = (-1)^{e_0} \prod_{i=1}^{r} p_i^{e_i}$$

mit $e_i \in \mathbb{N}_0$ gilt. Wir konstruieren nun zwei Quadrate x^2 und y^2, die Gleichung (14.1) erfüllen. Dazu wählen wir eine „geeignete“ (in Abhängig der Größe von a) Faktorbasis F_{p_r} und k beliebige Elemente $b_j^2 \in \mathbb{Z}/a\mathbb{Z}$ aus. All diejenigen b_j^2, die nicht F_{p_r}-glatt sind, verwerfen wir sofort wieder, also sind alle b_j^2 $(1 \leq j \leq k)$ F_{p_r}-glatt und lassen sich als

$$b_j^2 \equiv (-1)^{e_{j_0}} \prod_{i=1}^{r} p_i^{e_{j_i}} \bmod a \tag{14.3}$$

mit $e_{j_i} \in \mathbb{N}_0$ $(1 \leq j \leq k$ und $0 \leq i \leq r)$ schreiben. Wir betrachten die Exponenten e_{j_i} im Körper $\mathbb{Z}/2\mathbb{Z}$: Für gerade Exponenten e_{j_i} gilt $e_{j_i} \equiv 0 \bmod 2$, für ungerade $e_{j_i} \equiv 1 \bmod 2$. Als nächstes schreiben wir alle Exponenten e_{j_i} in eine Matrix

$$A = \begin{array}{c} \\ b_1^2 \\ b_2^2 \\ b_3^2 \\ \vdots \\ b_k^2 \end{array} \begin{array}{c} \begin{array}{cccccc} -1 & p_1 & p_2 & \ldots & p_r \end{array} \\ \begin{pmatrix} e_{1_0} & e_{1_1} & e_{1_2} & \ldots & e_{1_r} \\ e_{2_0} & e_{2_1} & e_{2_2} & \ldots & e_{2_r} \\ e_{3_0} & e_{3_1} & e_{3_2} & \ldots & e_{3_r} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e_{k_0} & e_{k_1} & e_{k_2} & \ldots & e_{k_r} \end{pmatrix} \end{array} \equiv \begin{array}{c} \\ b_1^2 \\ b_2^2 \\ b_3^2 \\ \vdots \\ b_k^2 \end{array} \begin{array}{c} \begin{array}{cccccc} -1 & p_1 & p_2 & \ldots & p_r \end{array} \\ \begin{pmatrix} 0/1 & 0/1 & 0/1 & \ldots & 0/1 \\ 0/1 & 0/1 & 0/1 & \ldots & 0/1 \\ 0/1 & 0/1 & 0/1 & \ldots & 0/1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0/1 & 0/1 & 0/1 & \ldots & 0/1 \end{pmatrix} \end{array} \bmod 2, \tag{14.4}$$

wobei die j-te Zeile aus den Exponenten e_{j_i} der Primfaktorzerlegung des Elements b_j^2 besteht. Nun wählen wir beliebige Zeilen der Matrix A so, dass diese komponentenweise aufaddiert modulo 2 den Nullvektor ergeben, also besitzt diese Vektorsumme nur noch gerade Exponenten. Anders ausgedrückt wählen wir eine Teilmenge $A_0 \subseteq \{1, 2, \ldots, k\}$ so, dass

$$\sum_{j \in A_0} \begin{pmatrix} e_{j_0} \\ e_{j_1} \\ e_{j_2} \\ \vdots \\ e_{j_r} \end{pmatrix} = \begin{pmatrix} \sum_{j \in A_0} e_{j_0} \\ \sum_{j \in A_0} e_{j_1} \\ \sum_{j \in A_0} e_{j_2} \\ \vdots \\ \sum_{j \in A_0} e_{j_r} \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \bmod 2 \tag{14.5}$$

gilt.
Zusammenfassend ist eine Menge von Zahlen $\{b_j \mid j \in A_0\}$ gegeben, deren Quadrate sich nach Gleichung (14.3) als

$$b_j^2 \equiv (-1)^{e_{j_0}} \prod_{i=1}^{r} p_i^{e_{j_i}} \bmod a \tag{14.6}$$

schreiben lassen. Multiplizieren wir diese Quadrate miteinander, erhalten wir

$$\prod_{j\in A_0} b_j^2 \equiv \prod_{j\in A_0}\left((-1)^{e_{j_0}}\prod_{i=1}^{r} p_i^{e_{j_i}}\right) \equiv (-1)^{\sum_{j\in A_0} e_{j_0}} \prod_{i=1}^{r} p_i^{\sum_{j\in A_0} e_{j_i}} \bmod a. \tag{14.7}$$

Nach Gleichung (14.5) sind alle Summen $\sum_{j\in A_0} e_{j_i}$ $(0 \le i \le r)$ in (14.7) gerade, also alle Exponenten gerade, und es ist auf der linken wie auf der rechten Seite in (14.7) ein Quadrat gegeben. Daher folgt

$$\left(\prod_{j\in A_0} b_j\right)^2 \equiv (-1)^{2\cdot\frac{\sum_{j\in A_0} e_{j_0}}{2}} \prod_{i=1}^{r} p_i^{2\cdot\frac{\sum_{j\in A_0} e_{j_i}}{2}} \equiv \left((-1)^{\frac{\sum_{j\in A_0} e_{j_0}}{2}} \prod_{i=1}^{r} p_i^{\frac{\sum_{j\in A_0} e_{j_i}}{2}}\right)^2 \bmod a. \tag{14.8}$$

Definieren wir

$$x :\equiv \prod_{j\in A_0} b_j \bmod a \quad \text{und} \quad y :\equiv (-1)^{\frac{\sum_{j\in A_0} e_{j_0}}{2}} \prod_{i=1}^{r} p_i^{\frac{\sum_{j\in A_0} e_{j_i}}{2}} \bmod a, \tag{14.9}$$

so erfüllt das Paar (x, y) Gleichung (14.1). Erfüllt (x, y) auch Gleichung (14.2), was durchschnittlich zumindest jedes zweite Mal der Fall ist, so erhalten wir die echten Teiler $p = \gcd(x \pm y, a)$. Wird Gleichung (14.2) nicht erfüllt, so wählen wir eine neue Teilmenge A_0 aus und fügen gegebenenfalls neue Zeilen zu A hinzu.

Beispiel 14.0.3. Sei $a = 400003$, $F_{41} = \{-1, 2, 3, \ldots, 37, 41\}$, $b_1 = 1095$, $b_2 = 1674$, $b_3 = 2000$, $b_4 = 2001$ und $b_5 = 4000$. Daraus folgt

$$\begin{aligned} b_1^2 &\equiv -984 \equiv (-1)\cdot 2^3 \cdot 3 \cdot 41 \bmod 400003 \\ b_2^2 &\equiv 2255 \equiv 5 \cdot 11 \cdot 41 \bmod 400003 \\ b_3^2 &\equiv -30 \equiv (-1)\cdot 2 \cdot 3 \cdot 5 \bmod 400003 \\ b_4^2 &\equiv 3971 \equiv 11 \cdot 19^2 \bmod 400003 \\ b_5^2 &\equiv -120 \equiv (-1)\cdot 2^3 \cdot 3 \cdot 5 \bmod 400003 \end{aligned}$$

und wir stellen die Matrix

$$A = \begin{array}{c} \\ -984 \\ 2255 \\ -30 \\ 3971 \\ -120 \end{array} \begin{array}{c} \begin{array}{ccccccc} -1 & 2 & 3 & 5 & 11 & 19 & 41 \end{array} \\ \begin{pmatrix} 1 & 3 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 \\ 1 & 3 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \equiv \begin{array}{c} \begin{array}{ccccccc} -1 & 2 & 3 & 5 & 11 & 19 & 41 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \bmod 2 \tag{14.10}$$

auf, wobei wir Spalten mit lauter Nullen weggelassen haben. Setzen wir $A_0 = \{1, 2, 3, 4\}$, so gilt

$$\sum_{j=1}^{4} \begin{pmatrix} e_{j_0} \\ e_{j_1} \\ e_{j_2} \\ e_{j_3} \\ e_{j_4} \\ e_{j_5} \\ e_{j_6} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 0 \\ 2 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \bmod 2. \tag{14.11}$$

Damit erfüllen

$$x := \prod_{j=1}^{4} b_j = 1095 \cdot 1674 \cdot 2000 \cdot 2001 \equiv 242019 \bmod 400003$$

und

$$y := (-1)^{\frac{\sum_{j=1}^{4} e_{j_0}}{2}} \prod_{i=1}^{6} p_i^{\frac{\sum_{j=1}^{4} e_{j_i}}{2}} = (-1)^1 \cdot 2^2 \cdot 3^1 \cdot 5^1 \cdot 11^1 \cdot 19^1 \cdot 41^1 \equiv 114137 \bmod 400003.$$

Gleichung (14.1), und da auch $242019 \not\equiv \pm 114137 \bmod 400003$ gilt, ist

$$p = \gcd(x - y, a) = \gcd(127882, 400003) = 1487$$

ein echter Teiler von 400003.

In Beispiel 14.0.3 benötigen wir für 13 Primzahlen aus F_{41} lediglich 5 Zeilen in der Matrix A. Wie viele Gleichungen $b_j^2 \equiv (-1)^{e_{j_0}} \prod_{i=1}^{r} p_i^{e_{j_i}} \bmod a$ müssen wir im Allgemeinen für die Matrix A aufstellen, sodass wir modulo 2 linear abhängige Zeilen erhalten?
Die Matrix $A \in M_{k \times r+1}(\mathbb{Z}/2\mathbb{Z})$ besteht aus k Zeilen und $r+1$ Spalten. Eine lineare Abhängigkeit erreichen wir, wenn A mehr Zeilen als Spalten besitzt, also $k > r+1$ gilt. Daher müssen wir maximal $r+2$ Elemente $b_j^2 \in \mathbb{Z}/a\mathbb{Z}$ finden, die F_{p_r}-glatt sind, um Gleichung (14.5) zu erzwingen. Wenn wir Pech haben, erfüllen die daraus resultierenden x- und y-Werte Eigenschaft (14.2) nicht und wir müssen weitere Zeilen zu A hinzufügen. In der Regel wird das wie oben erwähnt nicht oft der Fall sein.
Das Problem an der Idee von Legendre ist, dass wir keine Möglichkeit haben, F_{p_r}-glatt Elemente b_j^2 zu konstruieren. Hat r eine Größe von etwa 10^4, so müssen wir eventuell über 10000 Gleichungen für A und damit über 10000 entsprechende b_j^2 finden. Genau hier kommen wir zu dem Punkt, wo Legendres Idee aufhört und wo die folgenden Algorithmen (Kettenbruchalgorithmus, quadratisches Sieb und Zahlkörpersieb) mit ihren verschiedenen Ideen ansetzen. Die Grundlage liefert immer Legendre, aber für die Konstruktion von geeigneten b_j^2 hat jeder der Algorithmen einen eigenen Ansatz.

15 Der Kettenbruchalgorithmus

Der Kettenbruchalgorithmus (englisch: CFRAC=Continued Fraction Factorization) wurde im Jahr 1931 von Derrick Henry Lehmer und Ralph Ernest Powers veröffentlicht und Anfang der 70er-Jahre von Michael Morrison und John Brillhart computergerecht aufbereitet. Wie am Ende des letzten Kapitels erwähnt, ist die Basis des Algorithmus' die Idee von Legendre, nach der wir ausreichend viele F_{p_r}-glatte Werte b_j^2 für eine geeignete Faktorbasis F_{p_r} benötigen, sodass wir die quadratische Kongruenz $x^2 \equiv y^2 \bmod a$ bilden und so eine zusammengesetzte Zahl a faktorisieren können. Die Grundlagen der Kettenbrüche haben wir in Kapitel 5 vorbereitet.

15.1 Der Algorithmus

Wir wollen eine zusammengesetzte ungerade natürliche Zahl a, keine echte Potenz, mittels der quadratischen Kongruenz $x^2 \equiv y^2 \bmod a$ faktorisieren. Dazu wählen wir eine Faktorbasis F_{p_r} und entwickeln $\sqrt{a}$ in einen regelmäßigen Kettenbruch. Dieser hat nach Satz 5.0.8 die Form

$$\sqrt{a} = [a_0; \overline{a_1, \dots a_s, 2a_0}]$$

mit Vorperiode 1 und Periode $s+1$. Nun betrachten wir mit wachsendem Index j die j-ten Näherungsbrüche $\frac{b_j}{c_j}$, konkreter deren Zähler $b_j \bmod a$. Diese berechnen wir nach Satz 5.0.4 durch $b_0 := a_0$ und $b_1 := a_0a_1+1$ und die Rekursion $b_j = a_jb_{j-1}+b_{j-2}$ für $j \geq 2$. Mittels der Probedivision überprüfen wir, ob für einen Index $b_j^2 \bmod a$ F_{p_r}-glatt ist. Ist das der Fall, so speichern wir dessen Exponentenvektor $(e_{j_i})_{1\leq i\leq r}$ modulo 2, andernfalls wird die Zahl verworfen. Die Wahrscheinlichkeit für $b_j^2 \bmod a$ F_{p_r}-glatt zu sein ist nach Satz 5.0.6 wesentlich besser, als für beliebige Quadrate, da $|b_j^2 \bmod a| < 2\sqrt{a}\ \ \forall j$ gilt. Haben wir genügend – also mindestens $r+2$ – F_{p_r}-glatte Zähler berechnet, so erhalten wir die Matrix (vergleiche Gleichung (14.4))

$$A = \begin{array}{c} \\ b_1^2 \\ b_2^2 \\ b_3^2 \\ \vdots \\ b_k^2 \end{array} \begin{array}{c} \begin{array}{ccccc} -1 & p_1 & p_2 & \dots & p_r \end{array} \\ \begin{pmatrix} e_{1_0} & e_{1_1} & e_{1_2} & \dots & e_{1_r} \\ e_{2_0} & e_{2_1} & e_{2_2} & \dots & e_{2_r} \\ e_{3_0} & e_{3_1} & e_{3_2} & \dots & e_{3_r} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e_{k_0} & e_{k_1} & e_{k_2} & \dots & e_{k_r} \end{pmatrix} \end{array} \equiv \begin{array}{c} \\ b_1^2 \\ b_2^2 \\ b_3^2 \\ \vdots \\ b_k^2 \end{array} \begin{array}{c} \begin{array}{ccccc} -1 & p_1 & p_2 & \dots & p_r \end{array} \\ \begin{pmatrix} 0/1 & 0/1 & 0/1 & \dots & 0/1 \\ 0/1 & 0/1 & 0/1 & \dots & 0/1 \\ 0/1 & 0/1 & 0/1 & \dots & 0/1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0/1 & 0/1 & 0/1 & \dots & 0/1 \end{pmatrix} \end{array} \bmod 2.$$

Analog zu Kapitel 14, Gleichung (14.5) wählen wir eine Teilmenge A_0 an Zeilen aus, sodass

$$\sum_{j\in A_0} \begin{pmatrix} e_{j_0} \\ e_{j_1} \\ e_{j_2} \\ \vdots \\ e_{j_r} \end{pmatrix} = \begin{pmatrix} \sum_{j\in A_0} e_{j_0} \\ \sum_{j\in A_0} e_{j_1} \\ \sum_{j\in A_0} e_{j_2} \\ \vdots \\ \sum_{j\in A_0} e_{j_r} \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \bmod 2$$

gilt, und definieren

$$x :\equiv \prod_{j\in A_0} b_j \bmod a \quad \text{und} \quad y :\equiv (-1)^{\frac{\sum_{j\in A_0} e_{j_0}}{2}} \prod_{i=1}^{r} p_i^{\frac{\sum_{j\in A_0} e_{j_i}}{2}} \bmod a,$$

die die Kongruenz $x^2 \equiv y^2 \bmod a$ erfüllen. Gilt zusätzlich $x \not\equiv \pm y \bmod a$, so haben wir die echten Teiler $p = \gcd(x \pm y, a)$ gefunden. Andernfalls greifen wir auf eine andere Teilmenge A_0 zurück und fügen gegebenenfalls neue Zeilen zur Matrix A hinzu.

Beispiel 15.1.1. Sei $a = 37949$, und wir wählen $F_{p_r} = F_{20} = \{-1, 2, 3, \ldots, 19\}$. Die ersten 9 Glieder der Kettenbruchentwicklung von $\sqrt{9073}$ sind

$$\sqrt{9073} = [a_0; a_1, a_2, \ldots, a_8] = [194; 1, 4, 7, 1, 3, 55, 2, 2].$$

Daraus berechnen wir

$$b_0 := a_0 = 194 \text{ und } b_1 := a_0a_1 + 1 = 195$$

und

$$b_2 = 974, \quad b_3 = 7013, \quad b_4 = 7987, \quad b_5 = 7760, \quad b_6 = 30974, \quad b_7 = 3852, \quad b_8 = 5310$$

durch $b_j := a_jb_{j-1} + b_{j-2}$. Wir setzen

$$b := (b_0, b_1, \ldots, b_8)^t = (194, 195, 974, 7013, 7987, 7760, 30974, 3852, 5310)^t.$$

Der modulo a reduzierte Vektor b^2 (im Intervall $]-\frac{a}{2}, \frac{a}{2}]$) des elementweise quadrierten Vektors b lautet

$$b^2 \equiv (-313, 76, -49, 265, -100, 7, -155, 155, -7)^t \bmod a.$$

Mittels der Probedivision verwerfen wir all diejenigen Einträge des Vektors b^2 wieder, die nicht F_{20}-glatt sind und es verbleibt der Vektor

$$b^2 \equiv (76, -49, -100, 7, -7)^t \bmod a.$$

Diese Einträge faktorisieren wir mit der Probedivision und erhalten die Exponentenmatrix

$$A = \begin{array}{c} \\ 76 \\ -49 \\ -100 \\ 7 \\ -7 \end{array} \begin{array}{c} \begin{array}{ccccc} -1 & 2 & 5 & 7 & 19 \end{array} \\ \begin{pmatrix} 0 & 2 & 0 & 0 & 1 \\ 1 & 0 & 2 & 0 & 0 \\ 1 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \equiv \begin{array}{c} \\ 76 \\ -49 \\ -100 \\ 7 \\ -7 \end{array} \begin{array}{c} \begin{array}{ccccc} -1 & 2 & 5 & 7 & 19 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \bmod 2,$$

wobei wir die Spalten ohne von Null verschiedenen Eintrag weglassen. Die zweite und die dritte Zeile von A summieren sich zum Nullvektor modulo 2, also setzen wir $A_0 = \{2, 3\}$ und erhalten nach (14.9)

$$x \equiv 37742 \bmod 37949 \quad \text{und} \quad y \equiv 37879 \bmod 37949.$$

Da $37742 \not\equiv \pm 37879 \bmod 37949$ gilt, erhalten wir den echten Teiler $p = \gcd(x-y, a) = \gcd(-137, 37949) = 137$.

Algorithmus 15.1.2. (Kettenbruchalgorithmus)
Wir wollen eine zusammengesetzte ungerade Zahl a faktorisieren, die keine echte Potenz ist.

1. Teste, ob a eine echte Potenz ist. Wenn nicht, fahre mit Punkt 2 fort, sonst haben wir einen echten Teiler gefunden.
2. Wähle eine Faktorbasis $F_{p_r} = \{-1, p_1, p_2, \ldots, p_r\}$ für eine Primzahl p_r.
3. Berechne die Kettenbruchentwicklung $\sqrt{a} = [a_0; \overline{a_1, \ldots a_s, 2a_0}]$.
4. Berechne die Zähler b_j der j-ten Näherungsbrüche durch

$$b_0 := a_0, \ b_1 := a_0a_1 + 1 \text{ und } b_j = a_jb_{j-1} + b_{j-2} \text{ für } j \geq 2.$$

5. Berechne mindestens $r+2$ F_{p_r}-glatte Elemente $b_j^2 \bmod a$ inklusive der Exponenten e_{j_i} der Primfaktorzerlegung von $b_j^2 \bmod a$ durch die Probedivision.

6. Erstelle die Matrix $A = (e_{j_i})_{(1\leq j\leq r+2, 0\leq i\leq r)}$ nach Gleichung (14.4) und finde eine Teilmenge $A_0 \subseteq \{1,\ldots,r+2\}$ (etwa durch Gauß'sche Elimination), sodass

$$\sum_{j\in A_0} (e_{j_i})_{(0\leq i\leq r)} \equiv \vec{0} \bmod 2$$

   gilt.

7. Setze $x \equiv \prod_{j\in A_0} b_j \bmod a$ und $y \equiv (-1)^{\frac{\sum_{j\in A_0} e_{j_0}}{2}} \prod_{i=1}^{r} p_i^{\frac{\sum_{j\in A_0} e_{j_i}}{2}} \bmod a$.

8. Teste, ob $x \not\equiv \pm y \bmod a$ gilt. Wenn nicht, wähle eine neue Teilmenge A_0 und erhöhe gegebenenfalls die Anzahl an Zeilen von A. Sonst berechne die Teiler

$$p = \gcd(x \pm y, a).$$

15.2 Laufzeit und Eigenschaften

Der Kettenbruchalgorithmus war eine lange Zeit der effizienteste Algorithmus zum Faktorisieren, bevor das quadratische Sieb (Kapitel 16) und das Zahlkörpersieb (Kapitel 17) entwickelt wurden. Besonders als in den 70er-Jahren für damalige Zeiten schnelle Computer zum Einsatz kamen und durch Brillhart und Morrison eine entsprechende Implementierung entwickelt wurde, konnte eine neue Dimension der Faktorisierung erreicht werden. Insbesondere konnte die Fermat-Zahl $F_7 = 2^{2^7}+1$ faktorisiert werden (siehe [44]), eine Zahl mit 39 Dezimalzahlen. Die Laufzeit vom Kettenbruchalgorithmus ist subexponentiell und liegt mit heuristischen Annahmen bei

$$L_a[\tfrac{1}{2}; \sqrt{2}] = O\left(e^{(\sqrt{2}+o(1))\sqrt{n_a \log(n_a)}}\right).$$

Der entscheidende Vorteil gegenüber der allgemeinen Methode von Legendre ruht in Korollar 5.0.6. In Kapitel 14 haben wir beliebige Zahlen b_j gewählt, sodass es für $b_j^2 \bmod a$ unwahrscheinlich ist F_{p_r}-glatt zu sein. Beim Kettenbruchalgorithmus wählen wir im Gegensatz dazu spezielle Zahlen b_j, die die Größe von $b_j^2 \bmod a$ beschränken und zwar nach Korollar 5.0.6 durch

$$|b_j^2 \bmod a| < 2\sqrt{a}.$$

Dadurch steigt die Wahrscheinlichkeit für b_j^2 F_{p_r}-glatt zu sein, und – da der Großteil des Rechenaufwands vom Kettenbruchalgorithmus in der Faktorisierung der b_j^2 steckt – sinkt gleichzeitig der Rechenaufwand für den Algorithmus. Aber trotz dieser Beschränkung der b_j^2 besteht weiterhin das Problem, dass zu viel Zeit in der Faktorisierung der b_j^2 verschwindet. Eine mögliche Verbesserung wäre statt der Probedivision ein schnelleres Faktorisierungsverfahren, wie beispielsweise eines der Pollard-Methoden, einzusetzen. Aber aufgrund der Masse an Faktorisierungen (bei größeren Zahlen sind 10000 keine Seltenheit) ändert das nichts daran, dass in den Zerlegungen zu viel Zeit verloren geht. Eine Verbesserung dieses Problems steckt in der Idee des Siebens ($\rightsquigarrow$ Quadratisches Sieb und Zahlkörpersieb), wo wir uns das aufwändige Faktorisieren der b_j^2 durch die Probedivision ersparen.

16 Das quadratische Sieb

Wie der Kettenbruchalgorithmus basiert auch das quadratische Sieb (englisch: QS=Quadratic Sieve) auf der Idee von Fermat (Kapitel 13) und Legendre (Kapitel 14). Entwickelt wurde der Algorithmus Anfang der 80er-Jahre (1982) von Carl Pomerance (siehe [56]) und löste damals den Kettenbruchalgorithmus als schnellste Faktorisierungsmethode ab. Aber auch heute gilt das quadratische Sieb noch als eine der besten Möglichkeiten, beliebige Zahlen zu faktorisieren. Als noch effizienter gilt aktuell das Zahlkörpersieb – mehr dazu in Kapitel 17.
Wir berechnen wie in Kapitel 14 mittels der Kongruenz

$$x^2 \equiv y^2 \bmod a$$

einen Faktor p von a. Dazu bestimmen wir analog zu Legendres Methode eine Faktorbasis F_{p_r} und Elemente b_j, sodass b_j^2 F_{p_r}-glatt ist. Der große Unterschied liegt wie beim Kettenbruchalgorithmus darin, wie wir die Elemente b_j^2 erzeugen. Beim Legendre-Algorithmus wählen wir dafür beliebige Elemente und hoffen, dass diese zum Quadrat F_{p_r}-glatt sind. Beim Kettenbruchalgorithmus betrachten wir systematisch die j-ten Teilbrüche der Kettenbruchentwicklung von $\sqrt{a}$ und wählen deren Zähler b_j. Der große Rechenaufwand beim Überprüfen der Glattheit ist aber weiterhin vorhanden.

16.1 Der Algorithmus

Sei a eine zusammengesetzte ungerade natürliche Zahl, die keine Potenz ist. Wir bestimmen die Zahl $\lfloor\sqrt{a}\rfloor$ und definieren für eine festgelegte gerade Schranke

$$M \approx L_a[\tfrac{1}{2}; 1] = O\left(e^{\sqrt{\log(a)\log\log(a)}}\right) \in \mathbb{N}$$

das Intervall

$$I := \left[\lfloor\sqrt{a}\rfloor - \frac{M}{2}, \lfloor\sqrt{a}\rfloor + \frac{M}{2}\right] \cap \mathbb{N}.$$

Weiters wählen wir eine Primzahl $p_r \approx \sqrt{M}$ für die Faktorbasis F_{p_r}. Für eine genaue Herleitung der Größenordnungen von M und p_r siehe Kapitel 16.3.
Wir betrachten nun das Polynom

$$f(b) := b^2 - a$$

und die Menge

$$S = \{f(b_j) \mid b_j \in I\}$$

für Zahlen $b_j \in I$. Die Elemente $b_j^2 \in S$ (eigentlich haben die Elemente von S die Form $b_j^2 - a$, wir kürzen sie aber mit b_j^2 ab) besitzen modulo a einen betragsmäßig geringen Wert und sind durch $M\sqrt{a} + M^2 \approx M\sqrt{a}$ beschränkt. Diese Schranke ist zwar nicht so klein wie beim Kettenbruchalgorithmus (vergleiche mit Korollar 5.0.6), doch beim quadratischen Sieb werden im Gegensatz zum Kettenbruchalgorithmus auch garantiert geringere Werte angenommen, wenn die Zahlen b_j nahe bei $\sqrt{a}$ liegen. Durch den Einsatz eines sogenannten Siebs – das im Folgenden erklärt wird – ersparen wir uns beim quadratischen Sieb zusätzlich die Probedivision bei der Überprüfung der Glattheit.
Ist m eine natürliche Zahl mit $m \mid f(b)$, so folgt

$$m \mid f(b + km) \quad \forall k \in \mathbb{N}, \tag{16.1}$$

da

$$f(b+km) = (b+km)^2 - a = b^2 + 2km + k^2m^2 - a = \underbrace{f(b)}_{m|} + \underbrace{2km}_{m|} + \underbrace{k^2m^2}_{m|} \Rightarrow m \mid f(b+km)$$

gilt. Wählen wir $m = q \in F_{p_r}$ und finden ein Element $b_j \in I$, sodass $q \mid f(b_j) = b_j^2 - a$ gilt, so folgt automatisch $q \mid f(b_j + kq) = (b_j + kq)^2 - a \quad \forall k \in \mathbb{N}$. Schreiben wir die Menge S als Liste an und ist das Element $b_j^2 - a$ durch q teilbar, dann ist auch das Element um q Schritte weiter unten (bzw. oben) in der Liste wieder durch q teilbar und das Element um weitere q Schritte wieder usw., ohne dass eine Division notwendig wäre ($\rightsquigarrow$ Sieben). Um ein b_j für eine vorgegebene Primzahl q zu finden, sodass $q \mid f(b_j) = b_j^2 - a$ gilt, lösen wir die Kongruenz

$$b_0^2 \equiv a \bmod q \quad (\iff q \mid b_0^2 - a). \tag{16.2}$$

Eigenschaft (16.2) bedeutet zusätzlich, dass wir die Faktorbasis F_{p_r} verkleinern können, denn wir nehmen lediglich die Primzahlen $q \leq p_r$ in F_{p_r} auf, sodass a ein quadratischer Rest modulo q ist, also $\left(\frac{a}{q}\right) = 1$ gilt (siehe Kapitel 4). Die Primzahl $q = 2$ sowie -1 nehmen wir an dieser Stelle immer in die Faktorbasis auf. Für alle anderen Primzahlen $\leq p_r$ besitzt Gleichung (16.2) gar keine Lösung b_0 und werden damit aus der Faktorbasis entfernt. Die neue, verkleinerte Faktorbasis bezeichnen wir mit F'_{p_r}.
Wir wenden Eigenschaft (16.1) aber nicht nur auf Primzahlen q, sondern auch auf Primzahlpotenzen q^i an. Folglich können wir so alle Elemente $b_j^2 \in S$ vollständig faktorisieren, die F'_{p_r}-glatt sind.
Wie sieht das Verfahren praktisch aus? Wir schreiben die Menge S als Liste an, wählen eine ungerade Primzahl q ($q = 2$ und -1 behandeln wir als Spezialfälle später) aus der Faktorbasis F'_{p_r} und beginnen mit dem Siebvorgang. Nach Satz 14.0.1 gibt es für Gleichung (16.2) genau zwei Lösungen $\pm b_0 \bmod q$ und somit ist $f(\pm b_0)$ durch q teilbar. Daraus folgt mit Gleichung (16.1), dass für alle $b_j \in I$ mit

$$b_j \equiv \pm b_0 \bmod q$$

die Zahl b_j^2 ($= b_j^2 \bmod a$) durch q teilbar ist. Nun dividieren wir all diese $b_j^2 \in S$ durch q und schreiben das Ergebnis statt dessen hin. Als nächstes sieben wir mit q^2. Wir berechnen wieder die Lösungen $\pm b_0 \bmod q^2$ von $b_0^2 \equiv a \bmod q^2$, bestimmen diejenigen b_j mit $b_j \equiv \pm b_0 \bmod q^2$ und dividieren die entsprechenden $b_j^2 \in S$ durch q (also insgesamt durch q^2, da nur solche b_j^2 in Frage kommen, die bereits durch q dividiert wurden). Dann sieben wir mit q^3, mit q^4 usw., bis es bei einer Primzahlpotenz q^i keine passenden b_j mehr gibt, die $b_j \equiv \pm b_0 \bmod q^i$ erfüllen, und wir zur nächsten Primzahl schreiten können. Dieses Prozedere führen wir mit allen Primzahlen der Faktorbasis und deren Potenzen durch.
Beim Spezialfall $q = -1$ ändern wir lediglich bei allen negativen b_j^2 das Vorzeichen. Für $q = 2$ und deren Potenzen verfahren wir analog zu den ungeraden Primzahlen und erhalten je nach Gestalt von a keine, eine, zwei oder vier Lösungen b_0 aus Kongruenz (16.2) (siehe Kapitel 4). Haben wir mit allen Primzahlen den Siebvorgang durchgeführt, so sind genau diejenigen b_j^2 F'_{p_r}-glatt – und können somit als Produkt von Primzahlen aus F'_{p_r} geschrieben werden –, bei denen am Ende in der Liste 1 steht.

Beispiel 16.1.1. Sei $a = 703$, $M = 8$ und $F_{p_r} = F_{13} = \{-1, 2, 3, 5, 7, 11, 13\}$, dann ist $\lfloor \sqrt{a} \rfloor = 26$, $I = [26 - 4, 26 + 4] \cap \mathbb{N} = \{22, 23, 24, 25, 26, 27, 28, 29, 30\}$ und $f(b) = b^2 - 703$. Als erstes entfernen wir alle p_i (außer -1 und 2) aus F_{13}, für die a kein quadratischer Rest modulo p_i ist. Es gilt nach Kapitel 4

$$\left(\frac{703}{3}\right) = 1, \quad \left(\frac{703}{5}\right) = -1, \quad \left(\frac{703}{7}\right) = -1, \quad \left(\frac{703}{11}\right) = -1, \quad \left(\frac{703}{13}\right) = 1$$

und wir verkleinern die Faktorbasis auf $F'_{13} = \{-1, 2, 3, 13\}$. Nun definieren wir zur besseren Übersicht die Matrix

$$\begin{pmatrix} b \\ b^2 \end{pmatrix} = \begin{pmatrix} 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \\ -219 & -174 & -127 & -78 & -27 & 26 & 81 & 138 & 197 \end{pmatrix}$$

deren erste Zeile b aus $b_j \in I$ und deren zweiten Zeile b^2 aus $f(b_j) = b_j^2 - a$ besteht. Wir beginnen den Siebvorgang bei $p_0 = -1$ und wechseln bei allen negativen Einträgen von b^2 das Vorzeichen, also

$$\begin{pmatrix} b \\ b^2 \end{pmatrix} = \begin{pmatrix} 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \\ 219 & 174 & 127 & 78 & 27 & 26 & 81 & 138 & 197 \end{pmatrix}.$$

Jetzt sieben wir mit $p_1 = 2$. Da $703 \equiv 1 \bmod 2$ gilt, dividieren wir alle Werte b_j^2 mit ungeradem b_j durch 2. Das sind 174 ($\leftrightarrow 23 \equiv 1 \bmod 2$), 78 ($\leftrightarrow 25 \equiv 1 \bmod 2$), 26 ($\leftrightarrow 27 \equiv 1 \bmod 2$) und 138 ($\leftrightarrow 29 \equiv 1 \bmod 2$), also

$$\begin{pmatrix} b \\ b^2 \end{pmatrix} = \begin{pmatrix} 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \\ 219 & 87 & 127 & 39 & 27 & 13 & 81 & 69 & 197 \end{pmatrix}.$$

Da 703 kein quadratischer Rest modulo 4 ist, sieben wir als nächstes mit $p_2 = 3$. Dafür berechnen wir die Lösungen $b_0 \equiv \pm 1 \bmod 3$ der Kongruenz

$$b_0^2 \equiv 703 \equiv 1 \bmod 3.$$

Wir dividieren alle b_j^2 mit $b_j \equiv \pm 1 \bmod 3$ durch 3, also 219 ($\leftrightarrow 22 \equiv 1 \bmod 3$), 174 ($\leftrightarrow 23 \equiv -1 \bmod 3$), 39 ($\leftrightarrow 25 \equiv 1 \bmod 3$), 27 ($\leftrightarrow 26 \equiv -1 \bmod 3$), 81 ($\leftrightarrow 28 \equiv 1 \bmod 3$) und 69 ($\leftrightarrow 29 \equiv -1 \bmod 3$):

$$\begin{pmatrix} b \\ b^2 \end{pmatrix} = \begin{pmatrix} 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \\ 73 & 29 & 127 & 13 & 9 & 13 & 27 & 23 & 197 \end{pmatrix}.$$

Als nächstes sieben wir mit $p_2^2 = 3^2$ und berechnen die Lösungen $b_0 \equiv \pm 1 \bmod 9$ von

$$b_0^2 \equiv 703 \equiv 1 \bmod 3^2.$$

Wir dividieren wieder alle b_j^2 durch 3 (also insgesamt durch 9), sodass $b_j \equiv \pm 1 \bmod 9$ gilt, also 9 ($\leftrightarrow 26 \equiv -1 \bmod 9$) und 27 ($\leftrightarrow 28 \equiv 1 \bmod 9$):

$$\begin{pmatrix} b \\ b^2 \end{pmatrix} = \begin{pmatrix} 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \\ 73 & 29 & 127 & 13 & 3 & 13 & 9 & 23 & 197 \end{pmatrix}.$$

Für $p_3^3 = 3^3$ und $p_3^4 = 3^4$ erhalten wir mit analoger Vorgehensweise

$$\begin{pmatrix} b \\ b^2 \end{pmatrix} = \begin{pmatrix} 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \\ 73 & 29 & 127 & 13 & 1 & 13 & 1 & 23 & 197 \end{pmatrix}.$$

Die Kongruenz $b_0^2 \equiv 703 \equiv 1 \bmod 3^5$ besitzt zwar die Lösungen $b_0 \equiv \pm 109 \bmod 3^5$, doch diese treten unter den b_j nicht mehr auf, weswegen wir zur nächsten Primzahl $p_6 = 13$ schreiten können. Die Lösungen von

$$b_0^2 \equiv 703 \equiv 1 \bmod 13$$

sind $b_0 \equiv \pm 1 \bmod 13$ und wir dividieren 13 ($\leftrightarrow 25 \equiv -1 \bmod 9$) und 13 ($\leftrightarrow 27 \equiv 1 \bmod 9$) durch 13 und erhalten

$$\begin{pmatrix} b \\ b^2 \end{pmatrix} = \begin{pmatrix} 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \\ 73 & 29 & 127 & 1 & 1 & 1 & 1 & 23 & 197 \end{pmatrix}.$$

Für die Kongruenz $b_0^2 \equiv 703 \bmod 13^2$ erhalten wir die Lösungen $b_0 \equiv \pm 14 \bmod 13^2$, die aber zu keinem der b_j passen. Damit beenden wir das Siebverfahren und lesen ab, welche der b_j^2 F'_{13}-glatt (und damit auch F_{13}-glatt) sind. Das sind $b_4^2 = -127$, $b_5^2 = -78$, $b_6^2 = -27$ und $b_7^2 = 26$.

Beispiel 16.1.1 wäre mit der Probedivision oder vielleicht bereits mit bloßem Auge wesentlich schneller gegangen, doch verdeutlicht das Beispiel die Funktionsweise des Siebens, einem entscheidenden Bestandteil beim quadratischen Sieb. Haben wir genügend viele F_{p_r}-glatte Elemente durch das Siebverfahren gefunden, so können wir analog zum Legendre-Algorithmus (Kapitel 14) und zum Kettenbruchalgorithmus (Kapitel 15) einen Teiler p von a berechnen.

16.1.1 Zusammenfassung

Wir wollen einen echten Faktor p von einer natürlichen Zahl a berechnen. Dazu bestimmen wir das Intervall $I := [\lfloor\sqrt{a}\rfloor - \frac{M}{2}, \lfloor\sqrt{a}\rfloor + \frac{M}{2}] \cap \mathbb{N}$, die Menge $S = \{b^2 - a \mid b \in I\}$ und eine Faktorbasis F_{p_r}. Aus der Faktorbasis entfernen wir alle ungeraden Primzahlen p_i mit

$$\left(\frac{a}{p_i}\right) \neq 1.$$

Weiters stellen wir eine sogenannte Siebtabelle auf, in der wir für jedes Element $b_j \in I$ eine Zeile anfertigen, um die Primfaktorzerlegung der F'_{p_r}-glatten Elemente zu bestimmen. Die Zeile von b_j besteht aus dem Element selbst, dem Wert $b_j^2 \in S$, dem reduzierten (also den durch die Primzahlen dividierten) Wert b_j^2 und je einem Spalteneintrag für jeden Primzahlexponenten e_{j_i} entsprechend der Anzahl der Elemente von F'_{p_r} (siehe Tabelle 16.1.4). Mit dem oben ausgeführten Siebvorgang überprüfen wir, welche der b_j^2 F'_{p_r}-glatt sind. Dividieren wir dabei eine Zahl b_j^2 durch eine Primzahl $p_i \in F'_{p_r}$ oder durch -1, erhöhen wir – bei Null beginnend – den entsprechenden Tabelleneintrag um 1. Am Ende des Siebvorgangs entfernen wir alle Zeilen b_j^2 aus der Tabelle, in der keine 1 in der Spalte „b_j^2 reduziert“ steht und folglich nicht F'_{p_r}-glatt sind. Für die übrig gebliebenen glatten Zahlen b_j^2 geben wir die Primfaktorzerlegung

$$b_j^2 = (-1)^{e_{j_0}} \prod_{i=1}^{r'} p_i^{e_{j_i}}$$

an. Setzen wir $r' + 1 = |F'_{p_r}|$ und haben zumindest $r' + 2$ Tabelleneinträge erzeugt, so schreiben wir, wie in den vorigen Kapiteln, die Exponenten als Matrix $A = (e_{j_i})_{(1 \leq j \leq r'+2, 0 \leq i \leq r')}$ an. Aus der Matrix A wählen wir eine Teilmenge $A_0 \subseteq \{1, \ldots, r' + 1\}$, sodass

$$\sum_{j \in A_0} \begin{pmatrix} e_{j_0} \\ e_{j_1} \\ e_{j_2} \\ \vdots \\ e_{j_r} \end{pmatrix} = \begin{pmatrix} \sum_{j \in A_0} e_{j_0} \\ \sum_{j \in A_0} e_{j_1} \\ \sum_{j \in A_0} e_{j_2} \\ \vdots \\ \sum_{j \in A_0} e_{j_r} \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \mod 2$$

gilt und setzen

$$x :\equiv \prod_{j \in A_0} b_j \mod a \quad \text{und} \quad y :\equiv (-1)^{\frac{\sum_{j \in A_0} e_{j_0}}{2}} \prod_{i=1}^{r'} p_i^{\frac{\sum_{j \in A_0} e_{j_i}}{2}} \mod a.$$

Gilt $x \not\equiv \pm y \mod a$, so haben wir die echten Teiler $p = \gcd(x \pm y, a)$ gefunden. Andernfalls müssen wir eine neue Teilmenge A_0 wählen und gegebenenfalls neue Zeilen zur Matrix A hinzufügen.

Algorithmus 16.1.2. (Quadratisches Sieb)
Wir berechnen einen Teiler einer zusammengesetzten ungeraden natürlichen Zahl a, die keine echte Potenz ist.

1. Teste, ob a eine echte Potenz ist. Wenn nicht, fahre mit Punkt 2 fort, sonst haben wir einen echten Teiler gefunden.

2. Setze $M = \lfloor L_a[\frac{1}{2}; 1]\rfloor$,
   $p_r \approx \sqrt{M} \in \mathbb{P}$ und
   $I = \left[\lfloor\sqrt{a}\rfloor - \frac{M}{2}, \lfloor\sqrt{a}\rfloor + \frac{M}{2}\right] \cap \mathbb{N}$.

3. Alle ungeraden Primzahlen $p_i \leq p_r$ mit $\left(\frac{a}{p_i}\right) = 1$, inklusive $q = 2$ und -1 nehmen wir in die Faktorbasis F'_{p_r} auf. Setze $r' + 1 = |F'_{p_r}|$.

4. Siebe aus der Folge $f(b_j) = b_j^2 - a$ mit $b_j \in I$ zumindest $r' + 2$ F'_{p_r}-glatte Werte heraus und notiere deren Primfaktorzerlegung

$$f(b_j) = (-1)^{e_{j_0}} \prod_{i=1}^{r'} p_i^{e_{j_i}}.$$

5. Definiere die Matrix

$$A = (e_{j_i})_{(1\leq j\leq r'+2, 0\leq i\leq r')}$$

   und finde eine Teilmenge A_0 an Zeilen von A (z.B. mit Gauß'scher Elimination), sodass

$$\sum_{j\in A_0} (e_{j_i})_{(0\leq i\leq r')} \equiv \vec{0} \bmod 2$$

   gilt.

6. Setze $x \equiv \prod_{j\in A_0} b_j \bmod a$ und $y \equiv (-1)^{\frac{\sum_{j\in A_0} e_{j_0}}{2}} \prod_{i=1}^{r} p_i^{\frac{\sum_{j\in A_0} e_{j_i}}{2}} \bmod a$.
   Teste, ob $x \not\equiv \pm y \bmod a$ gilt. Wenn nicht, wähle eine neue Teilmenge A_0 und erhöhe gegebenenfalls die Anzahl an Zeilen von A. Sonst berechne die Teiler

$$p = \gcd(x \pm y, a).$$

Beispiel 16.1.3. Sei $a = 1081$ ($\Rightarrow \lfloor\sqrt{1081}\rfloor = 32$), $F_{p_r} = F_7 = \{-1, 2, 3, 5, 7\}$ und $M = 12$. Wir setzen

$$I = [32 - 6, 32 + 6] \cap \mathbb{N} = \{26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38\}$$

und

$$S = \{b^2 - 1081 \mid b \in I\} = \{-405, -352, -297, -240, -181, -120, -57, 8, 75, 144, 215, 288, 363\}.$$

Es gilt $F'_7 = F_5 = \{-1, 2, 3, 5\}$, weil 1081 nur modulo 7 kein quadratischer Rest ist. Wir führen den Siebvorgang wie in Beispiel 16.1.1 durch und erhalten die Siebtabelle 16.1.4 für $a = 1081$.
Wir eliminieren jetzt alle b_j^2 aus der Tabelle, die nicht F_5-glatt sind, also alle b_j^2, in der keine 1 in der Spalte „b_j^2 reduziert“ steht und schreiben die Exponenten der Tabelle in die Exponentenmatrix

$$A = \begin{array}{c} \\ 26 \\ 29 \\ 31 \\ 33 \\ 34 \\ 35 \\ 37 \end{array} \begin{array}{c} \begin{array}{cccc} -1 & 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 1 & 0 & 4 & 1 \\ 1 & 4 & 1 & 1 \\ 1 & 3 & 1 & 1 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 4 & 2 & 0 \\ 0 & 5 & 2 & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ \text{-405} \\ \text{-240} \\ \text{-120} \\ 8 \\ 75 \\ 144 \\ 288 \end{array} \equiv \begin{array}{c} \begin{array}{cccc} -1 & 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \bmod 2.$$

b_j	b_j^2	b_j^2 reduziert	-1	2	3	5
26	-405	1	1	0	4	1
27	-352	11	1	5	0	0
28	-297	11	1	0	3	0
29	-240	1	1	4	1	1
30	-181	181	1	0	0	0
31	-120	1	1	3	1	1
32	-57	19	1	0	1	0
33	8	1	0	3	0	0
34	75	1	0	0	1	2
35	144	1	0	4	2	0
36	215	43	0	0	0	1
37	288	1	0	5	2	0
38	363	121	0	0	1	0

Tabelle 16.1.4: Siebtabelle zu $a = 1081$

Wählen wir beispielsweise $A_0 = \{2, 3, 7\}$, so erhalten wir

$$x \equiv 833 \bmod 1081 \quad \text{und} \quad y \equiv 363 \bmod 1081.$$

Da $833 \not\equiv \pm 363 \bmod 1081$ gilt, berechnen wir den Teiler $p = \gcd(x - y, a) = \gcd(470, 1081) = 47$.

16.2 Varianten des quadratischen Siebs

Gegenüber dem Kettenbruchverfahren ersparen wir beim quadratischen Sieb beim Überprüfen der F_{p_r}-Glattheit aufgrund des Siebverfahrens unnötige Probedivisionen. Trotzdem geht bei zahlreichen überflüssig durchgeführten Divisionen viel Zeit verloren, weil viele der b_j^2 am Ende nicht F_{p_r}-glatt sind. Rechnen wir mit den Logarithmen von b_j und den Primzahlen p_i, so können wir statt aufwendiger Divisionen „kostengünstigere“ Subtraktionen durchführen.

16.2.1 Logarithmisches Sieben

Es sei eine ganze Zahl b und ihre Primfaktorzerlegung $b = \prod_i p_i^{e_i}$ gegeben, dann gilt

$$\log(|b|) = \sum_i e_i \log(p_i).$$

Nehmen wir von allen Elementen $|b_j^2| \in S$ und $p_i \in F'_{p_r}$ den Logarithmus und ziehen beim Sieben $\log(p_i)$ von $\log(|b_j^2|)$ ab, so ersparen wir uns das „teure“ Dividieren. Würden wir dies mit den exakten Logarithmuswerten durchführen, so wäre ein b_j^2 genau dann F_{p_r}-glatt, wenn am Ende des Siebens 0 in der Spalte „b_j^2 reduziert“ herauskommt. Es zeigt sich, dass es gar nicht notwendig ist mit den exakten, sondern es genügt mit den gerundeten Logarithmuswerten $\log(p_i)$ und $\log(|b_j^2|)$ zu rechnen. Ist der Betrag von b_j^2 am Ende des Siebens klein genug, so ist das Element mit großer Wahrscheinlichkeit F_{p_r}-glatt. Genauer ist b_j^2 genau dann glatt, wenn der Betrag des Ergebnisses in der Spalte „b_j^2 reduziert“ beim Sieben mit den Logarithmen kleiner ist als die kleinste Primzahl $> p_r$. Haben wir auf diese Weise ausreichend viele glatte Werte erzeugt, berechnen wir wie in der „Basis-Version“ des QS einen Teiler.

Beispiel 16.2.1. Sei $a = 209$, $F_{p_r} = F_7$ und $M = 4$, dann ist

$$I = \{12, 13, 14, 15, 16\}, \quad S = \{-65, -40, -13, 16, 47\},$$

und wir reduzieren die Faktorbasis F_7 auf $F_7' = \{-1, 2, 5\}$. Wir bezeichnen mit $[\log(X)]$ die elementweise gerundeten Logarithmen der Beträge der Menge X, dann gilt

$$[\log(F_7' \backslash \{-1\})] = \{1, 2\} \quad \text{und} \quad [\log(S)] = \{4, 4, 3, 3, 4\}.$$

Alle b_j^2, die am Ende des Siebens dem Betrag nach weniger als $\log(11) \approx 2$ ausmachen, sind F_7-glatt. Führen wir das oben angeführte Siebverfahren mit Subtraktionen statt mit Divisionen durch, so hat die Menge $[\log(S)]$ am Ende die Form

$$[\log(S)] = \{2, -1, 3, -1, 4\}.$$

Der zweite und der vierte Wert aus $[\log(S)]$ sind vom Betrag her kleiner als 2, also die entsprechenden Werte -40 und 16 genau die F_7-glatten Elemente der Menge S.

16.2.2 Multipolynomiales quadratisches Sieb

Damit wir gute Chancen auf die Glattheit der b_j^2 haben, suchen wir vom Betrag her möglichst kleine b_j^2. Ist a groß, so sind die Zahlen b_j^2 klein, solange wir b_j nahe bei $\sqrt{a}$ wählen. Damit wir aber linear abhängige Zeilen in der Matrix A erzeugen können, brauchen wir viele solcher Werte b_j und sind damit gezwungen, auch Zahlen b_j zu verwenden, die großen Abstand zu $\sqrt{a}$ haben. Da wir nur mit dem einen Polynom $f(x) = x^2 - a$ arbeiten und $f(b_j) = b_j^2 - a$ quadratisch im Eingabewert b_j wachsen, erzeugen diese eine vom Betrag her große Zahl b_j^2. Wir können alternativ nicht nur mit einem, sondern mit vielen Polynomen

$$g(x) = kx^2 + 2mx + n, \text{ mit } k, m, n \in \mathbb{Z}$$

arbeiten. Der dadurch erweiterte Algorithmus nennt sich multipolynomiales quadratisches Sieb (englisch: MPQS=Multiple Polynomial Quadratic Sieve) und wurde von P. Montgomery und unabhängig davon von J.A. Davis, D.B. Holdridge und G.J. Simmons (siehe [24]) entwickelt. Definieren wir das Polynom $g(x)$ wie eben mit

$$m^2 - a = kn \tag{16.3}$$

und fordern, dass $k = u^2 v$ mit einer natürlichen Zahl u und einer F_{p_r}-glatten Zahl v gilt, dann folgt

$$k \cdot g(x) = k^2x^2 + 2kmx + kn = k^2x^2 + 2kmx + m^2 - a = (kx + m)^2 - a \equiv (kx + m)^2 \bmod a. \tag{16.4}$$

Auf der rechten Seite in (16.4) steht ein Quadrat modulo a und auf der linken Seite ein Quadrat ($\rightsquigarrow u^2$) mal einer F_{p_r}-glatten Zahl ($\rightsquigarrow v$) mal einer Zahl von der wir hoffen, dass sie auch glatt ist ($\rightsquigarrow g(x)$). Damit faktorisieren wir analog zur Basis-Version des QS mittels des Siebverfahrens und dem Matrixschritt eine natürliche Zahl.
Worin liegt der Gewinn der Einführung der Polynome $g(x)$? Laut den folgenden Überlegungen können wir die Werte $g(x)$ relativ klein wählen. Das Minimum $\min_x |k \cdot g(x)|$ modulo a liegt nach

$$kg\left(-\frac{m}{k}\right) \overset{(16.4)}{\equiv} \left(-k\frac{m}{k} + m\right)^2 \equiv 0 \bmod a \tag{16.5}$$

bei $x = -\frac{m}{k}$. Das Siebintervall

$$I = \left[-\frac{m}{k} - \frac{M}{2}, -\frac{m}{k} + \frac{M}{2}\right]$$

der Länge M zentrieren wir um den Punkt $-\frac{m}{k}$. Nun wählen wir die Werte k, m und n so, dass wir die obere Schranke von $g(x)$ minimieren. Angenommen, wir haben bereits ein ideales $k \in \mathbb{N}$ gefunden, so können wir m nach Eigenschaft (16.3) durch die minimale Lösung von

$$m^2 \equiv a \bmod k \tag{16.6}$$

wählen, also gilt $m \leq \frac{k}{2}$. Damit (16.6) eine Lösung hat, muss a auf jeden Fall ein quadratischer Rest modulo k sein. Für n setzen wir ebenfalls nach (16.3)

$$n = \frac{m^2 - a}{k}.$$

Jetzt wählen wir k so, dass $\max_{x \in I} |g(x)|$ minimal und a ein quadratischer Rest modulo k ist. Um das Maximum von $|g(x)|$ zu minimieren, muss

$$g\left(-\frac{m}{k}\right) = -g\left(-\frac{m}{k} + \frac{M}{2}\right) \tag{16.7}$$

gelten, da $g\left(-\frac{m}{k}\right) \leq g(x) \leq g\left(-\frac{m}{k} + \frac{M}{2}\right)$ gilt. Gleichung (16.7) ist äquivalent zu

$$k^2 \left(\frac{M}{2}\right)^2 = 2m^2 - 2kn \iff k = \frac{2\sqrt{2(m^2 - kn)}}{M} \overset{(16.3)}{\iff} k = \frac{2\sqrt{2a}}{M}. \tag{16.8}$$

Mit dem in (16.8) definierten k erhalten wir für $g(x)$ die obere Schranke

$$\max_{x \in I} |g(x)| = \left|-g\left(-\frac{m}{k} + \frac{M}{2}\right)\right| \overset{(16.7)}{=} \left|g\left(-\frac{m}{k}\right)\right| \overset{(16.4)}{=} \left|\underbrace{\frac{\left(-k\frac{m}{k} + m\right)^2}{k}}_{=0} - \frac{a}{k}\right| \overset{(16.8)}{=} \frac{Ma}{2\sqrt{2a}} = \frac{M\sqrt{a}}{2\sqrt{2}}.$$

Vergleichen wir diese Schranke mit der der Basisversion des quadratischen Siebs, die bei $M\sqrt{a}$ liegt, so sparen wir einen Faktor $2\sqrt{2}$ in der Beschränkung der b_j^2 ein. Noch mehr gewinnen wir durch die unterschiedlichen Polynome $g(x)$, die wir verwenden können. Sind die b_j^2 an einer beliebigen Stelle zu groß, wählen wir ein neues Polynom $g(x)$ und haben wieder Werte mit geringerer Größe, weswegen in der Praxis die Intervallgröße M beim MPQS nur noch in der Größenordnung

$$M \approx L_a[\frac{1}{2}; \frac{1}{2}] = O\left(e^{\frac{1}{2}\sqrt{\log(a)\log\log(a)}}\right)$$

gewählt wird. Nun geben wir konkret an, wie wir k wählen, wobei k nach (16.8) etwa die Größenordnung $\frac{2\sqrt{2a}}{M}$ haben, sich als Produkt einer F_{p_r}-glatten Zahl v und einem Quadrat u^2 schreiben lassen und a ein quadratischer Rest modulo k sein muss. Dazu wählen wir eine Primzahl $q \approx \sqrt{\frac{2\sqrt{2a}}{M}}$ mit $\left(\frac{a}{q}\right) = 1$ und setzen $k := q^2$. Mit dieser simplen Vorgehensweise erfüllen wir genau die gestellten Anforderungen.

16.2.3 Selbstinitialisierung und Variante mit großer Primzahl

Zwei weitere Erweiterungen des quadratischen Siebs sind die Selbstinitialisierung und die Variante mit großer Primzahl. Diese Erweiterungen finden sich auch bei der Implementierung der Funktion „$factor()$“ von PARI/GP wieder. Für genauere Details, wie auch für weitere Varianten des quadratischen Siebs, beispielsweise die schnelle Matrixvariante oder Zhangs spezielles quadratisches Sieb, sei der Leser auf [23], Kapitel 6.1 verwiesen.
Die sogenannte Selbstinitialisierung ist eine Erweiterung des MPQS (Kapitel 16.2.2). Beim MPQS besteht die Möglichkeit b_j^2 klein zu halten, indem wir das Polynom $g(x) = kx^2 + mx + n$ oft wechseln. Doch wie oft sollen wir das Polynom ändern, damit wir ein optimales, also möglichst schnelles Ergebnis erhalten? A priori würde man meinen, dass wir relativ oft ein neues Polynom $g(x)$

wählen sollten, damit b_j^2 klein bleibt. Weil die Berechnungen für die Parameter für neue Polynome ($\rightsquigarrow$ Initialisierung) bei häufigem Wechsel zu zeitintensiv sind, würde sich dies negativ auswirken. Bei der sogenannten Selbstinitialisierung wählen wir $k \in \mathbb{Z}$ so, dass wir viele Polynome $g(x)$ auf einmal erzeugen, um die Vorberechnungen nur einmal machen zu müssen. Dazu benötigen wir ein geeignetes k, das viele Lösungen für die Kongruenz (16.6) zur Verfügung stellt. Im vorigen Kapitel haben wir $k = q^2$ für eine passende Primzahl q gewählt. Das würde uns lediglich 2 Lösungen $\pm m$ liefern, von denen nur eine $m \leq \frac{k}{2}$ erfüllt. Da wir viele Lösungen aus Gleichung (16.6) wollen, wählen wir $k = q_1^2 \cdot q_2^2 \cdots q_s^2$ als Produkt von s verschieden Primzahlen, sodass a ein quadratischer Rest modulo k ist, und erhalten so mindestens 2^{s-1} Lösungen m von Gleichung (16.6) (siehe Satz 14.0.1). Von diesen Lösungen erfüllen mindestens 2^{s-2} die Abschätzung $m \leq \frac{k}{2}$, und damit erzeugen wir mindestens 2^{s-2} verschiedene Polynome $g(x)$.
Eine weitere Möglichkeit das quadratische Sieb zu erweitern, ist die Variante der großen Primzahl, die auch unabhängig von MPQS einsetzbar ist. Wir haben bereits in Kapitel 11.2.1 für Pollards $(p-1)$-Algorithmus (siehe auch Kapitel 12.2.1 für ECM) eine entsprechende Methode kennen gelernt, bei der wir potenzglatten Zahlen mit einem Primfaktor $> p_r$ zulassen. Haben wir für das QS eine Zahl $b_j^2 = c \cdot d$ mit einer Primzahl c mit $p_r < c \leq p_r^2$ und einer F_{p_r}-glatten Zahl d gefunden und finden wir ein weiteres $b_l^2 = c \cdot d'$ mit der gleichen Eigenschaft bei gleichem c, so können wir daraus eine weitere Zeile für unsere Matrix A gewinnen, ohne deren Spaltenanzahl erhöhen zu müssen. Aus

$$(b_j b_l)^2 \equiv dd'c^2 \bmod a$$

erhalten wir nämlich die Kongruenz

$$(b_j b_l c^{-1})^2 \equiv dd' \bmod a.$$

Wäre c nicht invertierbar, so hätten wir bereits einen Teiler $p = \gcd(c, a)$ gefunden.

16.3 Laufzeit und Eigenschaften

Neben dem Zahlkörpersieb (NFS) und der Elliptic Curve Method (ECM) gehört das quadratische Sieb (QS) zu den wichtigsten Faktorisierungsmethoden der heutigen Zeit. Durch den von C. Pomerance 1982 entwickelten Algorithmus konnten wesentlich größere Zahlen faktorisiert werden, als es bis dahin möglich war. Zuvor konnte man mit dem besten Verfahren, dem Kettenbruchalgorithmus, Zahlen mit rund 50 Dezimalstellen faktorisieren, während mit dem QS bald Zahlen mit bis zu 100 Stellen zerlegt werden konnten. Es wird vermutet, dass die Laufzeit mit einigen heuristischen Annahmen

$$L_a[\frac{1}{2}; 1] = O\left(\exp\left((1 + o(1))\sqrt{n_a \log(n_a)}\right)\right)$$

beträgt – ein Beweis steht noch aus. Wie kommt dieser Wert zustande? Für dessen skizzenhafte Herleitung – die wir im folgenden machen – sei bemerkt, dass der asymptotisch einzig relevante Anteil für die Laufzeit das Siebverfahren ausmacht. Alle anderen Berechnungen (zum Beispiel das Berechnen von $f(b_j)$ oder $\gcd(x \pm y, a)$) sind dagegen asymptotisch vernachlässigbar.

Definition 16.3.1. Die Funktion $\Psi : \mathbb{N} \times \mathbb{P} \to \mathbb{N}$ sei gegeben durch

$$\Psi(x, p_r) = |\ \{b \in \mathbb{N} \mid 1 \leq b \leq x,\ b\text{ist } F_{p_r}\text{-glatt}\}\ |,$$

also die Anzahl der F_{p_r}-glatten Zahlen $\leq x$.

Für eine beliebige Zahl $x \in \mathbb{N}$ ist daher die Wahrscheinlichkeit, dass wir eine F_{p_r}-glatte Zahl aus dem Intervall $[1, x]$ auswählen, gleich

$$\frac{\Psi(x, p_r)}{x}.$$

Daher müssen wir dem Kehrwert entsprechend

$$\frac{x}{\Psi(x, p_r)}$$

Zahlen wählen, um eine F_{p_r}-glatte Zahl zu finden. Da die von uns erzeugte Folge $f(b)$ eine spezielle Teilmenge aus dem Intervall $[1, x]$ ist, können wir nicht sicher sein, ob die Wahrscheinlichkeit für diese Werte glatt zu sein genauso groß ist wie für zufällig gewählte Folgen. Wir nehmen daher vereinfachend an, dass diese Wahrscheinlichkeiten gleich sind. Um die Größe von $f(b) = b^2 - a$ abzuschätzen, nehmen wir weiters an, dass die Zahlen b das Intervall

$$I := \left[\sqrt{a} - \frac{1}{2}a^{\epsilon}, \sqrt{a} + \frac{1}{2}a^{\epsilon}\right] \cap \mathbb{N}$$

für $0 < \epsilon < \frac{1}{2}$ durchlaufen, also gilt $|f(b)| < a^{\frac{1}{2}+\epsilon}$ für die Intervalllänge

$$x = a^{\frac{1}{2}+\epsilon}. \tag{16.9}$$

Damit die Matrix A garantiert linear abhängige Zeilen besitzt, erzeugen wir mindestens $r + 2 = \pi(p_r) + 2$ Werte $f(b)$. Die Laufzeit für den Siebvorgang für jeden dieser Werte beträgt nach [56]

$$\log(\log(p_r)),$$

das heißt, insgesamt beträgt die Laufzeit für den gesamten Siebvorgang für $\pi(p_r) \approx \pi(p_r) + 2$ Werte $f(b)$

$$\frac{\log(\log(p_r)) \cdot \pi(p_r) \cdot x}{\Psi(x, p_r)}. \tag{16.10}$$

Mit Hilfe des folgenden Resultats aus [18] können wir diesen Ausdruck minimieren.

Theorem 16.3.2. *Sei $\epsilon > 0$, dann gilt*

$$\frac{\Psi(x, x^{\frac{1}{u}})}{x} = u^{-u+o(1)}$$

gleichmäßig für $u \to \infty$ und $u < \frac{(1-\epsilon)\log(x)}{\log\log(x)}$.

Wählen wir für ein (noch) unbestimmtes u die Primzahl $p_r \approx x^{\frac{1}{u}}$ und schätzen mittels dem Primzahlsatz (Theorem 1.0.5)

$$\log(\log(p_r)) \cdot \pi(p_r) \approx p_r \approx x^{\frac{1}{u}}$$

ab, so erhalten wir aus (16.10) und Theorem 16.3.2 die Laufzeit

$$x^{\frac{1}{u}} \cdot u^u. \tag{16.11}$$

Wir wollen u so wählen, dass der Ausdruck (16.11) minimiert wird und nehmen dazu den Logarithmus aus (16.11), also

$$\frac{1}{u} \cdot \log(x) + u \cdot \log(u), \tag{16.12}$$

leiten (16.12) nach u ab und setzen den Ausdruck gleich Null, also

$$u^2(\log(u) + 1) = \log(x). \tag{16.13}$$

Aus der logarithmierten Gleichung (16.13) drücken wir

$$\log(u) = \frac{1}{2}\log(\log(x)) - \log(\log(u)+1) \approx \frac{1}{2}\log(\log(x)) \tag{16.14}$$

aus. Setzen wir dieses Resultat in Gleichung (16.13) ein, so erhalten wir

$$u \approx \sqrt{\frac{2\log(x)}{\log(\log(x))}}. \tag{16.15}$$

Aus diesem minimalen Wert für u berechnen wir die Größenordnung von p_r:

$$\begin{aligned} p_r &\approx x^{\frac{1}{u}} \approx x^{\sqrt{\frac{\log(\log(x))}{2\log(x)}}} \\ \Rightarrow \log(p_r) &\approx \sqrt{\frac{\log(\log(x))}{2\log(x)}} \cdot \log(x) \approx \sqrt{\frac{\log(\log(x)) \cdot \log(x)}{2}} \\ \Rightarrow p_r &\approx \exp\left(\left(\frac{1}{\sqrt{2}} + o(1)\right)\sqrt{\log(\log(x)) \cdot \log(x)}\right). \end{aligned} \tag{16.16}$$

Setzen wir den Wert p_r aus Gleichung (16.16) in (16.11) ein, so erhalten wir die Laufzeit

$$x^{\frac{1}{u}} u^u \approx \exp\left(\left(\sqrt{2} + o(1)\right)\sqrt{\log(\log(x)) \cdot \log(x)}\right). \tag{16.17}$$

Schließlich setzen wir $x = a^{\frac{1}{2}+\epsilon}$ aus Gleichung (16.9) in (16.16) bzw. (16.17) ein und wählen statt einem fixen $\epsilon < 0$ eine Funktion $o(1)$, die gegen Null konvergiert und erhalten

$$p_r \approx \exp\left(\left(\frac{1}{2} + o(1)\right)\sqrt{\log(\log(a)) \cdot \log(a)}\right) = L_a[\tfrac{1}{2}; \tfrac{1}{2}], \tag{16.18}$$

bzw.

$$x^{\frac{1}{u}} u^u \approx \exp\left((1 + o(1))\sqrt{\log(n_a) \cdot n_a}\right) = L_a[\tfrac{1}{2}; 1]. \tag{16.19}$$

Damit haben wir, neben der Laufzeit (16.19) des Algorithmus', auch den Wert p_r in (16.18) für die Faktorbasis F_{p_r} hergeleitet. Die Laufzeit vom quadratischen Sieb liegt genau im Bereich von ECM (vergleiche Kapitel 12.3). Der Vorteil vom QS ist, dass die Laufzeit kaum von der Größe der Primteiler p von a abhängt, daher ist QS besonders effizient, wenn die Zahl a keine kleinen Primteiler besitzt.

17 Das Zahlkörpersieb

Im Jahr 1993 veröffentlichten Arjen K. Lenstra und Hendrik W. Lenstra Jr. in [38] das Zahlkörpersieb (englisch: NFS=Number Field Sieve), das bis heute heuristisch schnellste bekannte Faktorisierungsverfahren. Beim allgemeinen Zahlkörpersieb (englisch: GNFS=General Number Field Sieve) verwenden wir wieder die Vorgehensweise der Faktorisierung von Fermat und von Legendre. Wir suchen zwei Quadrate x^2 und y^2 mit

$$x^2 \equiv y^2 \bmod a \tag{17.1}$$

und können dadurch mit großer Wahrscheinlichkeit (siehe Kapitel 14) die echten Teiler $p = \gcd(x \pm y, a)$ berechnen. Die beim QS in Anspruch genommene Hilfe des Polynoms $f : \mathbb{Z} \to \mathbb{Z}/a\mathbb{Z}$, $f(b) = b^2 - a$ zum Erzeugen glatter Zahlen $f(b)$ wollen wir beim Zahlkörpersieb an zwei Stellen verallgemeinern. Zum einen erweitern wir das Polynom, das beim QS auf die spezielle Form zweiten Grades fixiert ist, auf beliebige Polynome und zum anderen verändern wir die Grundmenge $\mathbb{Z}$ von f auf Zahlkörper aus Kapitel 3. Dadurch verbessern wir den Bestand der zu schnell größer werdenden Werte $f(b)$, die im QS quadratisch zum Eingabewert b anwachsen, wodurch die Wahrscheinlichkeit rasch sinkt, dass $f(b)$ glatt ist.
Des Weiteren geben wir beim QS von vornherein eines der Quadrate für Kongruenz (17.1) vor und berechnen nur ein zweites passendes Quadrat. Auch diese Herangehensweise ändern wir ab, sodass wir sowohl für die rechte, als auch für die linke Seite Quadrate suchen – auf den ersten Blick eine Verschlechterung der Vorgehensweise.

17.1 Der Algorithmus (Allgemeines Zahlkörpersieb)

Sei $a \in \mathbb{N}$ eine zusammengesetzte, ungerade natürliche Zahl, die sich nicht als Potenz darstellen lässt, R ein Ring und

$$\psi : R \to \mathbb{Z}/a\mathbb{Z}$$

ein vorerst beliebiger Ringhomomorphismus. Wir sehen später, wie wir die Parameter ψ und R genau wählen. Weiters sei $\gamma \in R$ so, dass $\gamma = \mu^2$ ein Quadrat in R und $\psi(\gamma) = y^2$ ein Quadrat in $\mathbb{Z}/a\mathbb{Z}$ ist. Daraus folgt

$$y^2 \equiv \psi(\gamma) \equiv \psi(\mu^2) \equiv \psi(\mu)^2 \equiv x^2 \bmod a$$

aus der Homomorphieeigenschaft von ψ für $x := \psi(\mu)$. Mit der Lösung von Gleichung (17.1) berechnen wir mittels $p = \gcd(x \pm y, a)$ einen Teiler von a.

17.1.1 Die Wahl der Abbildungen f und ψ

Für die Konstruktion der Abbildung ψ wählen wir ein irreduzibles, normiertes Polynom $f \in \mathbb{Z}[x]$ vom Grad $d \geq 1$. Wie wir den Grad d und die ganzzahligen Koeffizienten von f bestimmen, sehen wir gleich. Weiters sei $\alpha \in \mathbb{C}$ eine Nullstelle von f, die es nach Satz 1.0.7 geben muss. Nun betrachten wir den Ring $R = \mathbb{Z}[\alpha]$, der nach Definition 3.0.7 die Basis $\{1, \alpha, \ldots, \alpha^{d-1}\}$ besitzt und somit können wir alle Elemente $r \in \mathbb{Z}[\alpha]$ als

$$r = \sum_{i=0}^{d-1} r_i \alpha^i$$

mit $r_i \in \mathbb{Z}$ schreiben. Weiters sei $m \in \mathbb{Z}$ so gewählt, dass $f(m) \equiv 0 \bmod a$ gilt. Auch hier sehen wir noch, wie wir m wählen. Nun definieren wir den Ringhomomorphismus $\psi : \mathbb{Z}[\alpha] \to \mathbb{Z}/a\mathbb{Z}$ durch

$$r = \sum_{i=0}^{d-1} r_i \alpha^i \mapsto \sum_{i=0}^{d-1} r_i m^i \bmod a,$$

insbesondere gilt $\psi(\alpha) = m$ und $\psi(1) = 1$.
Für die oben angesprochenen, noch nicht festgelegten Parameter wählen wir zuerst den Grad d von f – je nach Größe von a und experimenteller Erfahrungen – folgendermaßen: Hat a eine Größe von unter 50 Dezimalstellen, so setzen wir $d = 2$, bei 50 bis 80 Dezimalstellen setzen wir $d = 3$, bei 80 bis 110 Stellen setzen wir $d = 4$ und bei 110 bis 130 Stellen setzen wir $d = 5$. Weiters setzen wir $m = \lfloor \sqrt[d]{a} \rfloor$ und entwickeln die Zahl

$$a = m^d + c_{d-1} m^{d-1} + \cdots + c_1 m + c_0$$

zur Basis m mit $c_i \in \mathbb{Z}/m\mathbb{Z}$. Dass der führende Koeffizient c_d in dieser Entwicklung tatsächlich gleich 1 ist, zeigt folgendes Lemma.

Lemma 17.1.1. *Seien $a, d, m \in \mathbb{N}$ mit $a > (2d)^d$ und $m = \lfloor \sqrt[d]{a} \rfloor$, dann gilt $a < 2m^d$.*

Durch die Entwicklung von a definieren wir das Polynom

$$f(z) = z^d + c_{d-1} z^{d-1} + \cdots + c_1 z + c_0,$$

das normiert, aber nicht zwingend irreduzibel ist. Doch wäre $f(z) = g(z) \cdot h(z)$ ein nichttriviales Produkt, so hätten wir die Faktorisierung $a = f(m) = g(m) \cdot h(m)$ gefunden. Sei also f o.B.d.A. irreduzibel.

17.1.2 Norm und Glattheit in $\mathbb{Z}$ und $\mathbb{Z}[\alpha]$

Alle Parameter für das Zahlkörpersieb sind bestimmt, sodass wir als nächsten Schritt „nur“ ein $\gamma \in \mathbb{Z}[\alpha]$ finden müssen, sodass γ und $\psi(\gamma)$ Quadrate in $\mathbb{Z}[\alpha]$ bzw. $\mathbb{Z}/a\mathbb{Z}$ bilden. Dazu wählen wir eine Menge S bestehend aus Paaren $(k, l) \in \mathbb{Z} \times \mathbb{Z}$ mit $\gcd(k, l) = 1$ und bilden die Elemente $k - l\alpha \in \mathbb{Z}[\alpha]$ und $k - lm \in \mathbb{Z}/a\mathbb{Z}$, die

$$\psi(k - l\alpha) = k - l\psi(\alpha) = k - lm \quad \forall (k, l) \in S$$

erfüllen. Des Weiteren wählen wir eine geeignete Teilmenge $S_0 \subseteq S$, sodass die Produkte

$$\prod_{(k,l)\in S_0} (k - l\alpha) = \mu^2 \quad \text{und} \quad \prod_{(k,l)\in S_0} (k - lm) = y^2 \tag{17.2}$$

Quadrate in den jeweiligen Ringen bilden. Aufgrund der Homomorphieeigenschaft von ψ folgt

$$x^2 \equiv \psi(\mu)^2 \equiv \psi\left(\mu^2\right) \equiv \psi\left(\prod_{(k,l)\in S_0} (k - l\alpha)\right) \equiv \prod_{(k,l)\in S_0} \psi(k - l\alpha) \equiv \prod_{(k,l)\in S_0} (k - lm) \equiv y^2 \bmod a.$$

Die Quadrate in (17.2) erzeugen wir, indem wir, ähnlich zur Menge S beim QS, nur Paare (k, l) in S zulassen, sodass $k - l\alpha$ bzw. $k - lm$ glatt sind, um daraus eine passende Teilmenge S_0 zu wählen. Für $k - lm \in \mathbb{Z}/a\mathbb{Z}$ kennen wir nach Definition 11.0.1 die Bedeutung von Glattheit, für Elemente $k - l\alpha \in \mathbb{Z}[\alpha]$ benötigen wir noch eine geeignete Definition (siehe Definition 17.1.4). Betrachten wir vorerst die Glattheit im Ring $\mathbb{Z}/a\mathbb{Z}$ und wählen eine Faktorbasis F_{p_r} (Definition 14.0.2) und eine geeignete Schranke M, sodass $0 < |k|, l \leq M$ gilt und definieren die Menge

$$S = \{(k, l) \in \mathbb{Z} \times \mathbb{Z} \mid 0 < |k|, l \leq M \text{ und } \gcd(k, l) = 1\}.$$

Aus S verwerfen wir ähnlich dem Siebverfahren aus Kapitel 16 (die Ausführung folgt in Kapitel 17.1.5) die Paare (k,l), sodass $k-lm$ nicht F_{p_r}-glatt sind. Aus den übrig gebliebenen Paaren bilden wir das oben angesprochene Quadrat $y^2 = \prod_{(k,l)\in S_0}(k-lm) \in \mathbb{Z}/a\mathbb{Z}$ mit einer geeigneten Menge $S_0 \subseteq S$. Als Kurzschreibweise definieren wir das homogene Polynom

$$G(k,l) = k - lm.$$

Zu beachten ist, dass wir gleichzeitig mit der selben Teilmenge S_0 ein Quadrat $\mu^2 = \prod_{(k,l)\in S_0}(k - l\alpha) \in \mathbb{Z}[\alpha]$ bilden müssen. Dazu definieren wir den Begriff der Norm, der Glattheit und der Faktorbasis in $\mathbb{Z}[\alpha] \subseteq \mathbb{Q}(\alpha)$.

Definition 17.1.2. Sei $f \in \mathbb{Q}[x]$ ein irreduzibles, normiertes Polynom vom Grad d mit Nullstelle $\alpha = \alpha_1 \in \mathbb{C}$. Weiters seien $\alpha_2, \ldots, \alpha_d \in \mathbb{C}$ die konjugierten Nullstellen und $r = \sum_{i=0}^{d-1} r_i\alpha^i \in \mathbb{Q}(\alpha)$. Die **algebraische Norm** $\mathcal{N} : \mathbb{Q}(\alpha) \to \mathbb{Q}$ definieren wir durch

$$\mathcal{N}(r) = \prod_{j=1}^{d}\left(\sum_{i=0}^{d-1} r_i\alpha_j^i\right).$$

Satz 17.1.3. *(a) Die Norm $\mathcal{N}$ bildet eine multiplikative Funktion, das heißt für $r,s \in \mathbb{Q}(\alpha)$ gilt $\mathcal{N}(rs) = \mathcal{N}(r)\mathcal{N}(s)$.*
(b) Für $r \in \mathbb{Z}[\alpha]$ gilt $\mathcal{N}(r) \in \mathbb{Z}$.

Sei $f(x) = (x-\alpha_1)\cdots(x-\alpha_d) = x^d + c_{d-1}x^{d-1} + \cdots + c_1x + c_0$ das in Kapitel 17.1.1 definierte Polynom. Für $r = k - l\alpha \in \mathbb{Z}[\alpha]$ berechnen wir die Norm durch

$$\mathcal{N}(k-l\alpha) = (k-l\alpha_1)(k-l\alpha_2)\cdots(k-l\alpha_d) = l^d(\frac{k}{l}-\alpha_1)\cdots(\frac{k}{l}-\alpha_d) = l^d f(\frac{k}{l}).$$

Für das homogenisierte Polynom $F(x,y) = x^d + c_{d-1}x^{d-1}y + \cdots + c_1xy^{d-1} + c_0y^d$ von $f(x)$ gilt

$$F(k,l) = l^d f(\frac{k}{l}) = \mathcal{N}(k-l\alpha).$$

Folgende vorläufige Definition einer algebraischen Faktorbasis und von algebraischer Glattheit im Ring $\mathbb{Z}[\alpha]$ adaptieren wir später zwei Mal (siehe Definitionen 17.1.11 und 17.1.17).

Definition 17.1.4. Sei $\mathcal{F}'_{p_r} = \{-1, p_1, p_2, \ldots, p_r\}$ eine Faktorbasis, die wir in diesem Zusammenhang als **algebraische Faktorbasis** bezeichnen. Eine algebraische Zahl $\gamma \in \mathbb{Z}[\alpha]$ heißt **$\mathcal{F}'_{p_r}$-glatt**, falls deren Norm $\mathcal{N}(\gamma)$ $\mathcal{F}'_{p_r}$-glatt ist, also sich als $\mathcal{N}(\gamma) = \pm\prod_{i=1}^{r} p_i^{e_i}$ schreiben lässt.

Lemma 17.1.5. *Sei $\gamma \in \mathbb{Z}[\alpha]$ ein Quadrat, dann ist $\mathcal{N}(\gamma)$ ein Quadrat in $\mathbb{Z}$.*

Bemerkung 17.1.6. *Die Umkehrung von Lemma 17.1.5 ist im Allgemeinen falsch. Ist $\mathcal{N}(\gamma) \in \mathbb{Z}$ ein Quadrat, so ist $\gamma \in \mathbb{Z}[\alpha]$ nicht zwingend ein Quadrat. Dies zeigt folgendes Beispiel:*

Beispiel 17.1.7. Sei $f(x) = x^2 + 1$, $\alpha = \alpha_1 = i, \alpha_2 = -i \in \mathbb{C}$ und $\mathcal{N}(k - li) = F(k,l) = (k+li)(k-li) = k^2 + l^2$. Das Element $5i \in \mathbb{Z}[i]$ ist kein Quadrat in $\mathbb{Z}[i]$ und lässt sich in die primen Elemente $5i = (2+i)(1+2i)$ zerlegen, und es gilt

$$F(0,5) = 25,\ F(2,1) = 5 \text{ und } F(1,2) = 5.$$

Das heißt, $5i$ hat als Norm ein Quadrat und ist selber keines, weil die zwei verschiedenen Primteiler von $5i$ die selbe Norm haben.

Haben wir also ein Quadrat $\mathcal{N}(\gamma) = \prod_{(k,l)\in S_0} F(k,l)$ gefunden, so können wir daraus nicht schließen, dass auch γ ein Quadrat ist. Dies ist aber eine Voraussetzung für das Zahlkörpersieb. Wir wollen die Lösung der Umkehrung von Lemma 17.1.5 einstweilen beiseite lassen und kommen erst in Kapitel 17.1.6 wieder darauf zurück. Zusätzlich haben wir das Problem, dass der Ring $\mathbb{Z}[\alpha]$ im Gegensatz zu $\mathbb{Z}[i]$ im Allgemeinen nicht faktoriell ist (siehe Bemerkung 3.0.10) und es somit keine eindeutige Faktorisierung gibt. Außerdem wollen wir das Auftreten von Einheiten in der Faktorisierung eines Elements $\gamma \in \mathbb{Z}[\alpha]$ vermeiden.

17.1.3 Glattheit in $\mathbb{Z}[\alpha]$ durch Ideale

In den folgenden Kapiteln definieren wir eine algebraische Faktorbasis $\mathcal{F}'$ bestehend aus Primidealen ersten Grades $\mathcal{P}$ (Definition 17.1.10) und zeigen, dass die von $k - l\alpha$ erzeugten Hauptideale

$$\langle k - l\alpha \rangle = \prod_{\mathcal{P}\in\mathcal{F}'} \mathcal{P}$$

in ein eindeutiges Produkt von Elementen der Faktorbasis zerfallen. Diese Zerlegung in Primideale ersten Grades ist im Gegensatz zur Zerlegung in $\mathbb{Z}[\alpha]$ im Wesentlichen eindeutig und frei von Einheiten. Wie wir gleich sehen, entspricht ein Primideal ersten Grades genau einem Paar (q, s), wobei q eine Primzahl und s ein Element der Menge

$$\mathcal{F}(q) := \{s \in \mathbb{Z}/q\mathbb{Z} \mid f(s) \equiv 0 \bmod q\}$$

ist. Die Menge dieser Paare bezeichnen wir mit

$$\mathcal{F} := \{(q, s) \mid q \in \mathbb{P} \text{ und } s \in \mathcal{F}(q)\}.$$

Wir verwenden nicht zufällig die Notation „$\mathcal{F}$“ für diese Menge, da dies unsere Faktorbasis wird, aber dazu später mehr.
Die von $k - l\alpha$ erzeugten Hauptideale $\langle k - l\alpha\rangle$ zerfallen nach Korollar 3.1.11 in ein Produkt von Primidealen $\mathcal{P}_i$ aus $\mathcal{O}_\alpha$, da $\mathcal{O}_\alpha$ ein Dedekind Ring ist. Als nächsten Schritt spezifizieren wir die in Definition 3.1.6 definierte Norm für Ideale auf Haupt- und Primideale.

Satz 17.1.8. *Sei $\gamma \in \mathcal{O}_\alpha$ und $\langle\gamma\rangle$ das von γ erzeugte Hauptideal, dann gilt*

$$\mathcal{N}(\langle\gamma\rangle) = |\mathcal{N}(\gamma)|.$$

Theorem 17.1.9. *Sei $\mathcal{P}$ ein echtes Ideal in $\mathcal{O}_\alpha$. Gilt*

$$\mathcal{N}(\mathcal{P}) = p$$

für eine Primzahl p, so ist $\mathcal{P}$ ein Primideal. Ist umgekehrt $\mathcal{P}$ ein Primideal, so gilt

$$\mathcal{N}(\mathcal{P}) = p^e$$

für eine Primzahl p und eine natürliche Zahl $e > 0$.

Beweis. Siehe [22], Seite 33. □

Die Norm für die Elemente $k - l\alpha$ bzw. die Hauptideale $\langle k - l\alpha\rangle$ lässt sich nach den vorangehenden Sätzen wie folgt berechnen:

$$|\mathcal{N}(k - l\alpha)| \overset{Satz}{\underset{17.1.8}{=}} \mathcal{N}\left(\langle k - l\alpha\rangle\right) \overset{Kor.}{\underset{3.1.11}{=}} \mathcal{N}\left(\prod_i \mathcal{P}_i^{f_i}\right) \overset{Satz}{\underset{3.1.8}{=}} \prod_i \mathcal{N}(\mathcal{P}_i)^{f_i} \overset{Thm.}{\underset{17.1.9}{=}} \prod_i p_i^{e_i f_i}. \tag{17.3}$$

Wir überarbeiten mit den erhaltenen Ergebnissen die Definition der algebraischen Faktorbasis und der Glattheit von Definition 17.1.4 für algebraische Zahlen.

Definition 17.1.10. Eine **algebraische Faktorbasis** $\mathcal{F}'$ ist eine Menge von Primidealen in $\mathcal{O}_\alpha$.

Definition 17.1.11. Sei $\mathcal{F}'$ eine algebraische Faktorbasis. Eine algebraische Zahl $\gamma \in \mathbb{Z}[\alpha]$ heißt **$\mathcal{F}'$-glatt**, falls sich das Hauptideal $\langle\gamma\rangle$ als Produkt von Primidealen aus $\mathcal{F}'$ schreiben lässt.

17.1.4 Primideale ersten Grades

In diesem Kapitel führen wir Primideale ersten Grades ein und zeigen, dass nur diese bei der Zerlegung der Hauptideale $\langle k - l\alpha\rangle$ auftreten. Weiters zeigen wir, dass jedes Primideal ersten Grades in $\mathbb{Z}[\alpha]$ genau einem Paar $(q, s) \in \mathcal{F}$ entspricht.

Definition 17.1.12. Ein Primideal $\mathcal{P}$ in $\mathcal{O}_\alpha$ heißt **Primideal ersten Grades**, falls

$$\mathcal{N}(\mathcal{P}) = p$$

für eine Primzahl p gilt.

Zur Erinnerung haben wir die Mengen

$$\mathcal{F}(q) = \{s \in \mathbb{Z}/q\mathbb{Z} \mid f(s) \equiv 0 \bmod q\} \quad \text{und} \quad \mathcal{F} = \{(q, s) \mid q \in \mathbb{P} \text{ und } s \in \mathcal{F}(q)\}$$

bereits definiert. Mit folgendem Theorem stellen wir die zentrale Verbindung zwischen der Menge der Primideale ersten Grades in $\mathbb{Z}[\alpha]$ und der Menge $\mathcal{F}$ her.

Theorem 17.1.13. *Sei $f \in \mathbb{Z}[x]$ ein irreduzibles, normiertes Polynom und $\alpha \in \mathbb{C}$ eine Nullstelle von f. Dann gibt es eine bijektive Korrespondenz zwischen der Menge der Primideale ersten Grades in $\mathbb{Z}[\alpha]$ und der Menge $\mathcal{F}$, das heißt, jedes Primideal ersten Grades entspricht genau einem Paar (q, s), wobei $q \in \mathbb{P}$ und $s \in \mathbb{Z}/q\mathbb{Z}$ mit $f(s) \equiv 0 \bmod q$ gilt.*

Beweis. Siehe [22], Seite 34. □

In Theorem 17.1.15 zeigen wir, dass in der Zerlegung von $\langle k - l\alpha\rangle$ tatsächlich nur Primideale ersten Grades auftreten. Zuvor betrachten wir nochmals die Exponenten f_i aus (17.3), die angeben, wie oft das Primideal $\mathcal{P}_i$ in der Zerlegung von $\langle k - l\alpha\rangle$ vorkommt. Fassen wir diese Exponenten als Gruppenhomomorphismen

$$f_{\mathcal{P}_i} : \mathbb{Q}(\alpha)^* \to \mathbb{Z}$$

auf, wo jedem Element $\gamma \in \mathbb{Q}(\alpha)^*$ die Häufigkeit des Primideals $\mathcal{P}_i$ in der Zerlegung von $\langle\gamma\rangle$ zugeordnet wird, so gilt folgendes Lemma:

Lemma 17.1.14. *Sei $\mathcal{P}$ ein Primideal in $\mathbb{Z}[\alpha]$, dann gilt für den eben definierten Gruppenhomomorphismus $f_{\mathcal{P}}$:*
(a) $f_{\mathcal{P}}(\gamma) \geq 0 \ \forall \gamma \in \mathbb{Q}(\alpha)^*$
(b) $f_{\mathcal{P}}(\gamma) > 0 \iff \mathcal{P} \mid \langle\gamma\rangle$
(c) $|\mathcal{N}(\gamma)| = \prod_i \mathcal{N}(\mathcal{P}_i)^{f_{\mathcal{P}_i}}$ für Primideale $\mathcal{P}_i$.

Theorem 17.1.15. *Seien $k, l \in \mathbb{Z}$ mit $\gcd(k, l) = 1$, $\gamma = k - l\alpha \in \mathbb{Z}[\alpha]$ und $\mathcal{P}$ ein Primideal in $\mathbb{Z}[\alpha]$. Dann gilt:*
(a) Ist $\mathcal{P}$ kein Primideal ersten Grades, so gilt $f_{\mathcal{P}}(k - l\alpha) = 0$.
(b) Ist $\mathcal{P}$ ein Primideal ersten Grades, das mit einem eindeutigem Paar $(q, s) \in \mathcal{F}$ nach Theorem 17.1.13 korrespondiert, so gilt

$$f_{\mathcal{P}}(k - l\alpha) = \begin{cases} e, & \text{falls } k \equiv ls \bmod q \\ 0, & \text{sonst} \end{cases}$$

wobei $q^e \mid \mathcal{N}(k - l\alpha)$ und $q^{e+1} \nmid \mathcal{N}(k - l\alpha)$ gilt.

Beweis. Siehe [22], Seite 36. □

Es treten folglich nur Primideale ersten Grades aus $\mathbb{Z}[\alpha]$ in der Zerlegung von $\langle k-l\alpha\rangle$ auf. Außerdem besagt Theorem 17.1.15 (b), wie häufig ein Primideal, das mit einem Paar (q,s) korrespondiert, auftritt, nämlich in der größtmöglichen Potenz von q in der Primfaktorzerlegung von $\mathcal{N}(k-l\alpha)$. Damit definieren wir das letzte Mal die algebraische Faktorbasis und die algebraische Glattheit (vergleiche Definitionen 17.1.10 und 17.1.11).

Definition 17.1.16. Für eine Primzahl p_r definieren eine **algebraische Faktorbasis** $\mathcal{F}_{p_r}$ durch

$$\mathcal{F}_{p_r} = \{(q,s) \mid q \in \mathbb{P},\ q \leq p_r \text{ und } s \in \mathcal{F}(q)\}.$$

Diese Paare entsprechen nach Theorem 17.1.13 Primidealen ersten Grades aus $\mathbb{Z}[\alpha]$.

Definition 17.1.17. Sei $\mathcal{F}_{p_r}$ eine algebraische Faktorbasis. Eine algebraische Zahl $\gamma \in \mathbb{Z}[\alpha]$ heißt **$\mathcal{F}_{p_r}$-glatt**, falls die Norm $\mathcal{N}(\gamma)$ als Produkt von Primzahlen $q \leq p_r$ geschrieben werden kann.

Im Endeffekt haben wir eine sehr ähnliche Definition von Glattheit algebraischer Zahlen wie anfangs in Definition 17.1.4. Der einzige Unterschied besteht darin, dass wir die zusätzliche Information besitzen, bei welchem Element $s \in \mathcal{F}(q)$ die Primzahl q die Norm $\mathcal{N}(\gamma)$ teilt. In die Idealtheorie übersetzt bedeutet das, dass wir wissen, welches Primideal (q,s) das Hauptideal $\langle\gamma\rangle$ teilt.

Zusammenfassung. Um eine natürliche Zahl a zu faktorisieren, wählen wir nach Kapitel 17.1.1 ein geeignetes Polynom $f \in \mathbb{Z}[x]$ mit Nullstelle $\alpha \in \mathbb{C}$ und ein $m \in \mathbb{N}$ mit $a \mid f(m)$. Dann wählen wir eine Menge S bestehend aus Paaren $(k,l) \in \mathbb{Z}\times\mathbb{Z}$ mit $\gcd(k,l) = 1$, sodass die Elemente $k - lm \in \mathbb{Z}$ und $k - l\alpha \in \mathbb{Z}[\alpha]$ bezüglich einer rationalen Faktorbasis F_{p_r} bzw. einer algebraischen Faktorbasis $\mathcal{F}_{p_r}$ glatt sind. Die Glattheit der Elemente stellen wir mittels eines Siebverfahrens fest, das wir in Kapitel 17.1.5 erläutern werden. Haben wir genügend Paare (k,l) gefunden, wählen wir eine Teilmenge $S_0 \subseteq S$, sodass die Produkte

$$\prod_{(k,l)\in S_0} G(k,l) \quad \text{und} \quad \prod_{(k,l)\in S_0} F(k,l)$$

jeweils ein Quadrat bilden. Wir können aber noch nicht aus dem algebraischen Norm-Produkt schließen, dass

$$\gamma = \prod_{(k,l)\in S_0} (k - l\alpha)$$

ein Quadrat in $\mathbb{Z}[\alpha]$ ist. Selbst wenn $\gamma = \mu^2$ ein Quadrat ist, können wir daraus nicht schließen, dass $\mu \in \mathbb{Z}[\alpha]$ gilt. Bevor wir diese beiden Probleme in Kapitel 17.1.6 behandeln, stellen wir die Siebverfahren für rationale und algebraische Zahlen vor.

17.1.5 Rationales und Algebraisches Sieb

Wir betrachten zuerst den einfacheren Fall des rationalen Siebs, wo wir sehr ähnlich zum QS (siehe Kapitel 16.1) verfahren. Gegeben sind Elemente der Form $k - lm \in \mathbb{Z}$ mit $(k,l) \in S = \{(k,l) \in \mathbb{Z}\times\mathbb{Z} \mid 0 < |k|, l \leq M \text{ und } \gcd(k,l) = 1\}$ und wir wollen diejenigen Paare (k,l) heraussieben, die für eine Faktorbasis $F_{p_r} = \{-1, p_1, \ldots, p_r\}$ glatt sind. Die Schranke $M \in \mathbb{N}$ wählen wir in der Größenordnung

$$M \approx L_a[\tfrac{1}{3}; \left(\tfrac{8}{9}\right)^{\frac{1}{3}}] = \exp\left(\left(\tfrac{8}{9}\right)^{\frac{1}{3}} n_a^{\frac{1}{3}} (\log(n_a))^{\frac{2}{3}}\right)$$

und die Primzahl p_r in der gleichen Größenordnung. Für Details zur Herleitung der Größenordnungen von M und p_r siehe Kapitel 17.3. Wir fixieren nun ein $l_1 \in \mathbb{N}$ mit $0 < l_1 \leq M$ und schreiben in die erste Spalte einer Matrix den Vektor $G(k, l_1)_{(0<|k|\leq M)}$. Als erstes ändern wir bei alle negativen Einträgen der Spalte das Vorzeichen, dann berechnen wir für die erste Primzahl $p_1 \in F_{p_r}$ eine Lösung k_0 der Kongruenz

$$k_0 \equiv l_1 m \bmod p_1.$$

Nun dividieren wir alle $k - l_1 m$ mit $k \equiv k_0 \bmod p_1$ durch die höchstmögliche Potenz von p_1. Dieses Prozedere führen wir mit allen Primzahlen aus F_{p_r} durch und gehen zur nächsten natürlichen Zahl $l_2 \in]0, M]$ über, definieren die zweite Spalte $G(k, l_2)_{(0<|k|\leq M)}$ der Matrix und führen das eben erklärte Siebverfahren durch, bis wir alle Zahlen $0 < l_i \leq M$ abgearbeitet haben. Der Matrixeintrag an der Stelle (k, l) ist genau dann F_{p_r}-glatt, wenn er schlussendlich den Wert „1“ hat. Notieren wir bei jeder Division die Primzahlpotenz, durch die wir dividiert haben, so bekommen wir die Primfaktorzerlegung der glatten Elemente mitgeliefert.
Das algebraische Sieb funktioniert auf eine ähnliche Weise. Für die selben Parameter M und p_r definieren wir die algebraische Faktorbasis $\mathcal{F}_{p_r} = \{(q, s) \mid q \leq p_r \text{ Primzahl und } s \in \mathcal{F}(q)\}$. Für die Siebbedingung gilt nach Theorem 17.1.15 folgendes Korollar.

Korollar 17.1.18. *Seien $k, l \in \mathbb{Z}$ teilerfremd und $q \in \mathbb{P}$, dann gilt*

$$q \mid F(k, l) \iff k \equiv ls \bmod q \text{ für ein } s \in \mathcal{F}(q).$$

Wir fixieren wieder eine natürliche Zahl l_1 mit $0 < l_1 \leq M$ und definieren die erste Spalte einer Matrix durch den Vektor $F(k, l_1)_{(0<|k|\leq M)}$. Bei allen negativen Werten $F(k, l_1)$ ändern wir das Vorzeichen und bestimmen für alle Primzahlen $q \leq p_r$ die Menge

$$\mathcal{F}(q) = \{s \in \mathbb{Z}/q\mathbb{Z} \mid f(s) \equiv 0 \bmod q\}.$$

Für die erste Primzahl q und für alle $s \in \mathcal{F}(q)$ berechnen wir die Lösungen k_s der Kongruenzen

$$k_s \equiv l_1 s \bmod q.$$

Weiters dividieren wir alle $F(k, l_1)$ mit $k \equiv k_s \bmod q$ für ein s durch die größtmögliche Potenz von q. Diesen Schritt wiederholen wir mit allen Primzahlen $q \leq p_r$ und gehen zur nächsten natürlichen Zahl $l_2 \in]0, M]$ über, die die zweite Spalte $F(k, l_2)_{(0<|k|\leq M)}$ definiert, und führen das eben erklärte Siebverfahren durch. Haben wir alle Werte $0 < l_i \leq M$ abgearbeitet, so sind genau die Elemente $k - l\alpha$ $\mathcal{F}_{p_r}$-glatt, deren entsprechenden Matrixeinträge letztendlich gleich „1“ sind. Merken wir uns bei jeder Division die Primzahlpotenz und auch bei welchem Wert $s \in \mathcal{F}(q)$ wir dividiert haben, so erhalten wir einerseits die Primfaktorzerlegung der Normen $F(k, l)$ und andererseits die Zerlegungen der Hauptideale $\langle k - l\alpha \rangle$.

17.1.6 Quadrate in $\mathbb{Z}[\alpha]$

Folgendes Lemma garantiert, dass $\mu \in \mathbb{Z}[\alpha]$ gilt – im Allgemeinen gilt $\mu \in \mathcal{O}_\alpha$–, wenn wir ein Quadrat $\gamma = \mu^2 \in \mathbb{Z}[\alpha]$ gegeben haben.

Lemma 17.1.19. *Sei $f(x) \in \mathbb{Z}[x]$ irreduzibel und normiert, $\alpha \in \mathbb{C}$ eine Nullstelle von f und $\mu \in \mathcal{O}_\alpha$, dann gilt $f'(\alpha) \cdot \mu \in \mathbb{Z}[\alpha]$.*

Setzen wir nach Lemma 17.1.19

$$\mu = f'(\alpha) \cdot \mu_0 \in \mathbb{Z}[\alpha]$$

für $\mu_0^2 = \prod_{(k,l)\in S_0}(k - l\alpha) \in \mathbb{Z}[\alpha]$, wobei $\mu_0 \in \mathcal{O}_\alpha$ gilt, so liegt das Quadrat

$$\mu^2 = f'(\alpha)^2 \prod_{(k,l)\in S_0} (k - l\alpha)$$

in $\mathbb{Z}[\alpha]$. Setzen wir entsprechend $x = \psi(\mu) = f'(m)\psi(\mu_0)$ und $y = f'(m)\sqrt{\prod_{(k,l)\in S_0}(k - lm)}$, so gilt

$$x^2 \equiv \psi(\mu^2) \equiv \psi(f'(\alpha)^2\mu_0^2) \equiv f'(m)^2 \prod_{(k,l)\in S_0} \psi(k - l\alpha) \equiv f'(m)^2 \prod_{(k,l)\in S_0} (k - lm) \equiv y^2 \bmod a.$$

Bevor wir eine hinreichende Bedingung herleiten, wann ein Element $\gamma \in \mathbb{Z}[\alpha]$ mit quadratischer Norm selbst ein Quadrat ist, lernen wir eine weitere notwendige Bedingung kennen. Wir definieren den Vektor

$$v(k - l\alpha) = \big(v_{(q,s)}(k - l\alpha)\big)_{(q,s)\in\mathcal{F}}$$

mit den Einträgen $v_{(q,s)}(k - l\alpha) := f_{\mathcal{P}}(k - l\alpha)$ nach Theorem 17.1.15, wobei $\mathcal{P}$ das mit dem Paar $(q, s) \in \mathcal{F}$ korrespondierende Primideal ersten Grades ist. Gilt nach Korollar 17.1.18 $q \mid k - ls$ für ein $s \in \mathcal{F}(q)$, so setzen wir $v_{(q,s)}(k - l\alpha)$ gleich dem größten Exponenten e, sodass q^e noch $\mathcal{N}(k - l\alpha)$ teilt, q^{e+1} aber nicht mehr. Gibt es so ein s aus $\mathcal{F}(q)$ nicht, ist der Eintrag gleich Null.

Satz 17.1.20. *Sei $\mathcal{F}_{p_r}$ eine algebraische Faktorbasis und S_0 eine Menge von teilerfremden Paaren (k, l), sodass die Elemente $k - l\alpha \in \mathbb{Z}[\alpha]$ $\mathcal{F}_{p_r}$-glatt sind. Bildet weiters*

$$\prod_{(k,l)\in S_0} (k - l\alpha)$$

ein Quadrat, so gilt für die eben definierten Vektoren

$$\sum_{(k,l)\in S_0} v(k - l\alpha) \equiv \vec{0} \bmod 2.$$

Beweis. Siehe [23], Seite 284. □

Für die Umkehrung von Lemma 17.1.5 und Satz 17.1.20 betrachten wir folgendes einfaches Beispiel im Ring $\mathbb{Z}$.

Beispiel 17.1.21. Die Zahl $b = 9$ ist ein Quadrat in $\mathbb{Z}$ und es gilt

$$\left(\frac{9}{\tilde{q}}\right) = 1$$

für alle Primzahlen $\tilde{q}$, die b nicht teilen. Die Zahl $b = 7$ ist kein Quadrat in $\mathbb{Z}$ und es gibt Primzahlen (z.B. $\tilde{q} = 3$), sodass $\left(\frac{7}{\tilde{q}}\right) = 1$ gilt und Primzahlen (z.B. $\tilde{q} = 5$), sodass $\left(\frac{7}{\tilde{q}}\right) = -1$ gilt.

Berechnen wir für viele Primzahlen $\tilde{q}$ das Legendre-Symbol $\left(\frac{b}{\tilde{q}}\right)$ für ein $b \in \mathbb{Z}$, so ist b mit großer Wahrscheinlichkeit ein Quadrat, wenn $\left(\frac{b}{\tilde{q}}\right) = 1$ für alle $\tilde{q}$ gilt. Diesen probabilistischen Ansatz wollen wir für die Umkehrung von Lemma 17.1.5 bzw. Satz 17.1.20 verwenden.

Lemma 17.1.22. *Sei $f \in \mathbb{Z}[x]$ ein irreduzibles, normiertes Polynom und $\alpha \in \mathbb{C}$ eine Nullstelle von f. Sei weiters $\tilde{q} \in \mathbb{P}$ und $\tilde{s} \in \mathbb{Z}$ so, dass $\tilde{q} \mid f(\tilde{s})$ und $\tilde{q} \nmid f'(\tilde{s})$ gilt. Sei $S_0 = \{(k, l) \in \mathbb{Z} \times \mathbb{Z} \mid \gcd(k, l) = 1\}$ so, dass $\tilde{q} \nmid k - l\tilde{s}$ $\forall (k, l) \in S_0$ gilt und $f'(\alpha)^2 \prod_{(k,l)\in S_0}(k - l\alpha)$ ein Quadrat ist, dann folgt*

$$\prod_{(k,l)\in S_0} \left(\frac{k - l\tilde{s}}{\tilde{q}}\right) = 1.$$

Lemma 17.1.22 ist eine weitere notwendige Bedingung, damit das Produkt $f'(\alpha)^2 \prod_{(k,l)\in S_0}(k-l\alpha)$ ein Quadrat ist. Doch haben wir genügend viele Paare $(\tilde{q},\tilde{s})$ gefunden, sodass $\tilde{q} \mid f(\tilde{s})$ und $\tilde{q} \nmid f'(\tilde{s})$ $\forall(\tilde{q},\tilde{s})$ gilt und keines der $\tilde{q}$ die Norm $\mathcal{N}(k-l\alpha)$ für $(k,l) \in S_0$ teilt, dann folgt mit großer Wahrscheinlichkeit aus

$$\prod_{(k,l)\in S_0} \left(\frac{k-l\tilde{s}}{\tilde{q}}\right) = 1 \ \forall(\tilde{q},\tilde{s}) \quad \text{und} \quad \sum_{(k,l)\in S_0} v(k-l\alpha) \equiv \vec{0} \bmod 2, \tag{17.4}$$

dass

$$\prod_{(k,l)\in S_0} (k-l\alpha) = \mu^2$$

ein Quadrat ist. Die Paare $(\tilde{q},\tilde{s})$ entsprechen nach Theorem 17.1.13 wieder Primidealen ersten Grades, die wir in diesem Zusammenhang als **quadratische Charaktere** bezeichnen. Die Menge der quadratischen Charaktere bezeichnen wir mit

$$\mathcal{B} := \{(\tilde{q},\tilde{s}) \in \mathbb{P}\times\mathbb{Z} \mid f(\tilde{s}) \equiv 0 \bmod \tilde{q} \text{ und } f'(\tilde{s}) \not\equiv 0 \bmod \tilde{q}\}$$

und nennen sie **Basis der quadratischen Charaktere**. Je mehr quadratische Charaktere wir für diesen Test nehmen, umso größer ist die Wahrscheinlichkeit, dass wir ein algebraisches Quadrat vorliegen haben, wobei es nach [23] sinnvoll ist, dass die Menge $\mathcal{B}$ Paare $(\tilde{q},\tilde{s})$ mit etwa $\lfloor 3\log(a)\rfloor$ Primzahlen $\tilde{q}$ umfasst. Wenn wir beim Matrixschritt in Kapitel 17.1.8 die Matrix A aufstellen, bilden wir auch Spalten bezüglich der quadratischen Charaktere. Da wir dort additiv arbeiten, setzen wir einen Eintrag gleich 1, falls $\left(\frac{k-ls}{q}\right) = -1$ gilt und gleich 0, falls $\left(\frac{k-ls}{q}\right) = 1$ gilt. Das entspricht einer Übertragung von der multiplikativen Gruppe $\mathbb{Z}/2\mathbb{Z}$ in die additive Gruppe $\mathbb{Z}/2\mathbb{Z}$, also muss in (17.4)

$$\sum_{(k,l)\in S_0} \left(\frac{k-l\tilde{s}}{\tilde{q}}\right) \equiv 0 \bmod 2 \ \forall(\tilde{q},\tilde{s})$$

gelten.

17.1.7 Quadratwurzeln in $\mathbb{Z}[\alpha]$

Haben wir durch die Verwendung quadratischer Charaktere mit großer Wahrscheinlichkeit ein algebraisches Quadrat $\gamma = \mu^2 \in \mathbb{Z}[\alpha]$ gegeben, so müssen wir im nächsten Schritt die Quadratwurzel aus γ ziehen. Betrachten wir kurz den analogen Fall in $\mathbb{Z}$, wo wir ebenfalls ein Quadrat $y^2 = f'(m)^2 \prod_{(k,l)\in S_0}(k-lm)$ konstruiert haben. Dort kennen wir die Primfaktorzerlegung von y^2 aus dem Siebverfahren, die ausschließlich gerade Primzahlpotenzen aufweist, und berechnen y durch das Halbieren der Potenzen. Dieses Verfahren lässt sich nicht eins zu eins auf algebraische Zahlen übertragen, denn dort kennen wir nur die Primfaktorzerlegung von $\mathcal{N}(\gamma)$. Üblicherweise berechnet man vom Polynom $h(x) = x^2 - \gamma \in \mathbb{Z}[\alpha][x]$ die Nullstellen. Da aber die Koeffizienten r_i von

$$\gamma = r_{d-1}\alpha^{d-1} + r_{d-2}\alpha^{d-2} + \cdots + r_1\alpha + r_0$$

von enormer Größe bei riesigen Zahlen a sind, ist die Nullstellenberechnung von $h(x)$ ein zu großer Rechenaufwand. Statt dessen bestimmen wir, wie in [13] vorgeschlagen, eine Primzahl q, sodass das Polynom $f(x)$ modulo q irreduzibel ist. Nach Theorem 2.0.7 ist $\mathbb{F}_q[x]/\langle f(x)\rangle$ ein Körper, und wir berechnen mit geringem Aufwand eine Nullstelle $\mu_q \bmod q$ von $h_q(x) = x^2 - \gamma \bmod q$, da die Koeffizienten modulo q reduziert sind. Wenn wir die Idee weiter verfolgen, so können wir mit dem selben Verfahren Lösungen μ_q modulo vieler Primzahlen q berechnen und „verkleben“ dann die Werte μ_q mit dem chinesischen Restsatz (Satz 2.0.8) zur Lösung μ. Es stellt sich das Problem, dass

es bei der Bestimmung der Nullstellen der Polynome $h_q(x)$ zwei Lösungen gibt – eine positive und eine negative – und wir wissen nicht, welche die „richtige“ für unsere Wurzel μ ist.
Alternativ können wir, wie in [21] vorgeschlagen, das Polynom $f(x)$ mit einem ungeraden Grad d wählen, sodass folgender Satz gilt.

Satz 17.1.23. *Sei $f \in \mathbb{C}[x]$ ein irreduzibles Polynom mit ungeradem Grad d, $\alpha \in \mathbb{C}$ eine Nullstelle von f und $\mathbb{Q}(\alpha) \cong \mathbb{Q}[x]/\langle f(x)\rangle$ nach Theorem 3.0.6 der entsprechende Zahlkörper, dann gilt*

$$\mathcal{N}(-\gamma) = -\mathcal{N}(\gamma).$$

Wählen wir bei der oben angesprochenen Methode immer nur die Lösungen μ_q mit positiver Norm, so können wir die Zahlen μ_q mit dem chinesischen Restsatz zu der gewünschten Lösung μ „verkleben“. Für genauere Details zur Berechnung der algebraischen Quadratwurzel, siehe [22], Kapitel 4.4 oder [69], Kapitel 5.4.2.

17.1.8 Der Matrixschritt

Haben wir ausreichend viele glatte Elemente $G(k,l) = k - lm$ bzw. $k - l\alpha$ gefunden, so können wir die Matrix A ähnlich wie beim QS in Kapitel 16 definieren. Für jedes Paar $(k,l) \in S$ definieren wir eine Zeile in A. In der ersten Spalte einer Zeile notieren wir den Eintrag 1 für negatives $G(k,l)$ und den Eintrag 0 für positives $G(k,l)$. In den nächsten r Spalten (wobei F_{p_r} unsere Faktorbasis ist) stehen die Exponenten e_i $(1 \leq i \leq r)$ der Primfaktorzerlegung $|G(k,l)| = \prod_{i=1}^{r} p_i^{e_i}$. Die nächsten

$$U = \sum_{\substack{q \leq p_r \\ q \in \mathbb{P}}} |\mathcal{F}(q)|$$

Spalten korrespondieren entsprechend der Anzahl der Primideale ersten Grades in der algebraischen Faktorbasis $\mathcal{F}_{p_r}$ mit den Primidealen $(q,s) \in \mathcal{F}_{p_r}$, in denen wir die Exponenten $f_{(q,s)}$ der Primidealzerlegung von $\langle k - l\alpha\rangle$ notieren. Die letzten $V = |\mathcal{B}|$ Spalten bestehen aus Einträgen bezüglich der Basis der quadratischen Charaktere. In einer Spalte eines quadratischen Charakters $(\tilde{q}, \tilde{s})$ setzen wir, wie schon am Ende von Kapitel 17.1.6 angesprochen, den Eintrag

$$\left(\frac{k - l\tilde{s}}{\tilde{q}}\right) = 1,$$

falls für das Legendre-Symbol $\left(\frac{k-l\tilde{s}}{\tilde{q}}\right) = -1$ gilt und

$$\left(\frac{k - l\tilde{s}}{\tilde{q}}\right) = 0,$$

falls für das Legendre-Symbol $\left(\frac{k-l\tilde{s}}{\tilde{q}}\right) = 1$ gilt. Die Matrix A hat schließlich die Form

$$A = \begin{array}{c} \\ (k_1,l_1) \\ (k_2,l_2) \\ \vdots \\ (k_z,l_z) \end{array} \begin{array}{c} \begin{array}{cccccccccc} -1 & p_2 & \dots & p_r & (q_1,s_1) & \dots & (q_U,s_U) & (\tilde{q}_1,\tilde{s}_1) & \dots & (\tilde{q}_V,\tilde{s}_V) \end{array} \\ \begin{pmatrix} e_{1_0} & e_{1_1} & \dots & e_{1_r} & f_{1_{(q_1,s_1)}} & \dots & f_{1_{(q_U,s_U)}} & \left(\frac{k_1-l_1\tilde{s}_1}{\tilde{q}_1}\right) & \dots & \left(\frac{k_1-l_1\tilde{s}_V}{\tilde{q}_V}\right) \\ e_{2_0} & e_{2_1} & \dots & e_{2_r} & f_{2_{(q_1,s_1)}} & \dots & f_{2_{(q_U,s_U)}} & \left(\frac{k_2-l_2\tilde{s}_1}{\tilde{q}_1}\right) & \dots & \left(\frac{k_2-l_2\tilde{s}_V}{\tilde{q}_V}\right) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ e_{z_0} & e_{z_1} & \dots & e_{z_r} & f_{z_{(q_1,s_1)}} & \dots & f_{z_{(q_U,s_U)}} & \left(\frac{k_z-l_z\tilde{s}_1}{\tilde{q}_1}\right) & \dots & \left(\frac{k_z-l_z\tilde{s}_V}{\tilde{q}_V}\right) \end{pmatrix} \end{array}. \tag{17.5}$$

Wie viele Zeilen z müssen wir für die Matrix A berechnen? Wenn wir uns an die letzten Kapiteln zurück erinnern, brauchen wir mehr Zeilen als Spalten, wenn wir garantiert eine Teilmenge $S_0 \subseteq S$

finden wollen, sodass die Zeilen $(k_i, l_i) \in S_0$ modulo 2 linear abhängig sind, das heißt, wir müssen mindestens

$$z > 1 + r + U + V$$

Zeilen (k_i, l_i) für die Matrix A konstruieren. Die Teilmenge $S_0 \subseteq S$ wählen wir so, dass

$$\sum_{j \in S_0} \begin{pmatrix} e_{j_0} \\ e_{j_1} \\ \vdots \\ e_{j_r} \\ f_{j_{(q_1,s_1)}} \\ \vdots \\ f_{j_{(q_U,s_U)}} \\ \left(\frac{k_j - l_j \tilde{s}_1}{\tilde{q}_1}\right) \\ \vdots \\ \left(\frac{k_j - l_j \tilde{s}_V}{\tilde{q}_V}\right) \end{pmatrix} = \begin{pmatrix} \sum_{j \in S_0} e_{j_0} \\ \sum_{j \in A_0} e_{j_1} \\ \vdots \\ \sum_{j \in A_0} e_{j_r} \\ \sum_{j \in A_0} f_{j_{(q_1,s_1)}} \\ \vdots \\ \sum_{j \in A_0} f_{j_{(q_U,s_U)}} \\ \sum_{j \in A_0} \left(\frac{k_j - l_j \tilde{s}_1}{\tilde{q}_1}\right) \\ \vdots \\ \sum_{j \in A_0} \left(\frac{k_j - l_j \tilde{s}_V}{\tilde{q}_V}\right) \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \mod 2 \tag{17.6}$$

gilt und wir die zwei Quadrate

$$y^2 = f'(m)^2 \prod_{(k,l) \in S_0} (k - lm) \quad \text{und} \quad \mu^2 = f'(\alpha)^2 \prod_{(k,l) \in S_0} (k - l\alpha)$$

bilden können, wobei μ^2 nach Kapitel 17.1.6 nur zu einer gewissen Wahrscheinlichkeit ein Quadrat ist.

17.1.9 Zusammenfassung

Wir wollen einen Faktor von einer ungeraden natürlichen Zahl a berechnen, die keine echte Potenz ist. Wir wählen ein Polynom f vom Grad d, wobei d durch die Größe der Zahl a bestimmt wird (siehe Kapitel 17.1.1). Weiters setzen wir $m = \lfloor \sqrt[d]{a} \rfloor$ und entwickeln

$$a = m^d + c_{d-1} m^{d-1} + \cdots + c_1 m + c_0$$

zur Basis m und definieren das Polynom $f(x) = x^d + c_{d-1} x^{d-1} + \cdots + c_1 x + c_0$. Eine der komplexen Wurzeln α von f bestimmt den Ring $\mathbb{Z}[\alpha]$. Weiters wählen wir eine Schranke $M \in \mathbb{N}$, definieren die Menge

$$S = \{(k, l) \in \mathbb{Z} \times \mathbb{Z} \mid 0 < |k|, l \leq M \text{ und } \gcd(k, l) = 1\}$$

und bilden die Elemente $k - lm \in \mathbb{Z}$ bzw. $k - l\alpha \in \mathbb{Z}[\alpha]$. Unser Ziel ist es eine Teilmenge $S_0 \subseteq S$ zu finden, sodass die Produkte

$$f'(m)^2 \prod_{(k,l) \in S_0} (k - lm) \quad \text{und} \quad f'(\alpha)^2 \prod_{(k,l) \in S_0} (k - l\alpha)$$

jeweils ein Quadrat in $\mathbb{Z}$ bzw. $\mathbb{Z}[\alpha]$ bilden. Wir wählen dafür eine geeignete Primzahl p_r nach Kapitel 17.1.5 und definieren eine rationale bzw. algebraische Faktorbasis

$$F_{p_r} = \{-1, p_1, \ldots, p_r\} \quad \text{bzw.} \quad \mathcal{F}_{p_r} = \{(q, s) \mid q \leq p_r \text{ Primzahl und } s \in \mathcal{F}(q)\},$$

wobei wir $\mathcal{F}(q) = \{s \in \mathbb{Z}/q\mathbb{Z} \mid f(s) \equiv 0 \bmod q\}$ setzen. Die Elemente von $\mathcal{F}_{p_r}$ entsprechen nach Kapitel 17.1.3 und Kapitel 17.1.4 Primidealen ersten Grades in $\mathbb{Z}[\alpha]$. Fassen wir die Elemente $k-l\alpha \in \mathbb{Z}[\alpha]$ als Hauptideale $\langle k - l\alpha\rangle$ auf, so haben diese, ebenfalls nach den Kapiteln 17.1.3 und 17.1.4, eine eindeutige Zerlegung in Primideale ersten Grades. Als nächsten Schritt behalten wir die Paare (k, l) mittels des Siebverfahrens aus Kapitel 17.1.5 in der Menge S, sodass $k - lm$ F_{p_r}- und $k - l\alpha$ $\mathcal{F}_{p_r}$-glatt sind. Haben wir nach Kapitel 17.1.8 zumindest $z = 2 + r + U + V$ solche Paare (k, l) gefunden und bestimmen eine Basis

$$\mathcal{B} := \{(\tilde{q}, \tilde{s}) \in \mathbb{P} \times \mathbb{Z} \mid f(\tilde{s}) \equiv 0 \bmod \tilde{q} \text{ und } f'(\tilde{s}) \not\equiv 0 \bmod \tilde{q}\}$$

aus quadratischen Charakteren mit etwa $\lfloor 3\log(a)\rfloor$ Primzahlen $\tilde{q}$, so stellen wir die Matrix A nach Gleichung (17.5) auf. Dann wählen wir eine Teilmenge $S_0 \subseteq S$ aus Zeilen der Matrix A, sodass Gleichung (17.6) erfüllt ist und berechnen die zwei Quadrate

$$y^2 = f'(m)^2 \prod_{(k,l)\in S_0} (k - lm) \in \mathbb{Z} \quad \text{und} \quad \gamma = \mu^2 = f'(\alpha)^2 \prod_{(k,l)\in S_0} (k - l\alpha) \in \mathbb{Z}[\alpha].$$

Aus y^2 ziehen wir aufgrund der Kenntnis der Primfaktorzerlegung von y^2 die Wurzel. Für das algebraische Quadrat γ, das nach Kapitel 17.1.6 nur zu einer gewissen Wahrscheinlichkeit eines ist, erhalten wir mittels Kapitel 17.1.7 die Wurzel μ aus γ und setzen $x := \psi(\mu)$. Schließlich berechnen wir die Teiler

$$p = \gcd(x \pm y, a)$$

von a. Ist p ein trivialer Teiler von a, so wählen wir eine neue Teilmenge S_0 bzw. fügen neue Zeilen zu A hinzu.

Algorithmus 17.1.24. (allgemeines Zahlkörpersieb)
Wir wollen eine zusammengesetzte ungerade natürliche Zahl a faktorisieren, die keine echte Potenz ist.

1. Teste, ob a eine echte Potenz und ungerade ist. Wenn nicht, fahre mit Punkt 2 fort, sonst haben wir einen echten Teiler gefunden.

2. - Setze $M = \lfloor L_a[\frac{1}{3}; \sqrt[3]{\frac{8}{9}}]\rfloor$,
     p_r als die größte Primzahl $\leq M$,
     $d = \lfloor \sqrt[3]{\frac{3\log(a)}{\log(\log(a))}}\rfloor$ und
     $m = \lfloor a^{\frac{1}{d}}\rfloor$.
   - Schreibe $a = m^d + c_{d-1}m^{d-1} + \cdots + c_1 m + c_0$ zur Basis m.
   - Definiere das Polynom $f(x) = x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0$.
   - Falls $f(x)$ nicht irreduzibel ist, gib die Faktorisierung $a = f(m) = g(m)h(m)$ aus. Andernfalls fahre fort.
   - Definiere die Polynome $F(x, y) = x^d + c_{d-1}x^{d-1}y + \cdots + c_1 xy^{d-1} + c_0 y^d$ und $G(x, y) = x - ym$.
   - Berechne $\mathcal{F}(q) = \{s \in \mathbb{Z}/q\mathbb{Z} \mid f(s) \equiv 0 \bmod q\}$ für alle Primzahlen $q \leq p_r$.
   - Setze $r' = \lfloor 3\log(a)\rfloor$ und berechne die Primzahlen $\tilde{q}$ mit $p_{r+1} \leq \tilde{q} \leq p_{r+r'}$.
   - Berechne die Menge $\mathcal{B} := \{(\tilde{q}, \tilde{s}) \mid f(\tilde{s}) \equiv 0 \bmod \tilde{q} \text{ und } f'(\tilde{s}) \not\equiv 0 \bmod \tilde{q}\}$.
   - Setze $V = |\mathcal{B}|$,
     $U = \sum_{\substack{q\leq p_r \\ q\in\mathbb{P}}} |\mathcal{F}(q)|$ und
     $z = 2 + r + U + V$.

3. Finde mittels des Siebverfahrens aus Kapitel 17.1.5 F_{p_r}-glatte Elemente $F(k, l) \cdot G(k, l)$ mit $0 < |k|, l \leq M$ und $\gcd(k, l) = 1$ heraus, füge diese zur Menge S hinzu und notiere die

Exponenten der Primfaktorzerlegungen von $F(k,l)$ und $G(k,l)$. Wiederhole diesen Vorgang so lange, bis mindestens z solche Elemente gefunden wurden, also $|S| \geq z$ gilt. Falls das nicht möglich ist, ändere die Startwerte p_r oder M.

4. Definiere die Exponentenmatrix A nach Gleichung (17.5).

5. Finde eine Teilmenge $S_0 \subseteq S$ (etwa durch das Gauß'sche Eliminationsverfahren), sodass Gleichung (17.6) erfüllt ist.

6. Berechne
$$y = f'(m)\sqrt{\prod_{(k,l)\in S_0}(k - lm)}$$
mit Hilfe der Primfaktorzerlegung der Elemente $k - lm$ und die Quadratwurzel
$$\mu = \sqrt{\gamma} = f'(\alpha)\sqrt{\prod_{(k,l)\in S_0}(k - l\alpha)}$$
in $\mathbb{Z}[\alpha]$ mit Hilfe eines der Verfahren aus Kapitel 17.1.7. Ist γ kein Quadrat, so wähle eine neue Teilmenge S_0 und erhöhe gegebenenfalls die Anzahl der quadratischen Charaktere. Sonst setze
$$y = \psi(\mu) \bmod a.$$

7. Berechne $p = \gcd(x \pm y, a)$.

Für ein ausführliches Beispiel zum allgemeinen Zahlkörpersieb siehe [22], Seite 61 ff.

17.2 Varianten des Zahlkörpersiebs

Ein wichtiger Ansatzpunkt zur Verbesserung des allgemeinen Zahlkörpersiebs ist die Wahl des Polynoms $f(x)$. Je kleiner der Grad und vor allem je kleiner die Koeffizienten von $f(x)$ sind, desto „schöner" ist die Gestalt von $\mathbb{Z}[\alpha]$ und desto geringer ist der Rechenaufwand im Algorithmus. Das sogenannte „spezielle Zahlkörpersieb" arbeitet mit einfacheren Polynomen und ist in seiner Laufzeit schneller als das allgemeine Zahlkörpersieb. Der Nachteil dieser Methode besteht darin – wie der Name verrät –, dass für diesen Algorithmus nur Zahlen mit spezieller Form zugelassen sind.

17.2.1 Das spezielle Zahlkörpersieb

Das spezielle Zahlkörpersieb (englisch: SNFS=Special Number Field Sieve) hat im Grunde die selbe Funktionsweise wie das GNFS. Der Unterschied liegt darin, dass die Zahl a in einer speziellen Form gegeben sein muss und wir dadurch eine einfachere Form des Rings $\mathbb{Z}[\alpha]$ erreichen. Die Elemente $k - l\alpha \in \mathbb{Z}[\alpha]$ können wir deswegen faktorisieren und damit das aufwändige Wurzelziehen aus Kapitel 17.1.7 vermeiden.
Sei
$$a = r^e - c \in \mathbb{N}$$
für eine kleine natürliche Zahl r und eine kleine ganze Zahl c. Beispielsweise eignen sich die Fermat-Zahlen $a = 2^{2^e} + 1$, die Mersenne-Zahlen $a = 2^p - 1$ ($p \in \mathbb{P}$) oder allgemeiner die Cunningham'schen Zahlen $a = r^e \pm 1$ für kleine r sehr gut. Nun wählen wir wie beim GNFS (siehe Kapitel 17.1.1) den Grad d von $f(x)$ in Abhängigkeit zur Größe von a. Dann berechnen wir die kleinste natürliche Zahl u, sodass $u \cdot d \geq e$ gilt, und definieren $v := c \cdot r^{u \cdot d - e}$. Wir setzen
$$f(x) = x^d - v \quad \text{und} \quad m = r^u.$$

Mit diesen Parametern und der Voraussetzung $r^e \equiv c \bmod a$ gilt

$$f(m) = m^d - v = r^{u \cdot d} - c \cdot r^{u \cdot d - e} \equiv r^{u \cdot d} - r^e \cdot r^{u \cdot d - e} \equiv 0 \bmod a,$$

also ist m eine Nullstelle von $f(x)$ modulo a. O.B.d.A ist das Polynom $f(x)$ irreduzibel, denn andernfalls berechnen wir einen nichttrivialen Faktor $h(m)$ oder $g(m)$ aus der echten Zerlegung $f(x) = g(x) \cdot h(x)$.
Sei α eine komplexe Nullstelle von $f(x)$ und $\mathcal{O}_\alpha$ der Ring ganzalgebraischer Zahlen in $\mathbb{Q}(\alpha)$. Für $\mathbb{Z}[\alpha]$ nehmen wir vereinfachend an, dass der Ring faktoriell ist (und damit ein Hauptidealring) und dass $\mathcal{O}_\alpha = \mathbb{Z}[\alpha]$ gilt, was im Allgemeinen nicht der Fall ist (siehe Bemerkungen 3.0.10 und 3.0.9). Andernfalls kann das spezielle Zahlkörpersieb mit einigen Abänderungen trotzdem durchgeführt werden (siehe [17], Kapitel 3.6 oder [63], Kapitel 3.10). Gilt unsere Annahme, so sind die Primideale ersten Grades Hauptideale und wir können wie folgt die Elemente $k - l\alpha$ eindeutig faktorisieren.
Wir ersetzen die Primideale ersten Grades (q, s) der algebraischen Faktorbasis $\mathcal{F}_{p_r}$ aus Kapitel 17.1.4 durch primitive Elemente $\delta_{(q,s)}$ der Primideale (q, s) und erweitern die Faktorbasis um die – nach dem Einheitensatz von Dirichlet (siehe [68], Seite 293 oder [47], Seite 41) endlich vielen – erzeugenden Elemente ϵ_i der Einheitengruppe $\mathcal{O}_\alpha^*$. Lässt sich nun das Hauptideal

$$\langle k - l\alpha \rangle = \prod (q, s)$$

einer $\mathcal{F}_{p_r}$-glatten algebraischen Zahl $k - l\alpha$ als Produkt von Primidealen ersten Grades schreiben, so können wir

$$k - l\alpha = \prod_i \epsilon_i \prod \delta_{(q,s)}$$

als Produkt für bestimmte primitive Elemente ϵ_i der Einheitengruppe $\mathcal{O}_\alpha^*$ darstellen. Haben wir genügend Paare $(k, l) \in S_0$ gefunden, sodass

$$\mu^2 = \prod_{(k,l) \in S_0} (k - l\alpha)$$

ein Quadrat ist, so können wir aufgrund der bekannten Faktorisierung der Elemente $k - l\alpha$ leicht die Wurzel ziehen. Der restliche Algorithmus verläuft ähnlich zum GNFS.
Eine PARI-Implementierung des speziellen Zahlkörpersiebs von Kevin Acres (ursprünglich von Scott Contini für Magma geschrieben) ist unter http://www.research-systems.com/primes/snfs.gp zu finden.

17.2.2 Vereinfachung des Polynoms $f(x)$

Es gibt auch Möglichkeiten, das Polynom $f(x)$ zu vereinfachen, wenn die Zahl a keine spezielle Form hat, und zwar wenn wir die sogenannte „Polynomgüte“ verbessern. Die Polynomgüte von $f(x)$ besagt, wie lange wir beim Sieben mit $f(x)$ brauchen, um möglichst viele glatte Werte zu erzeugen. Je mehr Werte das sind und je schneller das passiert, desto besser ist die Güte von $f(x)$. Dazu können wir beispielsweise bei der Auswahl von m mit verschiedenen Werten, also etwa $m = \lfloor \sqrt[d+1]{a} \rfloor$, $\lfloor \sqrt[d+2]{a} \rfloor$, $\lfloor \sqrt[d+3]{a} \rfloor, \ldots$, den Algorithmus starten. Das erzeugt verschiedene Polynome $f_m(x)$ und wir können (für den Anfang) feststellen, welches m sich am besten für die Auffindung glatter Werte eignet. Diese Vorgehensweise ist aber keine Garantie dafür, dass wir auf jeden Fall oder schneller einen Teiler von a finden.

17.2.3 Polynompaare

Für die Berechnung der Normen der algebraischen bzw. rationalen Elemente verwenden wir die homogenisierten Polynome $F(x, y) = y^d f(\frac{x}{y})$ bzw. $G(x, y) = y g(\frac{x}{y})$ für ein normiertes „algebraisches“

Polynom f vom Grad d, wobei $f(m) \equiv 0 \bmod a$ gilt und für ein „rationales“ Polynom $g(x) = x - m$. Wählen wir das „rationale“ Polynom g nicht vom Grad 1, sondern von beliebigem Grad und f, g beide irreduzibel (nicht zwingend normiert) mit $f(m) \equiv g(m) \equiv 0 \bmod a$, so lassen sich ähnlich zum GNFS zwei Quadrate konstruieren, um einen Teiler zu berechnen. Für Details siehe [23], Seiten 297-298.

17.2.4 Partielle Relationen

Ähnlich zu Kapitel 16.2.3, der Variante vom QS mit der großen Primzahl, erlauben wir bei der Version mit partiellen Relationen beim Siebvorgang nicht nur Paare (k, l) in der Menge S, sodass $k - lm$ F_{p_r}-glatt und $\mathcal{N}(k - l\alpha)$ $\mathcal{F}_{p_r}$-glatt sind, sondern auch Paare, wo einer der oder beide Werte nur „fast“ glatt sind. Nur fast glatt bedeutet in diesem Zusammenhang, dass genau einer der Primfaktoren von $k - lm$ bzw. $\mathcal{N}(k - l\alpha)$ nicht aus der Menge F_{p_r} stammen muss. Finden wir zwei Paare, die bei der gleichen Primzahl nicht glatt sind, so gewinnen wir aus diesen beiden Paaren eine neue Zeile für die Matrix A, ohne eine weitere Spalte in der Matrix A hinzufügen zu müssen.

17.2.5 Logarithmisches Sieben

Anstatt beim Siebvorgang Divisionen durchzuführen, können wir beim NFS, wie bei der entsprechenden Variante beim QS aus Kapitel 16.2.1, auch mit dem Logarithmus arbeiten und Subtraktionen durchführen. Wir verwenden beim logarithmischen Sieben den gerundeten Logarithmus der Zahlen $k - lm$ bzw. $\mathcal{N}(k - l\alpha)$ und ziehen beim Sieben den gerundeten Logarithmus der Primzahl p_i – durch die wir dividieren müssten – ab. Am Ende des Siebens sind genau die Elemente glatt, die betragsmäßig kleine Werte haben, genauer die Zahlen mit Betrag $\leq \log(p_r)$. Da die vielen, zum Teil überflüssigen Divisionen einen großen Teil des Rechenaufwandes ausmachen, ist die Ersparnis bei dieser Variante relativ groß.

17.3 Laufzeit und Eigenschaften

Das GNFS ist ein probabilistischer Algorithmus und eignet sich am Besten für große Zahlen a mit mindestens 100 Dezimalstellen, darunter gilt das QS als die schnellere Methode. Die heuristische Laufzeit für das GNFS beträgt

$$L_a[\frac{1}{3}; \sqrt[3]{\frac{64}{9}}] = O\left(\exp\left(\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right) n_a^{\frac{1}{3}} (\log(n_a))^{\frac{2}{3}}\right)\right)\right)$$

und ist damit asymptotisch schneller als das QS mit Laufzeit $L_a[\frac{1}{2}; 1]$. Wir wollen wie in Kapitel 16.3 für das QS auch für das GNFS die Herleitung der Laufzeit skizzieren. Das dafür notwendige Siebverfahren ist asymptotisch gesehen der einzig relevante Teil des Algorithmus', und wir nehmen an, dass die Wahrscheinlichkeit, dass die berechneten Zahlen $F(k, l) \cdot G(k, l)$ F_{p_r}-glatt sind genauso groß ist wie für eine zufällig gewählte Menge von Zahlen. Die Funktionsweise vom GNFS ist der vom QS sehr ähnlich, und wir wollen wie in Kapitel 16.3 daher wissen, wie viele Elemente wir aus der Menge $[1, x] \cap \mathbb{N}$ in Abhängigkeit einer natürliche Zahl x herausnehmen müssen, damit wir ausreichend viele F_{p_r}-glatte Zahlen erhalten. Nach Kapitel 16.3 beträgt diese Anzahl

$$L_x[\frac{1}{2}; \sqrt{2}] = O\left(\exp\left(\left(\sqrt{2} + o(1)\right) (\log(\log(x)) \cdot \log(x))^{\frac{1}{2}}\right)\right) \tag{17.7}$$

für eine Primzahl

$$p_r \approx L_x[\frac{1}{2}; \frac{1}{\sqrt{2}}].$$

Die obere Schranke x beim QS für $f(b) = b^2 - a$ liegt bei

$$x = a^{\frac{1}{2}+\epsilon}$$

für ein kleines, positives ϵ. Wie groß ist die obere Schranke für $F(k,l) \cdot G(k,l)$ beim Zahlkörpersieb? Die Polynome F und G haben die Form

$$F(k,l) = k^d + c_{d-1}k^{d-1}l + \cdots + c_1 k l^{d-1} + c_0 l^d \quad \text{bzw.} \quad G(k,l) = k - lm.$$

Für k und l gilt die Abschätzung

$$|k|, |l| \leq M,$$

für $m = \lfloor a^{\frac{1}{d}} \rfloor$ gilt $m \leq a^{\frac{1}{d}}$ und für die Koeffizienten c_i gilt

$$c_i \leq m \leq a^{\frac{1}{d}}.$$

Damit schätzen wir

$$\begin{aligned} |F(k,l) \cdot G(k,l)| &= \left|\left(k^d + c_{d-1}k^{d-1}l + \cdots + c_1 k l^{d-1} + c_0 l^d\right) \cdot (k - lm)\right| \\ &\leq (d+1)a^{\frac{1}{d}} M^d \cdot 2Ma^{\frac{1}{d}} \\ &= 2(d+1)a^{\frac{2}{d}} M^{d+1} \end{aligned}$$

ab. Setzen wir

$$x = 2(d+1)a^{\frac{2}{d}} M^{d+1} \tag{17.8}$$

als obere Schranke, so benötigen wir nach (17.7)

$$L_x[\frac{1}{2}; \sqrt{2}]$$

Paare (k,l), um ausreichend viele glatte Werte zu erzeugen. Die Schranke M muss daher

$$M^2 = L_x[\frac{1}{2}; \sqrt{2}] \approx p_r^2 \tag{17.9}$$

erfüllen. Setzen wir (17.9) in Gleichung (17.8) ein und logarithmieren, so erhalten wir

$$\log(x) \approx \log(2(d+1)) + \frac{2}{d}\log(a) + (d+1)\sqrt{\frac{1}{2}\log(x)\log(\log(x))}. \tag{17.10}$$

Der erste Term $\log(2(d+1))$ der rechten Seite in (17.10) ist asymptotisch vernachlässigbar. Für große d und a können wir den Term $(d+1)$ durch d ersetzen und erhalten

$$\log(x) \approx \frac{2}{d}\log(a) + d\sqrt{\frac{1}{2}\log(x)\log(\log(x))}. \tag{17.11}$$

Den Grad d wählen wir so, dass die Schranke x minimal ist, das heißt, wir leiten (17.11) nach d ab, also gilt

$$\frac{x'}{x} = \frac{-2}{d^2}\log(a) + \sqrt{\frac{1}{2}\log(x)\log(\log(x))} + \frac{dx'(1+\log(\log(x)))}{4x\sqrt{\frac{1}{2}\log(x)\log(\log(x))}}.$$

Nun setzen wir die Ableitung $x' = 0$ und berechnen daraus

$$d = (2\log(a))^{\frac{1}{2}} \left(\frac{1}{2}\log(x)\log(\log(x))\right)^{-\frac{1}{4}}.$$

Dieses d setzen wir in (17.11) ein und erhalten

$$(\log(x))^{\frac{3}{4}} = 2(2\log(a))^{\frac{1}{2}} \left(\frac{1}{2}\log(\log(x))\right)^{\frac{1}{4}}. \tag{17.12}$$

Gleichung (17.12) logarithmieren wir und setzen das Ergebnis

$$\log(\log(x)) = \frac{2}{3}\log(a)$$

in Gleichung (17.12) ein und erhalten

$$\log(x) = \frac{4}{\sqrt[3]{3}}(\log(a))^{\frac{2}{3}}(\log(\log(a)))^{\frac{1}{3}}.$$

Damit berechnen wir schließlich die Laufzeit

$$L_a[\frac{1}{3}; \sqrt[3]{\frac{64}{9}}] = O\left(e^{(\sqrt[3]{\frac{64}{9}}+o(1))n_a^{\frac{1}{3}}(\log(n_a))^{\frac{2}{3}}}\right)$$

für das Zahlkörpersieb. Um diese Laufzeit erreichen zu können, muss

$$d \approx \sqrt[3]{\frac{3\log(a)}{\log(\log(a))}}, \quad p_r \approx L_a[\frac{1}{3}; \sqrt[3]{\frac{8}{9}}] \quad \text{und} \quad M \approx p_r \tag{17.13}$$

gelten. Wie wir an dieser skizzenhaften Herleitung gesehen haben, spielt die Größe der Koeffizienten c_i des Polynoms $f(x)$ eine wichtige Rolle. Können wir die obere Schranke $a^{\frac{1}{d}}$ dafür senken, so sinkt automatisch die Laufzeit für das Zahlkörpersieb. Dies rechtfertigt auch die verschiedenen Variantenansätze für das Zahlkörpersieb zur Minimierung der Koeffizienten. Beispielsweise hat das SNFS (siehe Kapitel 17.2.1) die asymptotisch schnellere Laufzeit von

$$L_a[\frac{1}{3}; \sqrt[3]{\frac{32}{9}}]$$

aufgrund der betragsmäßig kleineren Koeffizienten.

Teil IV
Praktische Anwendungen

18 Faktorisierungsprogramme im Vergleich

1970 wurde die 39-stellige Fermat-Zahl $F_7 = 2^{2^7} + 1$ durch Michael Morrison und John Brillhart (siehe [44]) faktorisiert – ein Rekord. Einen Sommer lang mussten die beiden an der Zerlegung arbeiten und dafür eigens eine Implementierung für den Kettenbruchalgorithmus entwickeln. Will man heute diese Zahl faktorisieren, so muss man weder Implementierungen entwickeln, noch lange Berechnungen anstellen. PARI etwa faktorisiert F_7 in 0,2 Sekunden auf Knopfdruck. Wir wollen in diesem Kapitel die Möglichkeiten von einigen der heute aktuellen Faktorisierungsprogramme testen, inwieweit diese Programme Zahlen mit annähernd gleich großen Primteilern in 24 Stunden faktorisieren können[6]. Für diese Untersuchung verwenden wir Faktorisierungsrekorde aus der Vergangenheit (siehe Tabelle 18.0.1), wobei wir bei der Dezimalstellen- und Bitanzahl kleine Primteiler ignorieren und dabei das Produkt der größten Primfaktoren meinen.

Zahl	Größe der Primfaktoren[7]	Dezimalstellen	Bitanzahl	Jahr der Faktorisierung
$2^{2^7}+1$	p_{17}, p_{22}	39	128	1970
$3^{225}-1$	p_{15}, p_{15}, p_{18}	47	156	1981
$5^{91}-1$	p_{25}, p_{26}	51	169	1982
$11^{93}+1$	p_{22}, p_{43}	64	209	1983
$10^{71}-1$	p_{30}, p_{41}	71	236	1984
$5^{128}+1$	p_{26}, p_{62}	87	289	1986
$5^{160}+1$	p_{41}, p_{49}	90	299	1987
$11^{104}+1$	p_{41}, p_{60}	100	332	1988
2^{484+1}	p_{50}, p_{61}	111	369	1990
$10^{142}+1$	p_{32}, p_{36}, p_{50}	116	384	1991
RSA-120	p_{60}, p_{60}	120	397	1993

Tabelle 18.0.1: Testzahlen der Faktorisierungsprogramme

Im Folgenden stellen wir die Faktorisierungsprogramme kurz vor, deren Leistung wir untersuchen.

- **PARI/GP**[8] wurde ursprünglich von Henri Cohen und seinem Team 1985 entwickelt. Heute untersteht PARI der Lizenz von GPL (GNU General Public License) und wird von Karim Belabas und seinen Mitarbeitern geführt. Das kostenlose Programm PARI ist ein weit verbreitetes, komplettes Computeralgebra-System, das unter anderem Zahlenfaktorisierungen ermöglicht. Mit dem Befehl „factor(a)“ können Zahlen mittels der Probedivision, Shanks SQUFOF-Methode (siehe [19], Seite 427 oder [65]), Pollards-ρ-Methode, ECM und dem multipolynomialen QS zerlegt werden. Für weitere Informationen siehe Kapitel A und `http://pari.math.u-bordeaux.fr/`.
- **YAFU**[9] (=Yet Another Factorization Utility) ist ein 2010 entwickeltes kostenloses Programm

[6] mittels eines Intel® Core™ i3-350M 2,26 GHz Prozessors mit 4096 MB Arbeitsspeicher
[7] p_{xx} ist ein Primteiler mit xx Dezimalstellen
[8] Version 2.3.4
[9] Version 1.29

von Ben Buhrow, das rein für die Zahlenfaktorisierung ausgelegt ist. Dem Programm stehen Implementierungen von der Probedivision, von Pollards-ρ-Methode, vom Pollard-$(p-1)$-Algorithmus, von Williams-$(p+1)$-Methode, von ECM, von der Faktorisierungsmethode nach Fermat, von SQUFOF und von mehreren Varianten des QS zur Verfügung. Mit dem Faktorisierungsbefehl „factor(a)“ zeigt YAFU neben den Faktoren auch die zur Zerlegung verwendeten Methoden und Parameter an. Außerdem können die Faktorisierungsalgorithmen auch einzeln abgerufen werden. Für weitere Informationen und Download siehe http://sites.google.com/site/bbuhrow/home.

- **Mathematica**[10] ist eines der meist verwendeten Mathematikprogramme, ist kostenpflichtig und wird von Wolfram Research betrieben. Mathematica ist nicht auf die Algebra-Anwendung spezialisiert, besitzt aber einen Faktorisierungsbefehl „FactorInteger[a]“, der mittels Probedivision, dem Pollard-$(p-1)$-Algorithmus, ECM und QS ganze Zahlen zerlegen kann. Für weitere Informationen siehe http://www.wolfram.com/mathematica/.

- **GGNFS**[11] ist ein kostenloses, reines Faktorisierungsprogramm von Chris Monico, das vorwiegend mit dem GNFS arbeitet und mit dem laut Homepage http://www.math.ttu.edu/~cmonico/software/ggnfs/index.html Zahlen mit bis zu 140 Stellen faktorisiert werden können. Zum Auffinden kleinerer Primfaktoren (≤ 15 Stellen) verwendet GGNFS unter anderem die $(p-1)$-Methode, ECM und das QS. Eine detaillierte (Installations-) Anleitung ist unter der Homepage http://gilchrist.ca/jeff/factoring/nfs_beginners_guide.html von Jeff Gilchrist zu finden.

- **Dario Alpern** bietet online unter der Homepage http://www.alpertron.com.ar/ECM.HTM eine Implementierung von ECM, zu der man optional das QS mit Selbstinitialisierung zuschalten kann. Bei den Faktorisierungen aus Tabelle 18.0.1 haben wir aber ausschließlich mit ECM gearbeitet. Ein praktisches Tool bei dem auf Java gestützten Programm ist die Möglichkeit der parallelen Faktorisierung einer Zahl auf mehreren Computern.

Im Internet gibt es viele weitere kostenfreie Programme und Implementierungen für viele der in Teil III vorgestellten Faktorisierungsalgorithmen. Eine Link-Sammlung dazu ist in Anhang D zu finden. Das bekannte Mathematikprogramm „Matlab“ kann beispielsweise nicht die 10-stellige Fermat-Zahl F_5 zerlegen und ist daher nicht für die Faktorisierung geeignet.
Die siebente Fermat-Zahl wird von allen vorgestellten Programmen in maximal einer Sekunde in ihre Faktoren zerlegt und zeigt, dass der Fortschritt in den letzten 40 Jahren enorm war. Auch Zahlen mit 50 Dezimalstellen sind heute kein Problem mehr, so wird die 51-stellige Zahl $5^{91}-1$ von allen Programmen in ein paar Sekunden faktorisiert. Eine Ausnahme ist die ECM-Version von Dario Alpern, die die Zahl erst in über einer Minute zerlegt, weil die beiden relevanten Primteiler p_{25} und p_{26} von $5^{91}-1$ sehr nah beieinander liegen und eine Größe erreicht haben, bei der das Programm nicht mehr unmittelbar ein Ergebnis liefert. Bei der nächst größeren Zahl $11^{93}+1$ hat Darios ECM wieder nur noch eine Berechnungszeit von 8 Sekunden, da der kleinere der beiden großen Primteiler p_{22} einige Stellen weniger besitzt. Wie bereits in Kapitel 12.3 erwähnt, bestätigt uns diese Tatsache, dass ECM vor allem zum Aufspüren von „kleinen“ Teilern sehr nützlich ist und relativ unabhängig von der Größe der zu faktorisierenden Zahl ist. Diese Eigenschaft wiederholt sich bei den Zahlen $10^{71}-1$ mit 71 Stellen und $5^{128}+1$ mit 87 Stellen, wobei erstere Zahl bereits über 2,5 Stunden Rechenzeit für den 30-stelligen Teiler benötigt. Die Zahl $10^{142}+1$ kann ausschließlich von Darios ECM zerlegt werden – deren 116 Dezimalstellen sind zu groß für sämtliche anderen Programme und deren Methoden –, weil die beiden kleineren Primteiler p_{32} und p_{36} noch im machbaren Bereich von Darios ECM liegen. Warum für p_{32} und p_{36} nur 2 Stunden Rechenzeit benötigt werden, während für die Teiler p_{30} und p_{41} von $10^{71}-1$ eine halbe Stunde mehr an Rechenzeit beanspruchen (auch nach mehrmaligen Versuchen), muss an der Gestalt der Primteiler liegen.

[10] Version 8
[11] Version SVN 413

Zahl	PARI	YAFU	Mathematica	GGNFS	Darios ECM
$2^{2^7}+1$	172ms	562ms	218ms	1s	100ms
$3^{225}-1$	1s389ms	593ms	1s576ms	5s	400ms
$5^{91}-1$	1s856ms	1s353ms	8s580ms	2s	1m11s
$11^{93}+1$	2m15s877ms	16s761ms	3m32s957ms	1h9m	8s
$10^{71}-1$	3m27s029ms	59s440ms	29m12s310ms	1m7s	2h39m36s
$5^{128}+1$	25m40s400ms	1m42s353ms	32m1s710ms	1h10m	20m6s
$5^{160}+1$	7h56m	49m2s556ms	×	1h28m	×
$11^{104}+1$	×	5h19m5s	×	5h25m	×
$2^{484}+1$	×	×	×	23h18m	×
$10^{142}+1$	×	×	×	×	2h1m18s
RSA-120[12]	×	×	×	×	×

Tabelle 18.0.2: Testzahlen der Faktorisierungsprogramme

Die Programme PARI, YAFU und Mathematica haben sehr ähnliche Algorithmen implementiert und deshalb auch eine sehr ähnliche „Entwicklung“ der Rechenzeiten. In Tabelle 18.0.2 sehen wir, dass die Rechenzeiten der drei Programme, im Gegensatz zu Darios ECM, direkt proportional zu der Dezimalgröße der zu faktorisierenden Zahl wachsen. Der Grund liegt in der Verwendung vom QS (ECM ist zwar implementiert, wird aber nur bis zu einer gewissen Größen benutzt) bei der Faktorisierung großer Zahlen. Ein Manko von PARI ist die fehlende Implementation von Pollards $(p-1)$-Methode. So kann etwa die Zahl $a_1 = 7^{101}+2$ von Mathematica und YAFU in weniger als einer Sekunde zerlegt werden, während PARI dafür über 2,5 Minuten benötigt (siehe auch Kapitel 11.4). YAFU kann als schnellstes der drei Programme in 24 Stunden Zahlen mit etwa bis zu 100 Dezimalstellen faktorisieren.
Eine weitere auffällige Erscheinung ist die Berechnungszeit von über einer Stunde von GGNFS für die Zahl $5^{128}+1$, während anderen Programme hier mit maximal 3,5 Minuten auskommen. GGNFS benutzt für die Faktorisierung dieser Zahl das GNFS, während es beispielsweise für $5^{91}-1$ und $10^{71}-1$ das QS verwendet, das eine wesentlich geringere Anlaufzeit benötigt und deswegen bei dieser Größe von Zahlen (<100 Dezimalstellen) deutlich schneller ist. Auch im Vergleich zu anderen Programmen, wie PARI und YAFU, die das QS verwenden, ist GGNFS bis zu einer Zahlengröße von ca. 90 bis 100 Dezimalstellen langsamer. Danach steigt bei GGNFS die Rechenzeit für größer werdende Zahlen im Vergleich geringer an, sodass Zahlen mit mehr als 100 Dezimalstellen schneller oder überhaupt nur noch mit GGNFS faktorisiert werden, wie etwa bei der 111-stelligen Zahl $2^{484}+1$, die in knapp 24 Stunden zerlegt wird.
Zusammenfassend lässt sich sagen, dass sich zur Faktorisierung allgemeiner Zahlen bis 100 Stellen am besten die Programme PARI und YAFU eignen, wobei YAFU schneller als PARI arbeitet und auch Vorteile durch die Implementierung von Pollards $(p-1)$-Methode hat. Mathematica ist für „alltägliche“ Faktorisierungen auch geeignet, fällt aber in der Rechenzeit gegenüber den spezialisierten Programmen deutlich ab und kann Zahlen bis knapp 90 Dezimalstellen in 24 Stunden faktorisieren. Für Zahlen mit über 100 Dezimalstellen und etwa gleich großen Primteilern ist GGNFS aufgrund der Nutzung des GNFS das Schnellste. Die Implementierung von ECM von Dario Alpern kann sehr gut „kleine“ Teiler mit einer Größe von etwa 35 Dezimalstellen berechnen, auch wenn das heuristisch schnellere GNFS keine Teiler mehr findet.

[12] siehe Anhang F

Teil V
Rückblick und Ausblick

19 Geschichte der Zahlenfaktorisierung

Im Jahr 300 vor Christus setzte Euklid von Alexandria möglicherweise den Grundstein für den Beginn der Zahlenfaktorisierung durch die Entwicklung des Fundamentalsatzes der Arithmetik (Satz 1.0.4). Bewiesen wurde das Resultat erst um 1800 von C.F. Gauß. Mit dem Wissen der (im Wesentlichen) eindeutigen Primfaktorzerlegung war bis ins 17. Jahrhundert die Probedivision der einzige methodische Faktorisierungsalgorithmus, die keinem Erfinder zugeordnet werden kann. Zu dieser Zeit entwickelte Pierre Fermat seinen Faktorisierungsalgorithmus (Kapitel 13), der zwar keine schnellere Laufzeit als die Probedivision hat, aber die Basis für die meisten modernen Algorithmen legte. Damals war es noch weit verbreitet, sogenannte Faktorisierungstabellen am Ende von mathematischen Texten anzuhängen. Beispielsweise erstellte John Pell im Jahr 1668 eine Faktorisierungstabelle aller Zahlen bis 100000. Eine erste Weiterentwicklung der Idee von Fermat erfolgte im 18. Jahrhundert durch den von Legendre entwickelten Faktorisierungsalgorithmus mit Faktorbasen (Kapitel 14). Die Grenze der möglichen Zahlenfaktorisierung für allgemeine Zahlen lag damals bei 10 bis maximal 15 Stellen. Mit „allgemeinen Zahlen“ meinen wir Zahlen exklusive kleiner Primteiler. Faktorisierungen kleinerer Zahlen wurden bis Anfang des 20. Jahrhunderts weiter in Tabellen aufgelistet, etwa wurden 1909 so alle Zahlen bis 10 Millionen faktorisiert. D.H. Lehmer meinte dazu:

„More than half of the entries printed will never be read.
By 1909 my father had brought out his factor table for the first ten million.
This proved to be the factor table to end all factor tables.“

Die im Jahre 1903 von Frank Nelson Cole zerlegte 67. Mersenne-Zahl

$$M_{67} = 2^{67} - 1 = 147573952589676412927 = 193707721 \times 761838257287$$

mit 21 Dezimalstellen war vermutlich die erste Faktorisierung einer Zahl über 20 Stellen mit etwa zwei gleich großen Primteilern. Der Aufwand für diese Leistung umfasste damals 3 Jahre Sonntag für Sonntag ausgeführte Rechenarbeit, wofür man heute nicht einmal mehr 1 Sekunde benötigt (siehe beispielsweise Tabelle 10.3.1). Ein paar Jahre später wurden die Methoden erstmals durch den Einsatz von maschinellen Rechnern unterstützt. 1927 faktorisierte D.H. Lehmer mit Hilfe der sogenannten „bicycle chain sieve machine“, einer mit Fahrradketten betriebenen Maschine, die Zahl

$$10^{20} + 1 = 73 \times 137 \times 1676321 \times 5964848081.$$

Das erste Modell der Maschine konnte etwa 50 Zahlen in der Sekunde verarbeiten, 6 Jahre später hatte sie bereits eine Rechenleistung von 5000 Schritten pro Sekunde, womit die Zahlen

$$2^{79} - 1 = 2687 \times 202029703 \times 1113491139767$$

und

$$2^{93} + 1 = 3^2 \times 529510939 \times 715827883 \times 2903110321$$

faktorisiert wurden – Zahlen mit immerhin 21 bzw. 28 Dezimalstellen (exklusive der kleinsten Teiler). In den 70er-Jahren gab es einen weiteren großen Fortschritt einerseits durch den Kettenbruchalgorithmus, der zwar schon seit 1931 durch D. H. Lehmer und R. E. Powers bekannt war (siehe [36]), doch erst durch den Einsatz besserer Computer und die Implementierung von Michael A. Morrison und John Brillhart (siehe [44]) rekordtauglich wurde. Andererseits basiert der 1978 entwickelte

RSA-Algorithmus (Kapitel 8) – der eines der ersten entwickelten Public-Key-Verfahren war und diese eine große Vereinfachung in der Kryptographie darstellen – auf dem Faktorisierungsproblem. Mit einem Public-Key-Verfahren ist es möglich verschlüsselte Daten zu verschicken, ohne vorher mit dem Kommunikationspartner persönlich Schlüssel getauscht zu haben. Dadurch wurde ein kommerzielles Interesse an der Zahlenfaktorisierung geweckt, sodass es auch in der Forschung zu einer stärkeren Forcierung als bis dahin kam. 1991 wurde von der Firma RSA Laboratories die sogenannte RSA Factoring Challange ins Leben gerufen, ein mit Preisgeld dotierter Wettbewerb zum Faktorisieren bestimmter RSA-Zahlen (siehe http://www.rsa.com/rsalabs/node.asp?id=2091). Die RSA Factoring Challange wurde 2007 wieder eingestellt.
Mittels des Kettenbruchalgorithmus' wurde die 7. Fermat-Zahl

$$F_7 = 2^{2^7} + 1 = 59649589127497217 \times 5704689200685129054721,$$

eine Zahl mit bereits 39 Dezimalstellen, 1970 durch Michael A. Morrison und John Brillhart faktorisiert, wobei es damals noch unvorstellbar war, dass Zahlen mit über 100 Dezimalstellen jemals faktorisiert werden sollten. Martin Gardner versuchte sich 1977 an einer Zahl mit 129 Dezimalstellen, die als RSA-129 bekannt werden sollte, scheiterte jedoch und meinte danach in seiner Kolumne „Mathematical Recreations", dass die Faktorisierung der Zahl Millionen an Jahre benötigen würde. Er irrte, denn mit Hilfe der Entwicklung des QS 1982 wurde bereits 1994, 17 Jahre später,

$$\begin{aligned}\text{RSA-129} = {} & 3490529510847650949147849619903898133417764638493387843990820577 \\ & \times \\ & 3276913299326670954996198819083446141317764296799294253979828853 3\end{aligned}$$

durch Paul Leyland, Michael Graff, Derek Atkins und Arjen Lenstra mittels des multipolynomialen quadratischen Siebs (siehe Kapitel 16.2.2) faktorisiert. Die 100-Dezimalstellen-Grenze allgemeiner Zahlen wurde bereits im Jahr 1988 geknackt, als über das Internet die Zahl $11^{104} + 1$ ebenfalls mit dem MPQS faktorisiert wurde. Seit dieser Zeit ging es unter anderem wegen des 1993 entwickelten Zahlkörpersiebs mit der Rekordjagd Schlag auf Schlag. 1996 wurden Zahlen mit über 130 Stellen faktorisiert, 1999 mit über 150 Stellen und 2005 allgemeine Zahlen mit über 200 Stellen (siehe Tabelle 19.0.1 und Abbildung 21.0.1).

Jahr	Zahl	Dezimalstellen	Bitanzahl	Methode
1988	$11^{104} + 1$	100	332	MPQS
1991	$10^{142} + 1$	116	384	MPQS
1993	RSA-120	120	397	MPQS
1994	RSA-129	129	426	MPQS
1996	RSA-130	130	430	GNFS
1999	RSA-155	155	463	GNFS
2002	$2^{953} + 1$	158	954	GNFS
2003	RSA-576	174	576	GNFS
2005	RSA-200	200	663	GNFS
2009	RSA-768	232	768	GNFS

Tabelle 19.0.1: Faktorisierungsrekorde allgemeiner Zahlen seit 1988 (Stand 2012)

20 Aktueller Stand

Seit 2009 liegt der Faktorisierungsrekord für Zahlen in allgemeiner Form, also Zahlen ohne kleiner Primteiler und von keiner speziellen Form, bei 232 Dezimalstellen. Ein Team um Thorsten Kleinjung zerlegte damals mit dem GNFS (siehe [33]) die 768-Bit RSA-Zahl

$$\begin{aligned}\text{RSA-768} = {} & 33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489\\ & \times\\ & 36746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917.\end{aligned}$$

Das dafür verwendete „algebraische“ Polynom

$$\begin{aligned}f(x) = {} & 265482057982680\ x^6\\ & + 1276509360768321888\ x^5\\ & - 50068156978001383517968 28\ x^4\\ & - 46477854471727854271772677450\ x^3\\ & + 65254372619359893971096673718947 85\ x^2\\ & - 18185779352088594356726018862434803054\ x\\ & - 27756526679154388199521619971380110334312 0\end{aligned}$$

und „rationale“ Polynom (siehe Kapitel 17.2.3)

$$g(x) = 346610035504925018514458 29\ x - 12911874565800212231635477915748108 81$$

wurden in einem halben Jahr mittels 80 Prozessoren konstruiert. Die Rechenzeit für die gesamte Faktorisierung betrug über 4 Jahre auf einigen zusätzlichen Prozessoren, wobei allein für das Siebverfahren fast 2 Jahre bzw. 1500 Prozessor-Jahre[1] in Anspruch genommen wurden (siehe [33] bzw. [20]). Auf der Internetseite [20] findet man eine Liste mit dem aktuellen Stand der größten je faktorisierten allgemeinen Zahlen.
Mit der Elliptic Curve Method sind keine Rekordfaktorisierungen möglich, da sie – wie in Kapitel 12.3 angesprochen – nur besonders für Zahlen mit relativ kleinen Primfaktoren von aktuell bis zu etwa 70 Dezimalstellen geeignet, dafür aber weitgehend unabhängig von der Größe der zu faktorisierenden Zahl ist. Der zur Zeit größte mit ECM berechnete Primfaktor

$$p_{73} = 1808422353177349564546512035512530001279481259854248860454348989451026887$$

ist ein Teiler der Zahl $M_{1181} = 2^{1181} - 1$ und wurde 2010 von J. Bos, T. Kleinjung, A. Lenstra und P. Montgomery gefunden. Eine aktuelle Tabelle der größten gefundenen Faktoren mittels ECM findet man unter [77].

20.1 Faktorisierungen Zahlen spezieller Form

Die größte je faktorisierte Zahl ohne kleine Primteiler ist die 1039. Mersenne-Zahl

$$M_{1039} = 2^{1039} - 1,$$

eine Zahl mit 313 Dezimalstellen, die 2007 durch ein Team um Thorsten Kleinjung zerlegt wurde. Diese Zahl kann allerdings aufgrund ihrer speziellen Form durch das spezielle Zahlkörpersieb (siehe

Jahr	Zahl	Dezimalstellen	Bitanzahl	Methode
1994	$\frac{12^{151}-1}{11}$	162	535	SNFS
1997	$12^{167}+1$	181	599	SNFS
1998	$32633^{41}-1$	186	615	SNFS
1999	$\frac{10^{211}-1}{9}$	211	698	SNFS
2000	$2^{773}+1$	233	774	SNFS
2003	$2^{809}-1$	244	809	SNFS
2004	$2^{821}+2^{411}+1$	248	822	SNFS
2006	$\frac{6^{353}-1}{5}$	274	911	SNFS
2007	$2^{1039}-1$	313	1039	SNFS

Tabelle 20.1.1: Faktorisierungsrekorde spezieller Zahlen seit 1994 (Stand 2012)

Kapitel 17.2.1) leichter als Zahlen in allgemeiner Form faktorisiert werden. Auf der Internetseite [20] findet man eine Liste mit dem aktuellen Stand der größten je faktorisierten speziellen Zahlen.
Viele der Mersenne-Zahlen $M_p = 2^p - 1$ für eine Primzahl p wurden bereits faktorisiert, die meisten noch nicht. Die kleinste Mersenne-Zahl, die bis jetzt noch nicht komplett in seine Faktoren zerlegt wurde, ist die 280-stellige Zahl M_{929} mit einem 214-stelligen zusammengesetzten Faktor. Die kleinste Mersenne-Zahl ohne bekannten Faktor ist die 320-stellige Zahl M_{1061}. Eine Internetplattform mit aktuellem Stand für Faktorisierungsrekorde von Mersenne-Zahlen bietet [72] von Sam Wagstaff.
Die kleinste zusammengesetzte Fermat-Zahl $F_m = 2^{2^m} + 1$ ist F_5, 1732 von L. Euler faktorisiert. Im Jahr 1988 wurde die bisher größte Fermat-Zahl F_{11} durch R. P. Brent und F. Morain vollständig zerlegt. Dieser Rekord liegt allerdings schon 23 Jahre zurück. Seit damals konnte nur 1990 F_9 durch ein Team um A.K.Lenstra und M.S.Manasse und 1995 F_{10} durch R.P.Brent, aber keine der Fermat-Zahlen F_m mit $m \geq 12$ komplett faktorisiert werden. Für F_{12} ist beispielsweise die Zerlegung eines 1133-stelligen, zusammengesetzten Faktors unbekannt – mit der Faktorisierung ist erst in größerer Zukunft zu rechnen. Die kleinste Fermat-Zahl ohne bekannten Faktor ist F_{20} mit 315653 Dezimalstellen. Eine Internetplattform mit aktuellem Stand für Faktorisierungsrekorde von Fermat-Zahlen bietet [4] von Ray Ballinger und Mark Rodenkirch.

21 Die Zukunft des Faktorisierens

Wie große Zahlen werden wir in Zukunft faktorisieren können, wie geht die Entwicklung der Faktorisierungsrekorde weiter? Diese Frage lässt sich schwer beantworten. Zweifelsfrei lässt sich lediglich sagen, dass sie weiter gehen wird. Die Rechenleistung von Computern wird in den nächsten Jahren sicherlich noch verbessert werden. Nach dem unbewiesenen „Gesetz von Moore“ verdoppeln Computer alle 18 Monate ihre Geschwindigkeit – eine Vermutung, die jedenfalls in den letzten Jahren Gültigkeit hatte. Des Weiteren werden auch die aktuellen Algorithmen stetig weiterentwickelt und die Implementierungen verbessert. Es ist auch nicht ausgeschlossen, dass ein neuer, schnellerer Algorithmus entwickelt wird; die Veröffentlichung des Zahlkörpersiebs liegt bereits 19 Jahre zurück. Nehmen wir an, dass die Kurven aus Abbildung 21.0.1 ihre annähernd konstante Steigung beibehält, so könnten wir mit einem durchschnittlichen Zuwachs von 6-7 Dezimalstellen pro Jahr rechnen und im Jahr 2050 Zahlen bis etwa 500 Dezimalstellen faktorisieren. Das hätte starke Auswirkungen auf die Nutzer des RSA-Verfahrens, dessen Sicherheit (vermutlich) von der Komplexität des Faktorisierungsproblems lebt. Heutzutage werden teilweise für Verschlüsselungen 512-Bit Schlüssel (etwa 155

[1] Ein Prozessor-Jahr ist die Leistung von einem 2,2 GHz AMD Opteron in einem Jahr

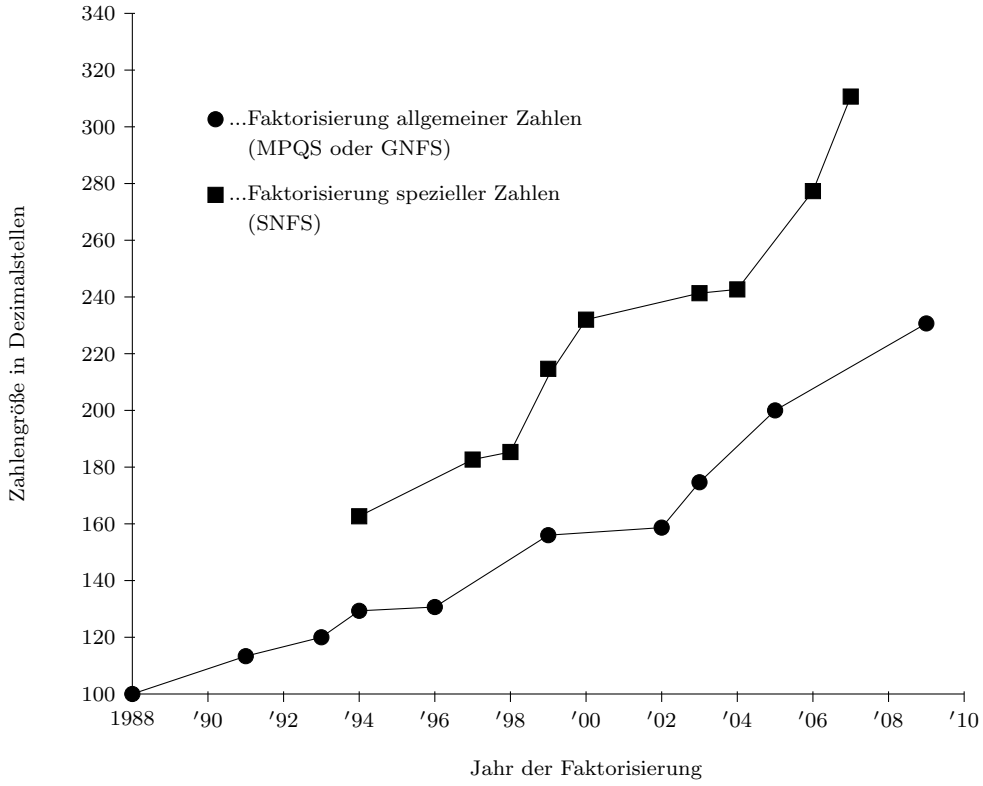

Abbildung 21.0.1: Entwicklung der Faktorisierungsrekorde (Stand 2012)

Dezimalstellen) verwendet, die als unsicher gelten. Als sicher gelten erst Schlüssel ab 1024 Bits (etwa 308 Dezimalstellen). Spätestens im Jahr 2025 müssten aber die Schlüssellänge wieder deutlich erhöht werden, da diese Zahlen dann faktorisiert werden könnten. Es könnte aber auch sein, dass, sollte der sogenannte Shor-Algorithmus zum Einsatz kommen, das RSA-Verfahren vollkommen nutzlos wird.

21.1 Der Shor-Algorithmus

1994 veröffentlichte Peter Shor den Shor-Algorithmus, einen probabilistischen polynomialen Algorithmus mit Laufzeit

$$O(n_a^3)$$

für eine zu faktorisierende Zahl a, der dadurch deutlich schneller als alle heute benutzbaren Algorithmen ist. Das Problem bei diesem Algorithmus liegt allerdings darin, dass sein Einsatz für einen Quantencomputer konzipiert wurde, also ein Gerät, dass es bis dato so nicht gibt. Würde ein solcher Computer gebaut werden, so würde die Rechenzeit für alle Faktorisierungen drastisch sinken und sämtliche kryptographischen Verfahren, die auf dem Faktorisierungsproblem beruhen, wie dem RSA-Verfahren (Kapitel 8), wären auf einem Schlag nutzlos.
Der Shor-Algorithmus wird in einen klassischen und einen Quantenteil unterteilt, wobei wir hier den Fokus auf den klassischen Teil legen.

Algorithmus 21.1.1. (Der Shor-Algorithmus)
Wir wollen einen Teiler einer natürliche Zahl a berechnen.

1. Wähle zufällig eine Zahl $b \in (\mathbb{Z}/a\mathbb{Z})^*$.
2. Bestimme mittels dem Quantenteil des Algorithmus' die Ordnung o_b von b in $(\mathbb{Z}/a\mathbb{Z})^*$.
3. Ist o_b ungerade oder gilt $b^{\frac{o_b}{2}} \equiv -1 \bmod a$, beginne mit einem neuen Startwert b. Andernfalls fahre fort.
4. Berechne den echten Teiler $p = \gcd(b^{\frac{o_b}{2}} - 1, a)$.

Wir überprüfen die Korrektheit von Algorithmus 21.1.1. Es gilt

$$a \mid b^{o_b} - 1 = \left(b^{\frac{o_b}{2}} - 1\right)\left(b^{\frac{o_b}{2}} + 1\right)$$

und

$$b^{\frac{o_b}{2}} \not\equiv 1 \bmod a.$$

Aus Punkt 3 des Algorithmus' folgt, dass $\frac{o_b}{2}$ eine natürliche Zahl ist und $a \nmid \left(b^{\frac{o_b}{2}} + 1\right)$ gilt und damit muss

$$1 < \gcd(b^{\frac{o_b}{2}} - 1, a) < a$$

gelten.
Die Ordnung o_b von b wird mittels eines Quantencomputers berechnet, der im Wesentlichen den Vorteil gegenüber heutigen Computern besitzt, dass er Operationen nicht hintereinander, sondern parallel ausführen kann. Für genauere Ausführungen siehe [23], Seiten 418-424. Quantencomputer sind aber nicht reine Zukunftsmusik, da 2001 bereits ein Modell gebaut wurde, das die Zahl 15 faktorisieren konnte (siehe [71]).

A Algebra Software PARI/GP

Das Programm PARI/GP ist eine von Henri Cohen und seinem Team entwickelte Algebra Software, die unter GPL (GNU General Public License) betrieben und von Karim Belabas und seinen Mitarbeitern geführt wird und kann unter der Homepage http://pari.math.u-bordeaux.fr/ kostenlos heruntergeladen werden. Einige der Faktorisierungsalgorithmen sind nicht in einer Pseudoprogrammiersprache, sondern als funktionstüchtige PARI-Codes verfasst, sodass man sie aus der Arbeit kopieren und in PARI verwenden kann. Um die angegebenen Algorithmen zum Laufen zu bringen, kopiert man den Algorithmus in einen Texteditor, speichert die Datei im PARI-Ordner als „DATEINAME.gp“, startet das Programm PARI und lädt dort den Algorithmus mittels der Eingabe „\r DATEINAME“.
PARI stellt die üblichen Funktionen bereits implementiert zur Verfügung, wie $gcd(a,b)$, $lcm(a,b)$, $Mod(a,n)$ bzw. $a\%n$, $sqrt(a)$, $ceil(a)$, $floor(a)$, usw. Weitere für uns interessante Funktionen sind:

$factor(a)$ berechnet die vollständige Faktorisierung von a

$isprime(a)$ testet, ob a eine Primzahl ist

$primepi(a)$ Primzahlzählfunktion (Definition 1.0.2)

$eulerphi(a)$ Euler'sche Phi-Funktion (Definition 1.0.2)

$znorder([a]_n)$ gibt die Ordnung vom Element $[a]_n \in (\mathbb{Z}/n\mathbb{Z})^*$ an

$contfrac(a,k)$ gibt den k-ten Näherungsbruch für $a \in \mathbb{R}$ an (Kapitel 5)

$kronecker(a,p)$ berechnet das Legendre-Symbol $\left(\frac{a}{p}\right)$ (Kapitel 4)

Für elliptische Kurven (Kapitel 6) und deren Anwendung (Kapitel 12) stellt PARI ebenfalls bereits vorprogrammierte Funktionen zur Verfügung. Eine elliptische Kurve definieren wir mittels eines Vektors „$E = [a1, a2, a3, a4, a6]$“ mit 5 Komponenten. Punkte der Kurve werden als zweidimensionale Vektoren $P = [x, y]$ eingegeben, das neutrale Element ∞ als [0]. Wollen wir zwei Punkte P und Q auf der Kurve E addieren, so geben wir „$elladd(E,P,Q)$“ ein.
Wir können in PARI auch selbst Funktionen, wie etwa Faktorisierungsalgorithmen, mit dem Befehl

```
algorithmus(a,b)=                    /* Name und Eingabevariablen des Algorithmus' */
 {local(c,d,e);                                    /* verwendete Variablen */
 ALGORTHMUS                                           /* Algorithmuskörper */
}                                                 /* Ende des Algorithmus' */
```

in einem Texteditor speichern und in PARI laden. Neben den vorprogrammierten Funktionen können wir die dafür üblichen Schleifen, wie for, if, $while$ usw. verwenden. Die verschiedenen Befehle werden dabei mit einem „;“ getrennt. Die selbst programmierten Funktionen

$logp(p,x)$ berechnet den p-Logarithmus von x

$poleval(pol,y)$ wertet das Polynom pol an der Stelle y aus

$elladdmod(E,P,Q,n)$ berechnet $P+Q$ mod n auf der elliptischen Kurve E

$ellpowmod(E,P,k,n)$ berechnet $k \cdot P$ mod n auf der elliptischen Kurve E

$multigcd(v)$ berechnet $\gcd(v_1, v_2, \ldots, v_k)$ für einen Vektor $v = (v_1, v_2, \ldots, v_k)$

$multilcm(v)$ berechnet $\mathrm{lcm}(v_1, v_2, \ldots, v_k)$ für einen Vektor $v = (v_1, v_2, \ldots, v_k)$

verwenden in dieser Arbeit und sind in Anhang B angeführt. Für die zahlreichen weiteren Funktionen und Möglichkeiten von PARI sei auf das Manual unter http://pari.math.u-bordeaux.fr/pub/pari/manuals/2.3.3/users.pdf verwiesen.

B PARI/GP-Programmierungen

B.1 Pollards $(p-1)$-Algorithmus

Folgender Algorithmus beschreibt die „Basis-Version“ von Pollards $(p-1)$-Algorithmus.

```
pollp1(a,B)=
{local(p,b,V,i);
 V=1;
 b=1;
 while(b==0||b==1,
        b=random(a)
        );
 p=gcd(b,a);
 if(p>1,
        print(p," ist ein Teiler von ",a)
        );
 for(i=1,B,
        V=lcm(V,i)
        );
 b=lift(Mod(b,a)^V);
 p=gcd(b-1,a);
 if(p>1&&p<a,
        print(p," ist ein Teiler von ",a)
        );
 if(p==1,
        print("B = ",B," ist zu klein fuer ",a)
        );
 if(p==a,
        print("B = ",B," ist zu gross fuer ",a)
        )
}
```

B.2 Variante der Elliptic Curve Method

Der im folgenden beschriebene Algorithmus zeigt die Variante mit großer Primzahl von ECM aus Kapitel 12.2.1.

```
ecmvar(a,B1,B2)=
{local(p,Q,Q0,E,a4,a6,q,q1,q2);
 p=gcd(6,a);
 if(p!=1,
         print(p," ist ein Teiler von ",a)
         );
 while(p==1||p==a,
         Q=[random(a),random(a)];
         a4=random(a);
         a6=Q[2]^2-Q[1]^3-Q[1]*a4;
         p=gcd(4*a4^3+27*a6^2,a);
         if(p!=1&&p!=a,
                 print(p," ist ein Teiler von ",a);
```

```
                break(5)
                );
        E=[0,0,0,a4,a6];
        q=2;
        while(q<=B,
                p=gcd(Q[2],a);
                if(p!=1&&p!=a,
                        print(p," ist ein Teiler von ",a);
                        break
                        );
                if(p==a,
                        break
                        );
                Q=ellpowmod(E,Q,q^(floor(logp(q,B))),a);
                q=nextprime(q++)
                );
        Q0=Q;
        Q=ellpowmod(E,Q,q,a);
        p=gcd(Q[2],a);
        if(p!=1&&p!=a,
                print(p," ist ein Teiler von ",a);
                break
                );
        if(p==1,
                q1=q;
                q2=nextprime(q++);
                while(q2<=B2,
                        d=q2-q1;
                        Q=elladdmod(E,Q,ellpowmod(E,Q0,d,a),a);
                        p=gcd(Q[2],a);
                        if(p!=1&&p!=a,
                                print(p," ist ein Teiler von ",a);
                                break
                                );
                        if(p==a,
                                break
                                );
                        q1=q2;
                        q2=nextprime(q2++)
                        )
                )
        );
 if(p==1||p==a,
        print("Es wurde kein Teiler gefunden")
        )
}
```

B.3 Lehmans Algorithmus

Folgender Algorithmus von Lehman bildet eine Variante des Fermat-Algorithmus aus Kapitel 13.

```
lehman(a)=
{local(x,b,k,c,z,p);
 b=a^(1/3);
 p=2;
 while(p<=floor(b),
         if(a%p==0,
                 print(p," ist ein Teiler von ",a);
                 break(5)
                 );
         p=nextprime(p++)
         );
 for(k=1,ceil(b),
        c=sqrt(4*k*a);
        for(x=ceil(c),floor(c+sqrt(b)/(4*sqrt(k))),
                z=x^2-4*k*a;
                if(z==sqr(floor(sqrt(z))),
                        p=gcd(floor(x+sqrt(z)),a);
                        print(p," ist ein Teiler von ",a);
                        break(5)
                        )
                )
        )
}
```

B.4 Der p-Logarithmus

Dieser Algorithmus berechnet den p-Logarithmus einer reellen Zahl x.

```
logp(p,x)=
{local();
 log(x)/log(p)
}
```

B.5 Auswerten eines Polynoms an der Stelle x_0

Dieser Algorithmus wertet das Polynom $f(x)$ an der Stelle x_0 aus.

```
poleval(pol,x0)=
{local(x);
 x=x0;eval(pol)
}
```

B.6 Multi-gcd und Multi-lcm

Der erste der beiden Algorithmen berechnet den größten gemeinsamen Teiler und der zweite das kleinste gemeinsame Vielfache der Zahlen $v_1, v_2, \ldots, v_k$. Beide Algorithmen stammen von Fernando

Rodriguez-Villegas (University of Texas at Austin) und Ariel Martin Pacetti (University of Texas at Austin) aus dem Jahr 2001.

```
multigcd(v)=
{local(mgcd);
 mgcd=gcd(v[1],v[2]);
 for(k=1,length(v)-2,
         mgcd=gcd(mgcd,v[k+2])
         );
 mgcd
}
```

```
multilcm(v)=
{local(mlcm);
 mlcm=lcm(v[1],v[2]);
 for(k=1,length(v)-2,
         mlcm=lcm(ans,v[k+2])
         );
 mlcm
}
```

B.7 Punkteaddition und Punktevervielfachung modulo n auf elliptischen Kurven

Der erste Algorithmus addiert zwei Punkte P_1 und P_2 modulo einer natürlichen Zahl n auf einer elliptischen Kurve E, der zweite berechnet das Vielfache k eines Punktes P modulo n auf einer elliptischen Kurve E. Beide Algorithmen stammen von Fernando Rodriguez-Villegas (University of Texas at Austin) und Ariel Martin Pacetti (University of Texas at Austin) aus dem Jahr 2001.

```
elladdmod(E,P1,P2,n)=
{local(aux,ans,lambda);
 if(P1==[0],
         return(P2),
         if(P2==[0],
                 return(P1)
                 )
         );
 if(P1!=P2,
         aux=bezout(P2[1]-P1[1],n);
         if(aux[3]!=1,
                 return([aux[3],n/aux[3]]),
                 lambda=(P2[2]-P1[2])*aux[1]
                 );
         [aux=((lambda^2-P1[1]-P2[1])%n),
         (-lambda*aux-(P1[2]-lambda*P1[1]))%n],
         aux=4*P1[1]^3+4*E[4]*P1[1]+4*E[5];
         aux=bezout(aux,n);
         if(aux[3]!=1,
```

```
                return([aux[3],n/aux[3]]),
                lambda=aux[1]
                );
        [ans=((P1[1]^4-2*E[4]*P1[1]^2-8*E[5]*P1[1]+E[4]^2)*lambda)%n,
        -((ans-P1[1])*bezout(2*P1[2],n)[1]*(3*P1[1]^2+E[4])+P1[2])%n]
        )
}
```

```
ellpowmod(E,P,k,n)=
{local(aux,ans,l,temp);
 aux=binary(k);
 ans=[0];
 l=length(aux);
 temp=P;
 for(k=1,l,
        if(aux[l-k+1]==1,
                ans=elladdmod(E,ans,temp,n)
                );
        temp=elladdmod(E,temp,temp,n)
        );
 ans
}
```

C Repeated Squaring

In diesem Kapitel stellen wir die aus der algorithmischen Zahlentheorie stammende Methode des wiederholten Quadrierens (repeated squaring method) in einer endlichen Gruppe vor. Folgendes Problem stellt sich in Gruppen, wie etwa in $(\mathbb{Z}/m\mathbb{Z}, +)$, $((\mathbb{Z}/m\mathbb{Z})^*, \cdot)$ oder $E(\mathbb{K}) : y^2 = x^3 + ax + b$: Gegeben sei ein Element b in einer endlichen Gruppe G und wir wollen die n-te Potenz $a = b^n$ berechnen. Ein naiver Ansatz dafür wäre, das Element wieder und wieder mit sich selbst zu multiplizieren. Bei großen Potenzen ist diese Variante des Rechnens ein sehr zeit- und arbeitsaufwendiges Unterfangen. Es gibt aber eine gute Möglichkeit, dies drastisch zu beschleunigen – die Methode des wiederholten Quadrierens.
Sei G eine Gruppe, $b \in G$ und $n = d_{k-1}2^{k-1} + d_{k-2}2^{k-2} + \cdots + d_1 2 + d_0$ die binäre Darstellung der zu berechneten Potenz. Es gilt $d_i \in \{0, 1\}$ und gesucht ist b^n. Wir berechnen iterativ die Potenzen b^{2^i} für natürliche Zahlen $0 \leq i \leq k-1$. Die Zweierpotenzen b^{2^i} mit $d_i = 0$ lassen wir danach sofort wieder beiseite. Übrig bleiben die Zweierpotenzen b^{2^i} mit $d_i = 1$, die zusammenmultipliziert genau das gewünschte Ergebnis

$$b^n = b^{d_{k-1}2^{k-1} + d_{k-2}2^{k-2} + \cdots + d_1 2 + d_0} = b^{d_0} b^{2d_1} b^{2^2 d_2} \cdots b^{2^{k-1} d_{k-1}}$$

liefern. Man beachte, dass die Potenzen b^{2^i} mit $d_i = 0$ gar nicht auftreten. Wir verfassen diesen Algorithmus nun in PARI für $G = (\mathbb{Z}/a\mathbb{Z})^*$.

Algorithmus C.0.1. (Repeated Squaring)

```
repsqr(b,n,m)=
{local(d,k,a);
 d=binary(n);
 k=length(d);
 a=1;
 b=b%m;
 for(i=1,k,
        if(d[i]==1,
                a=(a*b)%m
                );
        b=(b^2)%m
        );
 a
}
```

D Link-Sammlung von Implementierungen diverser Faktorisierungsalgorithmen

- http://www.crypto-world.com/FactorCode.html, Link-Sammlung vieler Faktorisierungspakete
- http://maxima.sourceforge.net/, Maxima ist ein ursprünglich am Massachusetts Institute of Technology (MIT) 1968 entwickeltes, kostenfreies Algebra-Programm
- http://www.sagemath.org/index.html, SAGE ist ein 2005 von William Stein entwickeltes, kostenfreies Algebra-Programm, das mehrere Programme, wie auch PARI/GP kombiniert.
- http://pgnfs.org/, Implementierung des GNFS von Per Leslie Jensen
- http://cado-nfs.gforge.inria.fr/, Implementierung in C/C++ des NFS von einer Gruppe um Shi Bai
- http://msieve.sourceforge.net/, Implementierung des GNFS und des quadratischen Siebs mit Selbstinitialisierung
- http://factor-by-gnfs.sourceforge.net/, Implementierung des GNFS
- http://kmgnfs.cti.gr/kmGNFS/Home.html, Implementierung des GNFS
- http://www.asahi-net.or.jp/~KC2H-MSM/cn/, mehrere Faktorisierungs-Werkzeuge von Satoshi Tomabechi
- http://arielqs.sourceforge.net/, Implementierung vom QS
- http://www.loria.fr/~zimmerma/records/ecmnet.html, Implementierung von ECM von Paul Zimmermann

E Faktorisierungszeiten im Überblick

Zahl	Probedivision	Pollard-ρ	Pollard-$(p-1)$	Elliptische Kurven
$M_{67} = 2^{67} - 1$	1m26s784ms	265ms	31ms	845ms
$M_{101} = 2^{101} - 1$	×	30s248ms	3s947ms	12s330ms
$M_{103} = 2^{103} - 1$	×	484ms	0ms	7s971ms
$M_{109} = 2^{109} - 1$	5m31s096ms	327ms	125ms	2s190ms
$M_{137} = 2^{137} - 1$	×	×	9m15s395ms	6m0s565ms
$M_{139} = 2^{139} - 1$	×	29s593ms	0ms	25s569ms
$M_{149} = 2^{149} - 1$	×	×	×	×
$M_{1039} = 2^{1039} - 1$	2s231ms	78ms	63ms	93ms
$F_6 = 2^{2^6} + 1$	140ms	31ms	16ms	624ms
$F_7 = 2^{2^7} + 1$	×	×	×	1m32s071ms
$F_8 = 2^{2^8} + 1$	×	3m6s687ms	×	1m36s807ms
$a_1 = \frac{7^{101}+2}{3 \cdot 13 \cdot 185723 \cdot 4944889067}$	×	×	31ms	2m58s543

Tabelle E.0.1: Faktorisierungszeiten im Überblick

F Faktorisierungen wichtiger Zahlen

RSA-120 = 327414555693498015751146303749141488063642403240171463406883
×
6933426671108301811973254018997006413619658631273366806730 13

RSA-576 = 3980750864240649373971255005503864911990643 6
2342526708406385189575946388957261768583317
×
4727721461074353025362230719730482246329146 9
5302097116459852171130520711256363590397527

RSA-200 = 3532461934402770121272604978198464368671197400197 6
2502364930346877612125367942320005854795652808834 9
×
7925869954478333033347085841480059687737975857364 2
1996073433034145576787281815213538140930474018546 7

RSA-768 = 3347807169895689878604416984821269081770479498371376856891
2431388982883793878002287614711652531743087737814467999489
×
3674604366679959042824463379962795263227915816434308764267
6032283815739666511279233373417143396810270092798736308917

M_{1039} = 5080711
×
5585366661993629126074920465831594496864
6527018488637648010052346319853288374753
×
2075818194644238276457048137035946951629397080073952098 81
2083870379272909032467938234314388414483488253405334476 91
1222302815832769652537609141018910524199389933410971162 43
5896206597216748116174900480365973557340925320542552368 9

$$\frac{6^{353}-1}{5} = 135095261330112651830775049635590807381121031111382732318390$$
$$846759744072165636542920143351738198057636666351316191686483$$
$$\times$$
$$720744381111301937643935864029025391613890867099707 8$$
$$170498495662717857340748450948116108762737328670417 8$$
$$679466051451768242073072242783688661390273684623521$$

$$2^{821}+2^{411}+1 = 750529374601164176649246785489326160366403 81$$
$$023147128390479077762437121481797484501905 33$$
$$\times$$
$$186319686839358535717514258381483689499331459111581549$$
$$941335271752388029836506101669417052429763321754426055$$
$$298687835774637311598798399850905850477294329024359 7$$

$$F_{11} = 319489 \times 974849 \times 167988556341760475137 \times 3560841906445833920513$$
$$\times$$
$$173462447179147555430258970864309778377421844723664084649347019 06$$
$$136357919287910885759103833040883717798381086845154642194071297 83$$
$$061341898642808260145427587085892438736855639731189488693991585 45$$
$$506611147420216132557017260564139394366945793220968665108959685 48$$
$$270538807264582855415193640191246493118254609287981573305779557 33$$
$$585049822792800909428725675915189121186227517143192297881009792 51$$
$$036035496917279912663527358783236647193154777091427745377038294 58$$
$$491891759032511093938132248604429857397165071105924446217754254 07$$
$$069130470346646436034913824417233065988341 77$$

Literatur

[1] Alpern, Dario: http://www.alpertron.com.ar/ENGLISH.HTM (zugegriffen am 01.03.2012) 39

[2] Applegate, David L.; Bixby, Robert E.; Chvátal, Vašek; Cook, William J.: *The traveling salesman problem*, Princeton University Press, 2006

[3] Agrawal, Manindra; Kayal, Neeraj; Saxena, Nitin: *PRIMES is in P*, Annals of Mathematics **160** (2004), Number 2, S. 781–793 19, 21, 25

[4] Ballinger, Ray; Rodenkirch, Mark: http://www.prothsearch.net/ (zugegriffen am 01.03.2012) 86

[5] Bourseau, Frank; Fox, Dirk; Thiel, Christoph: *Vorzüge und Grenzen des RSA-Verfahrens*, Datenschutz und Datensicherheit **26** (2002), S. 84-89

[6] Brassard, Gilles; Bratley, Paul: *Fundamentals of Algorithmics*, Prentice Hall, 1996

[7] Brent, Richard P.: *An Improved Monte Carlo Factorization Algorithm*, BIT **20** (1980), S. 176-184 30

[8] Brent, Richard P.: *Factoring Integers – an Introduction*, MSI & CECS, ANU, 2010

[9] Brent, Richard P.: *Recent Progress and Prospects for Integer Factorisation Algorithms*, Oxford University Computing Laboratory, Springer Verlag, 2000

[10] Brent, Richard P.: *Some Parallel Algorithms for Integer Factorization*, Oxford University Computing Laboratory, Springer Verlag, 1999

[11] Brent, Richard P.; Pollard, John M.: *Factorization of the Eighth Fermat Number*, Mathematics of Computation **36** (1981), Number 154, S. 627-630

[12] Buchmann, Johannes: *Faktorisierung großer Zahlen*, Spektrum der Wissenschaft **9** (1996), S. 80 ff

[13] Buhler J.P.; Lenstra, Hendrik W. Jr.; Pomerance, Carl: *Factoring integers with the number field sieve. The development of the number field sieve*, S. 50-94, Lecture Notes in Mathematics, 1554, Springer Verlag, 1993 70

[14] Bundesamt für Sicherheit in der Informationstechnik: BSI - Technische Richtlinie, 2008, https://www.bsi.bund.de/cae/servlet/contentblob/477256/publicationFile/30629/BSI-TR-02102_V1_0_pdf.pdf (zugegriffen am 01.03.2012) 24

[15] Bundschuh, Peter: *Einführung in die Zahlentheorie*, Springer Verlag, 5. Auflage, 2002

[16] Burde, Dietrich: *Cryptography. Lecture Notes*, Universität Wien, 2007, http://homepage.univie.ac.at/Dietrich.Burde/ (zugegriffen am 01.03.2012)

[17] Byrnes, Steven: *The Number Field Sieve*, Math 129 Final Paper, 2005 75

[18] Canfield, E. R.; Erdös, Paul; Pomerance, Carl: *On a problem of Oppenheim concerning „factorisatio numerorum“*, Journal of Number Theory **17** (1983), Number 1, S. 1-28 60

[19] Cohen, Henri: *A Course in Computational Algebraic Number Theory*, Springer Verlag, 1993 25, 79

[20] Contini, Scott: http://www.crypto-world.com/FactorWorld.html (zugegriffen am 01.03.2012) 85, 86

[21] Couveignes, Jean-Marc: *Computing a Square Root for the Number Field Sieve. The development of the number field sieve*, S. 95-102, Lecture Notes in Mathematics, 1554, Springer Verlag, 1993 71

[22] Costello, Craig: *Factoring Large Integers With The Number Field Sieve*, Queensland University of Technology, 2007 65, 66, 67, 71, 74

[23] Crandall, Richard; Pomerance, Carl: *Prime Numbers. A Computational Perspective*, Springer Verlag, 2nd Edition, 2000 25, 40, 43, 58, 69, 70, 76, 88

[24] Davis, James A.; Holdridge, Diane B.; Simmons Gustavus J.: Status Report on Factoring (At the Sandia National Laboratories), Advances in cryptology (1984), S. 183–215, Lecture Notes in Computer Science, Springer Verlag, 1985 57

[25] DeLeon, Melissa: *A Study of Sufficient Conditions for Hamiltonian Cycles*, Seton Hall University, 2000

[26] Deuring, Max: *Die Typen der Multiplikatorenringe elliptischer Funktionenkörper*, Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg (1941), Volume 14, S. 197-272 40

[27] Forster, Otto: *Algorithmische Zahlentheorie*, Vieweg Verlag, 1996 12, 24, 36

[28] Gabriele, Nebe: *Faktorisieren ganzer Zahlen*, RWTH Aachen, 1999

[29] Goldreich, Oded: *P, NP, and NP-completeness*, Cambridge University Press, 2010 21

[30] Gregg, John Aaron: *On Factoring Integers and Evaluating Discrete Logarithms*, Harvard College, 2003

[31] Hartmann, Peter: *Faktorisierungsalgorithmen natürlicher Zahlen*, Universität Hamburg, 2007

[32] Kellerer, Hans; Pferschy, Ulrich; Pisinger, David: *Knapsack problems*, Springer Verlag, 2004

[33] Kleinjung, Thorsten et al.: *Factorization of a 768-bit RSA modulus*, Advances in cryptology—CRYPTO 2010, S. 333–350, Lecture Notes in Computer Science, 6223, Springer Verlag, 2010 85

[34] Koblitz, Neal: *A Course In Number Theory And Cryptography*, Springer Verlag, 2nd Edition, 1991 17

[35] Kraitchik, Maurice: *Théorie des Nombres*, Gauthier-Villars, 1926 44

[36] Lehmer, D. H.; Powers, R. E.: *On Factoring Large Numbers*, Bulletin of the American Mathematical Society, **37** (1931), Number 10, S. 770–776 83

[37] Lenstra, Arjen K.: *Integer Factoring*, Designs Codes and Cryptography **19** (2000), Number 2-3, S. 101-128

[38] Lenstra, Arjen K.; Lenstra, Hendrik W. Jr.: *The development of the number field sieve*, Springer Verlag, 1993 62

[39] Lenstra, Arjen K.; Lenstra, Hendrik W. Jr.; Manasse, M.S.; Pollard, John M.: *The number field sieve*, Association for Computing Machinery (1990), S. 564-572

[40] Lenstra, Hendrik W. Jr.: *Factoring integers with elliptic curves*, Annals of Mathematics **126** (1987), Number 3, S. 649-673 37

[41] Lutzmann, Bernhard: Quadratic Number Fields that are Euclidean but not Norm-Euclidean, Universität Wien, 2009

[42] Montgomery, Peter L.: *A Survey of Modern Integer Factorization Algorithms*, CWI Quarterly **7** (1994), Number 4, S. 337–366

[43] Montgomery, Peter L.: *Speeding the Pollard and elliptic curve methods of factorization*, Mathematics of Computation **48** (1987), S. 243-264

[44] Morrison, Michael A.; Brillhart, John: *A Method of Factoring and the Factorization of F_7*, Mathematics of Computation **29** (1975), Number 129, S. 183-205 2, 50, 79, 83

[45] Müller-Stach, Stefan; Piontkowski, Jens: *Elementare und algebraische Zahlentheorie*, Vieweg Verlag, 2006 25

[46] Narkiewicz, Wladyslaw: *The development of prime number theory*, Springer Verlag, 2000

[47] Neukirch, Jürgen: *Algebraische Zahlentheorie*, Springer Verlag, 2007 75

[48] Oberfell, Jörg: *Faktorisieren mit elliptischen Kurven*, Universität Stuttgart, 2005 38

[49] Odlyzko, Andrew M.: *The future of integer factorization*, AT&T Bell Laboratories, 1995

[50] PARI/GP, version `2.3.4`, 2008, `http://pari.math.u-bordeaux.fr/`

[51] Pollard, John M.: *A Monte Carlo method for factorization*, Nordisk Tidskrift for Informationsbehandling **15** (1975), S. 331-334

[52] Pollard, John M.: *Factoring with cubic integers. The development of the number field sieve*, S. 4-10, Lecture Notes in Mathematics, 1554, Springer Verlag, 1993

[53] Pollard, John M.: *Theorems of Factorization and Primality Testing*, Proceedings of the Cambridge Philosophical Society, 1974

[54] Pomerance, Carl: *A Tale Of Two Sieves*, Notices of AMS **43** (1996), Number 12, S. 1473-1485

[55] Pomerance, Carl: *Smooth Numbers and the Quadratic Sieve*, MRSI, 2000

[56] Pomerance, Carl: *The Quadratic Sieve Factoring Algorithm*, University of Georgia, Springer Verlag, 1998 51, 60

[57] Pope, Steven: *On Problems of Cryptography. Primality Testing and Factoring Integers*, Queen's University, 2000

[58] Ram Pedaprolu Murty, Maruti: *Artin's conjecture for primitive roots*, Mathematical Intelligencer, McGill University, 1988

[59] Remmert, Reinhold; Ullrich, Peter: *Elementare Zahlentheorie*, Birkhäuser Verlag, 3. Auflage, 2008 9

[60] Ribenboim, Paulo: *Die Welt der Primzahlen. Geheimnisse und Rekorde*, Springer Verlag, 2. Auflage, 2011

[61] Rivest, Ronald L.; Shamir, Adi; Adleman, Leonard: *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the Association for Computing Machinery **21** (1978), Number 2, S. 120-126 23

[62] Rothe, Jörg: *Komplexitätstheorie und Kryptologie*, Springer Verlag, 2008

[63] Schmidt-Samoa, Katja: *Das Number Field Sieve*, Universität Kaiserslautern, 2002 75

[64] Schneier, Bruce: http://www.schneier.com/blog/archives/2007/05/307digit_number.html (zugegriffen am 01.03.2012)

[65] Shanks, Daniel: *SQUFOF notes*, Handgeschrieben 1975-1985, neu in Latex verfasst von Stephen McMath, 2004 79

[66] Shor, Peter: *Polynomial Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM Journal on Computing **26** (1997), Number 5, S. 1484–1509

[67] Smith, Frederick J.: *A Brief History of Factorization Techniques*, University of Washington, 2006

[68] Steward, Ian; Tall, David: *Algebraic Number Theory and Fermat's Last Theorem*, A K Peters, Third Edition, 2002 75

[69] Stöffler, Chrstian: *Faktorisieren mit dem General Number Field Sieve*, Universität Mannheim, 2007 71

[70] von Alexandria, Euklid: *Die Elemente*, http://www.opera-platonis.de/euklid/ (zugegriffen am 01.03.2012) 3

[71] Vandersypen, Lieven M. K.; Steffen, Matthias; Breyta, Gregory; Yannoni, Costantino S.; Sherwood, Mark H.; Chuang, Isaac L.: *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*, Nature **414** (2001), S. 883-887 88

[72] Wagstaff, Sam: http://homes.cerias.purdue.edu/~ssw/cun/ (zugegriffen am 01.03.2012) 86

[73] Washington, Lawrence C.: *Elliptic Curves. Number Theory And Cryptography*, Chapman & Hall/CRC Verlag, 2003 13, 16, 17

[74] Wegener, Ingo: *Komplexitätstheorie*, Springer Verlag, 2003 18

[75] West, Andrew Granville: *Bound Optimazation For Parallel Quadratic Sieving Using Large Prime Variations*, Washington and Lee University, 2007

[76] Zayer, Jörg: *Faktorisieren mit dem Number Field Sieve*, Universität des Saarlandes, 1995

[77] Zimmermann, Paul: http://www.loria.fr/~zimmerma/records/ (zugegriffen am 01.03.2012) 85

Zusammenfassung

Die vorliegende Diplomarbeit behandelt das Faktorisierungsproblem, also von der Berechnung eines Teilers einer zusammengesetzten ganzen Zahl, und primär den Methoden, die sowohl historisch als auch aktuell dafür zur Anwendung kommen. Zuvor führen wir alle mathematischen Grundlagen, die wir zur Vorstellung der Faktorisierungsalgorithmen benötigen, ein. Die im Hauptteil beinhalteten Methoden zur Teilerberechnung, deren Funktionsweisen wir detailliert vorstellen, sind die Probedivision, Pollards ρ- und $(p-1)$-Methode, die Elliptic Curve Method, die Methode nach Fermat, nach Legendre, der Kettenbruchalgorithmus, das quadratische Sieb und das Zahlkörpersieb. Für ein besseres Verständnis geben wir einige der Algorithmen als funktionstüchtigen Code des Algebra-Programms PARI/GP an und testen diese auf ihre Geschwindigkeit und Eigenschaften. Zwei Fragen schenken wir bei der Beschreibung dieser Methoden besondere Beachtung:

- Wie schnell arbeitet der jeweilige Algorithmus?
- Spielen Charakteristika der zu faktorisierenden Zahlen eine Rolle?

Diese Fragestellungen sind im Besonderen im Anwendungsbereich des Faktorisierungsproblems, dem RSA-Verfahren in der Verschlüsselungstheorie, von zentraler Bedeutung, da die Sicherheit des Verfahrens auf deren Komplexität beruht.
Danach nehmen wir aktuelle Faktorisierungsprogramme praktisch unter die Lupe, testen also mittels ausgewählter Zahlen ihre Faktorisierungsgeschwindigkeit. Abschließend erfolgt wir eine Zeitreise durch die Faktorisierungsgeschichte, eine Analyse des gegenwärtigen Forschungsstandes und ein Blick in die Zukunft der Zahlenfaktorisierung.

Abstract

This diploma thesis is about the factorization problem, i.e. to compute a factor of a composite integer, and primary the historical and current methods to do this. First of all we discuss the mathematical background we need to present the factorization algorithms. The main part of this thesis, the detailed introductions of the factorization algorithms, will cover the trial division algorithm, Pollard's ρ- and $(p-1)$-method, the elliptic curve method, Fermat's factorization method, Legendre's factorization method, the continued fraction method, the quadratic sieve and the number field sieve. For a better understanding we specify for some of the methods a working source code for the algebra software PARI/GP, which we will check for speed and properties. The two following questions will cause our special interest:

- How fast terminates the algorithm?
- Do the characteristics of the factoring numbers play a role?

These questions are in particular important for the RSA-algorithm in the cryptography, where the factorization problem plays the central part for the security.
Afterwards we test some of the current factorization softwares for their speed and finally we go on a time travel of the factorization history, analyze the state of the art and look into the future of the integer factorization.

Lebenslauf

Persönliche Daten

- Name: Manuel Wolf
- E-Mail: manu10000@gmx.net
- Geburtsdaten: 22.01.1985, Wien
- Wohnort: Wien
- Staatsbürgerschaft: Österreich

Schulische Ausbildung

- 1991-1995 Volksschule Stiftgasse, Wien
- 1995-2003 BG/BRG 6 Rahlgasse, Wien
- 2003 Matura mit gutem Erfolg bestanden

Zivildienst

- 2003-2004 Kuratorium Wiener Pensionisten-Wohnhäuser, Wien

Universitäre Ausbildung

- 2005-2008 1. Abschnitt Diplomstudium Mathematik, Universität Wien
- 2008-2012 2. Abschnitt Diplomstudium Mathematik, Universität Wien