Die Berechnung der Punktanzahl von elliptischen Kurven über endlichen Primkörpern

Diplomarbeit

von

Volker Müller

nach einem Thema von Prof. Dr. Buchmann,
am Fachbereich 14 - Informatik -
der Universität des Saarlandes

Saarbrücken, den 8. Oktober 1991

Mein besonderer Dank gilt Herrn Prof. Buchmann für dieses sehr interessante Thema sowie die Betreuung während der Ausarbeitung dieser Arbeit. Seine Kommentare und Rückfragen haben mich immer wieder dazu gebracht, einzelne Teile neu zu überdenken und besser zu verstehen.

Weiterhin möchte ich mich bei Ingrid Biehl, Stefan Düllmann, Oliver van Sprang und Ralf Roth für die immer gute Arbeitsatmosphäre und viele hilfreiche Diskussionen bedanken.

Last but not least gilt mein Dank auch meinen Eltern und meinem Bruder, die mich in meinem Studium bestens unterstützt haben und die bei Mißerfolgen häufig meine schlechte Laune ausbaden mußten.

Inhaltsverzeichnis

Einleitung

In dieser Arbeit beschäftigen wir uns mit dem Problem der Berechnung der Punktanzahl einer elliptischen Kurve über einem endlichen Primkörper.

Dieses Problem ist nicht nur für Zahlentheoretiker sehr interessant, sondern hat in letzter Zeit auch große Aufmerksamkeit unter den Kryptographie-Fachleuten erregt. Der Gebrauch von elliptische Kurven in der Public-Key-Kryptographie wurden sowohl von Neal Koblitz ([Ko87]) als auch von Victor Miller ([Mi86]) vorgeschlagen. Die Sicherheit dieser Kryptosysteme basiert auf dem Problem des diskreten Logarithmus in der Punktgruppe einer elliptischen Kurve. Die besten Algorithmen zur Lösung dieses Problems für eine beliebige elliptische Kurve ([Od86]) besitzen eine Laufzeit proportional zum größten Primfaktor der Gruppenordnung. Um also die Sicherheit eines solchen Systems zu garantieren, sollte man die Gruppenordnung der verwendeten elliptischen Kurve berechnen und faktorisieren. Da die in der Praxis benutzten Primkörper von der Größenordnung 10^{40} sind, ist die Faktorisierung einer berechneten Gruppenordnung einfach. Wir stellen in dieser Arbeit zwei Verfahren zur Bestimmung dieser Gruppenordnung vor und analysieren ihre Laufzeit.

Die Arbeit gliedert sich wie folgt:

- Im ersten Kapitel wiederholen wir einige Begriffe aus der Algebra und definieren den Begriff "elliptische Kurve". Anschließend stellen wir einige grundlegende Eigenschaften einer solchen elliptischen Kurve vor. Danach beschreiben wir einen ersten, allerdings ziemlich trivialen Algorithmus zur Bestimmung der Gruppenordnung.
- Im folgenden zweiten Kapitel stellen wir einen Grundalgorithmus zur Bestimmung der Gruppenordnung vor. Dieser Algorithmus basiert auf einer allgemeinen Idee von Daniel Shanks. Gleichzeitig beweisen wir die Korrektheit dieses Algorithmus.
- Im dritten Kapitel bestimmen wir die erwartete Laufzeit des Shanks Algorithmus und geben eine obere Schranke für den Platzbedarf des Shanks Algorithmus an.
- Im vierten Kapitel beschreiben wir unsere Implementierung des Shanks Algorithmus. Diese Implementierung wurde dazu benutzt, die im letzten Kapitel angege-

benen Resultate zu berechnen.

- Das fünfte Kapitel stellt einen weiteren Algorithmus, den Algorithmus von Schoof, zur Lösung unseres Problems vor. Gleichzeitig erläutern wir die theoretischen Grundlagen des Schoof-Algorithmus und beweisen damit seine Korrektheit.

- Im sechsten Kapitel wird die Laufzeit des Schoof-Algorithmus bestimmt.

- Um eine praktische Verbesserung des Shanks Algorithmus zu erreichen, kombinieren wir Ideen aus dem Shanks und dem Schoof-Algorithmus. Diese Kombination, die zuerst in [BuMü91] vorgestellt wurde, beschreiben wir im siebten Kapitel genauer.

- Als Anwendung der vorher vorgestellten Algorithmen zur Berechnung der Gruppenordnung stellen wir einen Algorithmus zur Bestimmung des Zerlegungstyps der Punktgruppe einer elliptischen Kurve vor. Anschließend beweisen wir die Korrektheit dieses Algorithmus und bestimmen die erwartete Laufzeit sowie den Platzbedarf des Algorithmus.

- Im letzten Kapitel geben wir einige praktische Laufzeitresultate an und stellen praktisch berechnete Ergebnisse vor.

Kapitel 1

Einführung

Im ersten Kapitel werden wir grundlegende Begriffe aus der Theorie der Körper wiederholen. Anschließend werden wir den Begriff "elliptische Kurve" definieren und einige grundlegende Eigenschaften einer solchen Kurve vorstellen. Am Ende des Kapitels wird ein erster – allerdings trivialer – Algorithmus zur Lösung unseres Problems der Bestimmung der Ordnung der Punktgruppe einer elliptischen Kurve über einem endlichen Primkörper angegeben.

1.1 Mathematische Einführung

Als erstes wiederholen wir einige grundlegende Begriffe und Sätze aus der Körpertheorie. Dabei verzichten wir auf die notwendigen Beweise zu diesen Aussagen. Diese und weiterführende Informationen findet man beispielsweise in dem Buch von Hungerford [Hun74].

- Ein **Körper** ist eine Menge K mit zwei Verknüpfungen '+' und '·', für die $(K, +)$ und $(K - \{0\}, \cdot)$ jeweils abelsche Gruppen sind und für die die Distributivität der beiden Verknüpfungen gilt.
- Ein **endlicher Körper** ist ein Körper mit nur endlich vielen Elementen.
- Die **Charakteristik** eines Körpers ist die kleinste positive Zahl p mit $p \cdot 1 = 0$ bzw. Null, falls keine solche Zahl existiert. Die Charakteristik eines Körpers ist entweder Null oder eine Primzahl.
- Der **Primkörper der Charakteristik** p ist der Durchschnitt aller Körper der Charakteristik p. Falls p eine Primzahl ist, ist der Primkörper der Charakteristik p isomorph zum Körper $\mathbb{F}_p := \mathbb{Z} \big/ p\mathbb{Z}$.

- Ein Körper K heißt **algebraisch abgeschlossen**, wenn jedes Polynom mit Koeffizienten aus K über K ganz in Linearfaktoren zerfällt.
- Zu jedem Körper K existiert ein Oberkörper $\overline{K}$, der algebraisch abgeschlossen ist. Alle algebraisch abgeschlossenen Oberkörper von K sind zueinander isomorph. Daher spricht man von dem **algebraischen Abschluß** von K.

Nach diesen grundlegenden Begriffen aus der Körpertheorie wenden wir uns nun den elliptischen Kurven zu. Zuerst werden wir die Definition einer elliptischen Kurve vorstellen. Anschließend werden dann einige grundlegende Sätze aus der Theorie der elliptischen Kurven vorgestellt.

Sei dazu im folgenden K ein Körper der Charakteristik $p > 3$ und sei $\overline{K}$ sein algebraischer Abschluß.

Definition 1.1 Seien $a, b \in K$ mit der Eigenschaft $4a^3 + 27b^2 \neq 0$.

Unter einer **elliptischen Kurve über dem Körper** K versteht man die Gesamtheit aller Lösungen aus $\overline{K} \times \overline{K}$ einer Gleichung der Form

$$E: \qquad y^2 = x^3 + ax + b \tag{1.1}$$

einschließlich der Lösung $\mathcal{O} := (\infty, \infty)$.

Die Menge aller Punkte auf E, d.h. die Menge

$$\{\ (x, y) \in \overline{K}^2\ ;\ y^2 = x^3 + ax + b\ \} \quad \cup \quad \{\ \mathcal{O}\ \},$$

bezeichnet man als $E(\overline{K})$.

Die Menge aller **K-rationalen Punkte** ist definiert als

$$E(K) = \{\ (x, y) \in K^2\ ;\ y^2 = x^3 + ax + b\ \} \quad \cup \quad \{\ \mathcal{O}\ \}. \tag{1.2}$$

Bemerkung 1.1 *Die Kurvengleichung (1.1) wird in der Literatur als* **kurze Weierstraß-Normalform** *bezeichnet. Die kurze Weierstraß-Normalform kann aus einer allgemeinen Kurvengleichung, der sogenannten* **Weierstraß-Gleichung**, *durch isomorphe Umformungen hergeleitet werden. Die allgemeine Weierstraß-Gleichung beschreibt eine elliptische Kurve in jedem beliebigen Körper, wohingegen die kurze Weierstraß-Normalform nur in Körpern der Charakteristik ungleich* 2 *und* 3 *eine elliptische Kurve korrekt beschreibt. Da wir elliptische Kurven nur über Primkörpern großer Charakteristik betrachten, wählen wir statt der allgemeinen Kurvengleichung die einfachere kurze Weierstraß-Normalform. Mehr Informationen über die allgemeine Weierstraß-Gleichungen findet man in [Si86].*

Bemerkung 1.2 *Man kann zeigen, daß die Bedingung $4a^3 + 27b^2 \neq 0$ aus Definition 1.1 äquivalent dazu ist, daß die rechte Seite $x^3 + ax + b$ der Kurvengleichung keine mehrfache Nullstelle in K besitzt. Eine Kurve, die diese Bedingung erfüllt, bezeichnet man als* **nicht-singulär**. *Bei elliptischen Kurven setzt man damit immer voraus, daß sie nicht-singulär sind.*

Es stellte sich schon früh heraus, daß die Gesamtheit aller K-rationalen Punkte einer elliptischen Kurve nicht nur eine einfache Menge bildet, sondern zusätzlich noch eine Struktur besitzt. Diese Struktur läßt sich durch einfache Formeln beschreiben, so daß es leicht möglich ist, in dieser Punktmenge praktisch zu rechnen. Dies wird im folgenden Satz erläutert.

Satz 1.1 *Sei K ein Körper und $a, b \in K$. Dann bildet die Menge der K-rationalen Punkte einer elliptischen Kurve $E: \ y^2 = x^3 + ax + b$ eine abelsche (i.a. additiv geschriebene) Gruppe.*

Die Addition wird dabei durch folgendes Gesetz gegeben:

- *Das neutrale Element der Gruppe $E(K)$ ist der Punkt $\mathcal{O}$.*
- *Zu einem Punkt $\mathcal{O} \neq P = (x, y) \in E(K)$ definieren wir als negativen Punkt*
$$-P \ := \ (x, -y) \quad \in E(K).$$
- *Für Punkte P_1 und P_2 mit $P_1 = -P_2$ wird die Summe als $P_1 + P_2 := \mathcal{O}$ definiert.*
- *Für Punkte $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ mit $P_1 \neq -P_2$ wird die Summe gegeben als $P_1 + P_2 = (x_3, y_3)$, wobei*
$$\begin{aligned} x_3 &= -x_1 - x_2 + \lambda^2 \\ y_3 &= -y_1 + \lambda \cdot (x_1 - x_3) \end{aligned}$$
mit
$$\begin{aligned} \lambda &= \frac{y_1 - y_2}{x_1 - x_2} \quad \text{falls } P_1 \neq \pm P_2. \\ \lambda &= \frac{3x_1^2 + a}{2y_1} \quad \text{falls } P_1 = P_2. \end{aligned}$$

Beweis: Betrachten wir die geometrische Bedeutung der Addition zweier Punkte, die in der folgenden Bemerkung 1.3 beschrieben wird, so erkennen wir leicht, daß alle Gruppenbedingungen außer der Assoziativität trivialerweise erfüllt sind. Die Assoziativität der Punktaddition kann mit großem Rechenaufwand durch Einsetzen der Koordinaten von drei beliebigen Punkten in die Additionsgleichungen nachgerechnet werden. Eine genaue Ausführung des Beweises findet man in dem Buch von Silverman [Si86]. ■

Bemerkung 1.3 *Geometrisch besitzt die Addition zweier Punkte einer elliptischen Kurve folgende Bedeutung:*

Seien P_1 *und* $P_2 \neq \pm P_1$ *zwei verschiedene Punkte einer elliptischen Kurve, die beide ungleich dem Nullpunkt* $\mathcal{O}$ *sind. Um die Summe der beiden Punkte zu bestimmen, legen wir zuerst eine Gerade durch die zwei Punkte* P_1 *und* P_2. *Diese Gerade schneidet die elliptische Kurve mit Sicherheit in einem dritten Punkt* Q. *Dann erhalten wir die Summe der beiden Punkte* P_1 *und* P_2 *durch Spiegelung des Punktes* Q *an der x-Achse, d.h.* $P_1 + P_2 = -Q$.

Zur Berechnung von $2 \cdot P_1$ *verwenden wir anstatt der Geraden die Tangente an die elliptische Kurve im Punkt* P_1. *Diese Tangente schneidet die Kurve ebenfalls in einem weiteren Punkt* Q' *und* $2 \cdot P_1$ *erhalten wir wiederum durch Spiegelung des Punktes* Q' *an der x-Achse, d.h.* $2 \cdot P_1 = -Q'$.

Beispiel 1.1 *In der folgenden Abbildung wird die Kurve*

$$E: \quad y^2 = x^3 - 10x + 10$$

über dem Körper der reellen Zahlen dargestellt. Außerdem werden die beiden interessanten Fälle der Punktaddition illustriert. Als Beispiel für die Punktaddition zweier verschiedener Punkte wählen wir dabei $P = (-3, \sqrt{13})$ *und* $Q = (1, -1)$. *Als Summe erhalten wir dann*

$$P + Q = \left(2 + \left(\frac{1+\sqrt{13}}{4}\right)^2, \frac{5+5\sqrt{13}}{4} + \left(\frac{1+\sqrt{13}}{4}\right)^3 - \sqrt{13}\right).$$

Zur graphischen Darstellung der Verdoppelung eines Punktes, d.h. der Berechnung von $2 \cdot H = H + H$ *für einen Punkt* H, *wählen wir den Punkt*

$$H \approx (-2.209736791, -4.615989934)$$

und erhalten dann als Summe

$$2 \cdot H = (6, \sqrt{166}).$$

Sei ab jetzt der Körper K der Primkörper der Charakteristik p für eine feste Primzahl $p > 3$, d.h. $K = \mathbb{Z}/p\mathbb{Z}$. In Analogie zur Literatur werden wir diesen Körper im folgenden auch mit $\mathbb{F}_p$ bezeichnen.

Unser Problem besteht darin, die Anzahl der $\mathbb{F}_p$-rationalen Punkte einer beliebigen elliptischen Kurve E über diesem Körper $\mathbb{F}_p$ zu bestimmen. Dabei sei die Charakteristik p des Körpers $\mathbb{F}_p$ eine möglichst "große" Zahl. In der Praxis bedeutet "groß" dabei, daß p von der Größenordnung 10^{25} ist. Folgendes Lemma besagt, daß die Ordnung $\#E(\mathbb{F}_p)$ der Gruppe $E(\mathbb{F}_p)$ immer endlich ist, d.h. daß es überhaupt Sinn macht, sich mit diesem Problem zu beschäftigen.

Lemma 1.1 *Falls K ein endlicher Körper ist, so ist die Ordnung der Punktgruppe $E(K)$ einer elliptischen Kurve E über dem Körper K endlich.*

Beweis: Das Lemma folgt direkt aus der Tatsache, daß es wegen der Endlichkeit von K maximal $(\#K)^2$ viele Lösungen $(x, y) \in K^2$ der Gleichung $y^2 = x^3 + ax + b$ geben kann. Zusätzlich zu den so gebildeten Punkten enthält die Gruppe $E(K)$ nur noch einen weiteren Punkt, den Nullpunkt $\mathcal{O}$. Also kann die Punktgruppe der K-rationalen Punkte höchstens $(\#K)^2 + 1$, also endlich viele, Punkte enthalten. ■

Dieses Lemma liefert eine recht unscharfe obere Abschätzung für die Ordnung der Punktgruppe der $\mathbb{F}_p$-rationalen Punkte einer elliptischen Kurve. Der folgende berühmte Satz von Hasse liefert eine deutlich bessere Schranke für die Ordnung der Punktgruppe einer elliptischen Kurve über einem endlichen Körper.

Satz 1.2 (Satz von Hasse)
Sei E eine elliptische Kurve über dem endlichen Körper $\mathbb{F}_q$ der Charakteristik p. Dann gilt für die Ordnung der Punktgruppe der $\mathbb{F}_q$-rationalen Punkte

$$|\,\#E(\mathbb{F}_q) - q - 1\,| \;\leq\; 2\sqrt{q},$$

d.h.

$$q + 1 - 2\sqrt{q} \;\leq\; \#E(\mathbb{F}_q) \;\leq\; q + 1 + 2\sqrt{q}. \tag{1.3}$$

Beweis: [Si86, Th. 1.1, Seite 131] ■

Bemerkung 1.4 *Das Intervall*

$$[\,q + 1 - 2\sqrt{q}, q + 1 + 2\sqrt{q}\,],$$

in dem nach diesem Satz die Ordnung der Punktgruppe $E(\mathbb{F}_q)$ liegen muß, bezeichnen wir im folgenden häufig als das **Hasse-Intervall**.

In dieser Arbeit benötigen wir nur die Spezialisierung des Satzes von Hasse auf den Fall, daß der endliche Körper der Primkörper der Charakteristik p ist, da wir nur die Ordnung der $\mathbb{F}_p$-rationalen Punkte einer elliptischen Kurve E über dem Primkörper $\mathbb{F}_p$ berechnen wollen. Dann lautet die Bedingung (1.3)

$$p+1-2\sqrt{p} \;\leq\; \#E(\mathbb{F}_p) \;\leq\; p+1+2\sqrt{p}.$$

Diese Schranken für die Gruppenordnung werden wir im folgenden noch häufig benötigen.

1.2 Ein erster Algorithmus zur Bestimmung der Gruppenordnung

In diesem Abschnitt stellen wir einen ersten einfachen Algorithmus zur Bestimmung der Ordnung der Punktgruppe $E(\mathbb{F}_p)$ einer elliptischen Kurve E über dem Primkörper $\mathbb{F}_p$ vor. Dieser Algorithmus leitet sich direkt aus der Definition eines Punktes einer elliptischen Kurve her. Es wird sich allerdings spätestens bei der Laufzeitanalyse herausstellen, daß dieser Algorithmus in der Praxis nur für Primkörper mit kleiner Charakteristik anwendbar ist. Der Algorithmus basiert auf folgendem Lemma.

Lemma 1.2 *Sei $E: y^2 = x^3 + ax + b$ die Gleichung einer elliptische Kurve über dem Primkörper $\mathbb{F}_p$. Dann gilt für die Ordnung der Punktgruppe $E(\mathbb{F}_p)$*

$$\#E(\mathbb{F}_p) \quad = \quad p+1+\sum_{x\in\mathbb{F}_p}\left(\frac{x^3+ax+b}{p}\right),$$

wobei $\left(\frac{c}{p}\right)$ für das Legendre-Symbol c über p steht.

Beweis: Die Punktgruppe $E(\mathbb{F}_p)$ der elliptischen Kurve E besteht nach Definition aus allen Punkten der Form $(x,y) \in \mathbb{F}_p^2$, die die Gleichung $y^2 = x^3 + ax + b$ erfüllen und einem weiteren Punkt $\mathcal{O}$.

Sei nun $x \in \mathbb{F}_p$ beliebig. Dann gibt es folgende Möglichkeiten für den Wert $d = x^3+ax+b$:

1. $d = 0$: dann ist $(x, 0)$ ein Punkt aus $E(\mathbb{F}_p)$.
2. $d \neq 0, d \equiv y^2 \bmod p$ Quadrat: dann sind (x, y) und $(x, -y)$ Punkte aus $E(\mathbb{F}_p)$.
3. $d \neq 0, d$ kein Quadrat: es existiert kein Punkt in $E(\mathbb{F}_p)$ mit x-Koordinate x.

Berechnen wir zu diesen Fällen den Wert $\left(\frac{x^3+ax+b}{p}\right)+1$, so erhalten wir jeweils

1. $\left(\frac{0}{p}\right)+1=0+1=1$
2. $\left(\frac{d}{p}\right)+1=1+1=2$
3. $\left(\frac{d}{p}\right)+1=-1+1=0$,

d.h. wir zählen exakt die Anzahl der Punkte aus $E(\mathbb{F}_p)$ mit x-Koordinate x. Bilden wir die Summe aller solcher Werte für alle $x \in \mathbb{F}_p$ und addieren noch eine Eins für den Punkt $\mathcal{O}$, so erhalten wir die Gruppenordnung $\#E(\mathbb{F}_p)$. Also gilt

$$\#E(\mathbb{F}_p) \quad = \quad 1+\sum_{x\in\mathbb{F}_p}\left(\left(\frac{x^3+ax+b}{p}\right)+1\right).$$

Durch Herausziehen der Eins aus der Summe erhalten wir hieraus die Behauptung des Lemmas. ■

Wir formulieren nun den Algorithmus:

Algorithmus 1.1 (Charaktersummen-Algorithmus)

Eingabe: *elliptische Kurve* $E: y^2=x^3+ax+b$ *über Primkörper* $\mathbb{F}_p$.
Ausgabe: *Gruppenordnung* $\#E(\mathbb{F}_p)$.

1. *Setze* $e=p+1$.
2. *Für alle* $x=0$ *bis* $p-1$ *setze* $e=e+\left(\frac{x^3+ax+b}{p}\right)$.
3. *Gib Ergebnis* e *zurück.*

Dieser Algorithmus 1.1 besitzt den Vorteil, daß er sehr einfach und seine Korrektheit direkt verständlich ist. Allerdings wird dies durch den gravierenden Nachteil einer schlechten Zeit-Komplexität aufgehoben. Daher ist Algorithmus 1.1 in der Praxis nur für Primkörper mit kleiner Charakteristik anwendbar. Im folgenden Lemma beschreiben wir die Zeit-Komplexität des Algorithmus genauer.

Lemma 1.3 *Algorithmus 1.1 benötigt* $O(p\cdot\log(p)^2)$ *Bitoperationen zur Berechnung der Anzahl der* $\mathbb{F}_p$*-rationalen Punkte einer elliptischen Kurve* E *über dem Primkörper* $\mathbb{F}_p$.

Beweis: Zum Beweis des Lemmas beachte man, daß die Berechnung des Legendre-Symbols die Laufzeit $O(\log(p)^2)$ besitzt (das Legendre-Symbol wird noch genauer im dritten Abschnitt betrachtet, dort findet man auch eine Rechtfertigung für diese Behauptung). Damit folgt die Behauptung des Lemmas sofort, da wir für alle $x \in \mathbb{F}_p$ jeweils ein Legendre-Symbol auswerten müssen. Alle anderen Operationen werden offensichtlich von dieser Zeit dominiert. ∎

Kapitel 2

Der Algorithmus von Shanks

2.1 Die Idee

Daniel Shanks stellte 1970 in [Sh70] einen Algorithmus zur Bestimmung der Klassenzahl eines imaginär quadratischen Zahlkörpers vor. Es hat sich schnell gezeigt, daß das Prinzip dieses sogenannten **Babystep-Giantstep Algorithmus** auch zur Bestimmung der Ordnung anderer Gruppen verwendet werden kann. Der Algorithmus basiert auf der folgenden einfachen Beobachtung.

Satz 2.1 (Satz von Lagrange)
Sei x ein Element der Gruppe G. Dann teilt die Ordnung des Elementes x die Gruppenordnung $\#G$.

Beweis: [Hun74, Kor. 4.6, Seite 39] ■

Sei im folgenden $\mathbb{F}_p$ der Primkörper der Charakteristik p und sei die Gruppe G die Gruppe der $\mathbb{F}_p$ -rationalen Punkte einer elliptischen Kurve E über dem Körper $\mathbb{F}_p$. Wir beschreiben nun die Anwendung der Babystep-Giantstep Idee von Shanks zur Berechnung der Ordnung der Punktgruppe $E(\mathbb{F}_p)$.

Dazu wählen wir einen zufälligen Punkt $P \in E(\mathbb{F}_p)$ aus und bestimmen seine Ordnung. Dann kann nach Satz 2.1 die Gruppenordnung nur noch ein Vielfaches der Punktordnung sein. Dies schränkt die Gruppenordnung $\#E(\mathbb{F}_p)$ auf bestimmte Werte ein. Außerdem wissen wir nach dem Satz von Hasse 1.2, daß die Gruppenordnung im Hasse-Intervall liegen muß. Ist die Ordnung des zufällig gewählten Punktes P groß "genug", dann liegt nur ein Vielfaches der Punktordnung in dem Hasse-Intervall und die Gruppenordnung ist damit eindeutig bestimmt. Damit haben wir das Problem der Bestimmung der Gruppenordnung $\#E(\mathbb{F}_p)$ verlagert auf die Bestimmung von Vielfachen der Ordnung eines Punktes P im Hasse-Intervall.

Es stellt sich nun natürlich die Frage, wie wir Vielfache der Punktordnung eines Punktes P in einem vorgegebenen Intervall möglichst effizient bestimmen können. Eine sicherlich sehr ineffiziente Lösung erhalten wir, wenn wir alle Zahlen des Intervalls sukzessive testen. Dies führt in unserem Fall zu $O(\sqrt{p}\,)$ vielen Tests.

Bei Anwendung von Shanks' Babystep-Giantstep Idee zerlegen wir das Hasse-Intervall in gleichmäßige Teilintervalle der Länge k. Zuerst berechnen wir dann alle Punkte $i \cdot P$ für $i = 0, \ldots, k-1$ und speichern diese mit dem zugehörigen Index i in einer Tabelle. Diese ersten Schritte nennt Shanks die **Babysteps**, da sich der Laufindex i nur in Einerschritten ändert. Anschließend berechnen wir für alle Grenzen x der Teilintervalle den Punkt $x \cdot P$. Liegt nun in dem Teilintervall unterhalb des Zerlegungspunktes x ein Vielfaches l der Ordnung von P, so ist $l \cdot P = \mathcal{O}$ und damit ist $x \cdot P = (l+h) \cdot P = h \cdot P$ in der Tabelle gespeichert, denn h muß dann kleiner sein als die Länge k der Teilintervalle. Mit Hilfe des abgespeicherten Index läßt sich das Vielfache der Punktordnung leicht berechnen. Überprüfen wir also nach Berechnung von $x \cdot P$, ob dieser Punkt schon in der Tabelle vorkommt, so finden wir ein Vielfaches der Punktordnung, wenn dies in dem Teilintervall unterhalb von x liegt. Da wir diese Tests nur an den Zerlegungspunkten durchführen, gehen wir — anschaulich gesprochen — in Schritten der Größe k durch das vorgegebene Intervall. Daher werden diese Schritte auch als **Giantsteps** bezeichnet.

Man sieht leicht ein, daß dieses Verfahren optimal ist, wenn die Anzahl der Babysteps gleich der Anzahl der Giantsteps ist. Dies erreichen wir, wenn wir k größenordnungsmäßig als Quadratwurzel aus der Intervallänge wählen. In unserem Fall erhalten wir also $k \approx 2\sqrt[4]{p}$ (das Hasse-Intervall besitzt die Länge $4\sqrt{p}$).

Es bleibt noch anzumerken, daß dieser Algorithmus nur dann eine Lösung findet, wenn mindestens ein Punkt der elliptischen Kurve eine genügend große Ordnung besitzt. Gibt es einen solchen Punkt nicht (was für spezielle Kurven zutrifft), so müssen wir mit Hilfe des Zerlegungstyps der Punktgruppe weitere Information berechnen und damit dann die Gruppenordnung bestimmen. Weitere Informationen über die Berechnung und Anwendung dieser Information findet man im Abschnitt über die Korrektheit des Algorithmus.

2.2 Der Algorithmus

Nachfolgend bringen wir eine genaue Beschreibung des Shanks Algorithmus für elliptische Kurven. Der Algorithmus gliedert sich in zwei Teile:

1. In den Schritten 1 bis 6 wählen wir zufällig einen Punkt P und bestimmen Vielfache seiner Ordnung im Hasse-Intervall. Dies geschieht mit dem oben schon kurz

angesprochenen Babystep-Giantstep Verfahren. Die Babysteps werden dabei in Schritt 2, die Giantsteps in Schritt 4 durchgeführt. Die Tests, ob diese Information schon ausreicht, um die Gruppenordnung zu bestimmen, werden in den darauffolgenden Schritten 5 und 6 erledigt. Im folgenden Abschnitt über die Korrektheit des Algorithmus werden wir diesen Teil — d.h. die Schritte 1 bis 6 — häufig auch als den *ersten Teil des Algorithmus* bezeichnen.

2. Im *zweiten Teil des Algorithmus* — d.h. in den Schritten 7 bis 11 — bestimmen wir die Ordnung der Punktgruppen von elliptischen Kurven, bei denen es keinen Punkt ausreichend großer Ordnung gibt. Dazu bestimmen wir in den Schritten 8 und 9 zuerst die Ordnung eines weiteren zufälligen Punktes P' in der Gruppe $E(\mathbb{F}_p)$ und dann in Schritt 10 die Ordnung von P' in der Faktorgruppe $E(\mathbb{F}_p)/\langle P\rangle$, wobei P ein im ersten Teil bestimmter Punkt ist. Dabei verwenden wir wiederum eine Babystep-Giantstep Strategie. Die Babysteps finden nun in Schritt 10.2, die Giantsteps in Schritt 10.4 statt. Zuletzt testen wir in Schritt 11 noch, ob nun die Gruppenordnung eindeutig bestimmt ist, ansonsten gehen wir zum Anfang des zweiten Teils und wählen einen anderen zufälligen Punkt P'.

Nachfolgend nun eine genaue Beschreibung des Algorithmus. Im folgenden Abschnitt werden wir uns dann mit der Korrektheit der einzelnen Schritte beschäftigen.

Algorithmus 2.1 (Shanks Algorithmus)

Eingabe: *elliptische Kurve E über dem Primkörper $\mathbb{F}_p$, $p \geq 37$.*
Ausgabe: *Gruppenordnung $\#E(\mathbb{F}_p)$.*

Schritt 1: *Wähle einen zufälligen Punkt P aus $E(\mathbb{F}_p)$, setze $k = \lceil 2\sqrt[4]{p}\,\rceil$.*

Schritt 2: *Für $i = 0, \ldots, k-1$ tue*

Falls $i \cdot P = \mathcal{O}$, gehe zurück zu Schritt 1,
ansonsten speichere $(i \cdot P, i)$ in einer Tabelle T.

Schritt 3: *Setze $Q = k \cdot P$.*

Schritt 4: *Für $j = 1, \ldots, k$ tue*

1. *Berechne den Punkt $H_j = \lfloor p + 1 - 2\sqrt{p}\,\rfloor \cdot P + j \cdot Q$.*
2. *Falls $H_j = i \cdot P$ für einen Tabelleneintrag $(i \cdot P, i)$, so speichere das Paar (j, i).*

Schritt 5: *Falls nur ein Paar (j, i) abgespeichert wurde, so gilt*

$$\#E(\mathbb{F}_p) \quad = \quad \lfloor p + 1 - 2\sqrt{p}\,\rfloor + k \cdot j - i$$

und der Algorithmus terminiert.

Ansonsten berechne mit zwei abgespeicherten Paaren (j, i) *und* (j', i')

$$l \;=\; ord(P) \;=\; \min\{d\,;\; d > 0,\; d \cdot P = \mathcal{O},\; d \mid k \cdot (j - j') - (i - i')\,\}\,.$$

Schritt 6: *Falls* $l < \sqrt{p} - 1$, *gehe zurück zu Schritt* 1.

Schritt 7: *Wähle zufällig einen Punkt* P' *aus* $E(\mathbb{F}_p)$.

Schritt 8: *Bestimme die Menge*

$$L = \{\, y\,;\; l \mid y,\; p + 1 - 2\sqrt{p} \le y \le p + 1 + 2\sqrt{p},\; y \cdot P' = \mathcal{O}\,\}\,.$$

Falls $L = \{x'\}$, *so terminiere mit Ergebnis*

$$\#E(\mathbb{F}_p) \quad = \quad x',$$

ansonsten wähle ein $x \in L$ *aus.*

Schritt 9: *Berechne* $l' \;=\; ord(P') \;=\; \min\{z\,;\; z > 0,\; z \mid x,\; z \cdot P' = \mathcal{O}\}$.

Falls $ord(P') = l' > l = ord(P)$, *setze* $P = P', l = l'$ *und gehe zu Schritt* 7.

Schritt 10: *Für alle Teiler* d *von* l' *in aufsteigender Reihenfolge tue*

1. *Setze* $\tilde{k} = \lceil \sqrt{l}\, \rceil$.
2. *Für* $\tilde{i} = 0, \ldots, \tilde{k} - 1$ *tue*
   (a) *Berechne den Punkt* $\tilde{i} \cdot P$.
   (b) *Falls* $\tilde{i} \cdot P = d \cdot P'$, *so setze* $d' = d$ *und gehe zu Schritt* 11, *ansonsten speichere* $(\tilde{i} \cdot P, \tilde{i})$ *in einer Tabelle* T'.
3. *Setze* $\tilde{Q} = \tilde{k} \cdot P$.
4. *Für* $\tilde{j} = 1, \ldots, \tilde{k}$ *tue*
   (a) *Berechne* $\tilde{H}_{\tilde{j}} = \tilde{j} \cdot \tilde{Q} - d \cdot P'$.
   (b) *Falls* $\tilde{H}_{\tilde{j}} = \tilde{i} \cdot P$ *für einen Eintrag* $(\tilde{i} \cdot P, \tilde{i})$ *der Tabelle* T', *so setze* $d' = d$ *und gehe zu Schritt* 11.

Schritt 11: *Bestimme die Menge*

$$L' = \{\, x\,;\; (d' \cdot l) \mid x,\; p + 1 - 2\sqrt{p} \le x \le p + 1 + 2\sqrt{p}\,\}.$$

Falls $L' = \{x'\}$, *terminiere mit*

$$\#E(\mathbb{F}_p) \quad = \quad x',$$

ansonsten gehe zurück zu Schritt 7.

2.3 Korrektheit des Verfahrens

In diesem Abschnitt wollen wir die Korrektheit des obigen Algorithmus 2.1 beweisen. Sei dazu ein Primkörper $\mathbb{F}_p$ und eine elliptische Kurve $E : y^2 = x^3 + ax + b$ über diesem Körper $\mathbb{F}_p$ fest vorgegeben.

Wie schon im vorherigen Abschnitt bemerkt, kann Algorithmus 2.1 in zwei Teile gegliedert werden:

- der erste Teil umfaßt die Schritte 1 bis 6.
- der zweite Teil umfaßt die darauffolgenden Schritte 7 bis 11.

Beide Teile können eventuell mehrmals durchlaufen werden. Zur Bestimmung der erwarteten Laufzeit des Shanks Algorithmus ist es daher von besonderem Interesse, die erwartete Anzahl der Schleifendurchläufe zu bestimmen. Dies wird im folgenden Kapitel geschehen. In diesem Abschnitt interessiert uns nur die Korrektheit der einzelnen Schritte und damit die Korrektheit des ganzen Algorithmus.

Bevor wir uns dem eigentlichen Beweis der Korrektheit zuwenden, beweisen wir zuerst, daß die "Darstellung" der Zahlen des Hasse-Intervalls als Paare der Form (j, i) eindeutig ist und daß alle natürlichen Zahlen des Hasse-Intervalls in dieser Form dargestellt werden können. Diese Form wird beispielsweise in Schritt 4 und 5 des Algorithmus benutzt.

Lemma 2.1 *Die Abbildung*

$$\begin{aligned} \phi : \{1, \ldots, k\} \times \{0, \ldots, k-1\} &\longrightarrow \mathbb{Z} \\ (j, i) &\longmapsto \lfloor p + 1 - 2\sqrt{p} \rfloor + j \cdot k - i, \end{aligned}$$

mit $k = \lceil 2\sqrt[4]{p} \rceil$ *ist injektiv und für das Hasse-Intervall* I *gilt*

$$I \cap \mathbb{Z} \subseteq im(\phi).$$

Beweis: Die Wohldefiniertheit der Abbildung ϕ folgt direkt aus der Wohldefiniertheit der Addition in $\mathbb{Z}$. Beweisen wir danach zuerst die Injektivität von ϕ. Nehmen wir dazu an, die Abbildung ϕ wäre nicht injektiv. Dann gibt es für eine Zahl $x \in im(\phi)$ zwei verschiedene Urbilder (j_x, i_x) und (j'_x, i'_x), d.h.

$$\begin{aligned} x &= \lfloor p + 1 - 2\sqrt{p} \rfloor + j_x \cdot k + i_x \\ &= \lfloor p + 1 - 2\sqrt{p} \rfloor + j'_x \cdot k + i'_x. \end{aligned}$$

Damit gilt

$$(j_x - j'_x) \cdot k = i'_x - i_x.$$

Nach Wahl von i'_x und i_x ist $|i'_x - i_x| < k$. Also muß $j_x - j'_x = 0$ sein. Damit erhalten wir $i'_x = i_x$ und $j'_x = j_x$, was im Widerspruch zu unserer Annahme steht. Also ist die Abbildung ϕ injektiv.

Nun zeigen wir, daß alle natürlichen Zahlen des Hasse-Intervalls

$$I = [\, p + 1 - 2\sqrt{p}\,,\; p + 1 + 2\sqrt{p}\,]$$

im Bild von ϕ liegen. Dazu zeigen wir zuerst, daß die kleinste und die größte natürliche Zahl von I diese Eigenschaft besitzen. Man beachte, daß p eine Primzahl ist und daß daher die kleinste naürliche Zahl des Hasse-Intervalls I die Zahl $\lfloor\, p + 1 - 2\sqrt{p}\rfloor + 1$ ist. Diese Zahl besitzt das Urbild $(1, k-1)$, denn

$$\phi((1, k-1)) \;=\; \lfloor\, p + 1 - 2\sqrt{p}\rfloor + 1 \cdot k - (k-1) \;=\; \lfloor\, p + 1 - 2\sqrt{p}\rfloor + 1.$$

Weiterhin erkennen wir sofort, daß die größte Zahl z des Bildes von ϕ für $i = 0$ und $j = k$ angenommen wird. Für diesen Wert z gilt

$$\begin{aligned} z &= \lfloor\, p + 1 - 2\sqrt{p}\,\rfloor + k \cdot k \\ &> \quad p + 1 - 2\sqrt{p} - 1 + (2\sqrt[4]{p})^2 \\ &= \quad p + 2\sqrt{p}. \end{aligned}$$

Da z nach Konstruktion eine natürliche Zahl ist und die größte natürliche Zahl des Hasse-Intervalls die Zahl $\lceil p + 2\sqrt{p}\,\rceil$ ist, folgt damit direkt, daß auch die größte natürliche Zahl des Hasse-Intervalls dargestellt wird.

Damit gilt, daß alle natürlichen Zahlen des Hasse-Intervalls im Bild von ϕ liegen. Ist nämlich $x = \phi((j,i))$, so erhält man ein Urbild von $x+1$ durch

$$\phi^{-1}(x+1) = \begin{cases} (j, i-1) & \text{falls } i > 0. \\ (j+1, k-1) & \text{falls } i = 0. \end{cases}$$

Dies verifiziert man direkt durch Nachrechnen. Damit haben wir Lemma 2.1 vollständig bewiesen. ■

Bemerkung 2.1 *Man beachte, daß im allgemeinen* $I \cap \mathbb{Z} \neq im(\phi)$ *gilt, d.h. es existieren auch einige Zahlen außerhalb des Hasse-Intervalls, die ein "gültiges" Urbild besitzen. Dies wird dadurch verursacht, daß wir für die Schrittweite der Giantsteps nicht den exakten Wert* $2\sqrt[4]{p}$ *nehmen können, sondern diesen Wert zur nächsten ganzen Zahl aufrunden müssen.*

Damit können wir jeder natürliche Zahl x des Hasse-Intervalls eindeutig ein Paar (j,i) mit $\phi((j,i)) = x$ zuordnen. Weiterhin können wir aus einem gegebenem Paar (j,i) leicht die entsprechende Zahl des Hasse-Intervalls berechnen, was beispielsweise in Schritt 5 benötigt wird. Damit können wir diese "Zahlendarstellung " zum Speichern von Zwischenergebnissen benutzen.

Wir haben in der Einleitung schon die Idee des Shanks Algorithmus angegeben. Da die Ordnung eines Punktes die Gruppenordnung teilt, versucht man durch zufällige Wahl von Punkten die Gruppenordnung zu bestimmen. Was geschieht aber, wenn sowohl $\#E(\mathbb{F}_p)$ als auch $2 \cdot \#E(\mathbb{F}_p)$ in dem Hasse-Intervall liegen? Dann können wir dieses Verfahren offensichtlich nicht mehr anwenden, denn für alle Punkte $P \in E(\mathbb{F}_p)$ gilt sowohl $(\#E(\mathbb{F}_p)) \cdot P = \mathcal{O}$ als auch $(2 \cdot \#E(\mathbb{F}_p)) \cdot P = \mathcal{O}$. Damit können wir nicht mehr unterscheiden, ob die Zahl $\#E(\mathbb{F}_p)$ oder $2 \cdot \#E(\mathbb{F}_p)$ die korrekte Lösung ist. Deshalb müssen wir die Primzahlen verbieten, für die dieser Fall auftreten kann. Dazu beachte man, daß nach dem Satz von Hasse 1.2

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p}$$

ist. Damit kann dieser Fall sicherlich nicht mehr auftreten, wenn wir alle Primzahlen p mit

$$2 \cdot (p + 1 - 2\sqrt{p}) \leq p + 1 + 2\sqrt{p}$$

ausschließen. Lösen wir diese Gleichung auf, so erhalten wir

$$p \leq 17 + 12 \cdot \sqrt{2} < 34.$$

Damit kann dieses Problem für alle Primzahlen ab 37 nicht mehr auftreten. Daher brauchen wir bei der Eingabe des Shanks Algorithmus 2.1 die zusätzliche Voraussetzung, daß die Charakteristik des Körpers mindestens 37 ist. Dies bedeutet für uns allerdings keine echte Einschränkung, da wir die Ordnung der elliptischen Kurven nur über Primkörpern wirklich großer Charakteristik berechnen wollen.

Zum Beweis der Korrektheit des Shanks Algorithmus 2.1 werden wir im folgenden einige Lemmata beweisen, aus denen dann die Korrektheit des Gesamtalgorithmus leicht gefolgert werden kann. Diese Lemmata beschäftigen sich jeweils mit Teilschritten des ersten und zweiten Teils des Shanks Algorithmus.

2.3.1 Korrektheit des ersten Teils

In diesem Abschnitt beschäftigen wir uns mit der Korrektheit des ersten Teils des Algorithmus. Dabei handelt es sich, wie schon öfters erwähnt, um die Schritte 1 bis 6. Wir beweisen, daß der Algorithmus bei Terminierung im ersten Teil das korrekte Ergebnis berechnet. Terminiert der Algorithmus nicht, so wird ein Punkt der Ordnung mindestens $\sqrt{p} - 1$ bestimmt. Wir werden beweisen, daß so ein Punkt immer existieren muß.

Proposition 2.1 *Wenn der Shanks Algorithmus 2.1 im ersten Teil terminiert, so berechnet er die Ordnung der Punktgruppe der elliptischen Kurve E korrekt.*

Beweis: Zum Beweis dieser Proposition benötigen wir zuerst folgendes Lemma:

Lemma 2.2 *Genau dann wird das Paar (j,i) in Schritt 4.2 des Shanks Algorithmus 2.1 abgespeichert, wenn für den in Schritt 1 gewählten Punkt P*

$$(\lfloor p+1-2\sqrt{p}\,\rfloor + j\cdot k - i)\cdot P = \mathcal{O}$$

gilt.

Beweis: (Lemma)
Ein Paar (j,i) wird in Schritt 4.2 nur dann abgespeichert, wenn $H_j = i\cdot P$ für einen Tabelleneintrag $(i\cdot P, i)$ ist. Also gilt dann

$$H_j \;=\; (\lfloor p+1-2\sqrt{p}\,\rfloor + j\cdot k)\cdot P \;=\; i\cdot P$$

und damit

$$(\lfloor p+1-2\sqrt{p}\,\rfloor + j\cdot k - i)\cdot P \;=\; \mathcal{O}.$$

Sei nun $x\cdot P = \mathcal{O}$ für eine natürliche Zahl $x \in [p+1-2\sqrt{p}, p+1+2\sqrt{p}]$. Nach Lemma 2.1 gibt es für diese Zahl x ein eindeutig bestimmtes Paar $(j,i) \in \{1,\ldots,k\}\times\{0,\ldots,k-1\}$ mit $\phi((j,i)) = x$. Damit gilt nach Definition von ϕ

$$\mathcal{O} = x\cdot P = (\lfloor p+1-2\sqrt{p}\,\rfloor + j\cdot k - i)\cdot P$$

und somit

$$H_j = (\lfloor p+1-2\sqrt{p}\,\rfloor + j\cdot k)\cdot P = i\cdot P.$$

Da $0 \le i < k$ ist, ist $(i\cdot P, i)$ ein Element der Tabelle T und damit wird in Schritt 4.2 des Algorithmus das Paar (j,i) abgespeichert. ■ (Lemma)

Zum Beweis der Proposition beachten wir, daß der Shanks Algorithmus im ersten Teil nur in Schritt 5 terminieren kann. Außerdem terminiert er in diesem Schritt nur dann, wenn in Schritt 4 genau ein Paar (j,i) abgespeichert wurde.

Nach Satz 2.1 teilt die Ordnung des in Schritt 1 gewählten Punktes P die Gruppenordnung $\#E(\mathbb{F}_p)$. Daher gibt es mindestens eine Zahl x – nämlich die Ordnung $\#E(\mathbb{F}_p)$ der Gruppe $E(\mathbb{F}_p)$ – in dem Hasse-Intervall, die ein Vielfaches der Punktordnung von P ist. Außerdem haben wir in Lemma 2.1 schon gesehen, daß wir bei Wahl der Variablen i und j wie im Algorithmus das ganze Hasse-Intervall durchtesten. Folglich wird nach Lemma 2.2 mindestens ein Paar (j,i) – nämlich die Darstellung der Gruppenordnung – in Schritt 4 abgespeichert. Wird also genau ein Paar (j,i) abgespeichert, dann muß dies die Darstellung der Gruppenordnung sein. Die zu dieser Darstellung korrespondierende natürliche Zahl wird in diesem Fall als Gruppenordnung ausgegeben, so daß der Algorithmus mit der richtigen Lösung terminiert. ■

Bemerkung 2.2 *Aus Lemma 2.2 folgt direkt die Korrektheit der Babystep-Giantstep Strategie. Weiterhin sollte man beachten, daß man eine analoge Strategie auch benutzen kann, um eine Zahl* $0 \leq x < m$ *mit* $x \cdot P = Q$ *für Punkte* $P, Q \in E(\mathbb{F}_p)$ *zu bestimmen. Dazu müssen wir einfach die Anzahl der Babysteps als* $\lceil \sqrt{m} \rceil$ *wählen sowie den Anfang des Suchintervalls entsprechend verschieben. Dann folgt auf exakt die gleiche Weise wie in Lemma 2.2 die Korrektheit dieser etwas veränderten Babystep-Giantstep Methode.*

Damit haben wir den Fall behandelt, daß wir in Schritt 1 einen Punkt P mit genügend großer Ordnung gewählt haben. Nun beschäftigen wir uns mit der Frage, welche Information wir gewinnen, wenn der Algorithmus nicht in Schritt 5 terminiert, d.h. wenn die Ordnung des Punktes P nicht groß genug ist und damit mindestens zwei Vielfache der Punktordnung im Hasse-Intervall liegen. Dann werden nach Lemma 2.2 mindestens zwei Paare (j, i) und (j', i') abgespeichert. Wir beweisen im folgenden Lemma, daß wir mit Hilfe dieser beiden Paare die Ordnung des Punktes P berechnen können.

Lemma 2.3 *Falls der Shanks Algorithmus 2.1 in Schritt* 5 *nicht terminiert, so wird die Ordnung des Punktes* P *berechnet.*

Beweis: Der Algorithmus terminiert genau dann nicht in Schritt 5, wenn mehr als ein Paar (j, i) im Schritt 4 abgespeichert wurde. Seien (j, i) und (j', i') zwei abgespeicherte Paare. Dann gilt nach Lemma 2.2

$$\begin{aligned} \mathcal{O} &= (\lfloor p + 1 - 2\sqrt{p} \rfloor + j \cdot k - i) \cdot P \\ &= (\lfloor p + 1 - 2\sqrt{p} \rfloor + j' \cdot k - i') \cdot P \end{aligned}$$

und damit

$$((j - j') \cdot k - (i - i')) \cdot P = \mathcal{O}.$$

Also muß der Wert $x = |(j - j') \cdot k - (i - i')|$ ein Vielfaches der Punktordnung sein.

Die genaue Punktordnung erhalten wir nun aus x, indem wir den kleinsten Teiler x' von x bestimmen, für den $x' \cdot P = \mathcal{O}$ ist. Genau diese Berechnung wird aber in Schritt 5 des Shanks Algorithmus durchgeführt, wenn zwei Paare (j, i) und (j', i') abgespeichert wurden. Damit folgt die Behauptung des Lemmas. ■

Falls in Schritt 1 also zufällig ein Punkt P gewählt wird, dessen Ordnung größer als die Intervallänge ist, terminiert der Shanks Algorithmus 2.1 schon in Schritt 5. Ein experimentelles Ergebnis unserer Implementierung ist die Tatsache, daß für fast alle elliptischen Kurven so ein Punkt existiert und auch direkt gewählt wird, daß also die Schritte 1 bis 5 nur einmal durchlaufen werden und der Algorithmus dann mit der korrekten Lösung terminiert. Wie schon in der Einleitung erwähnt, gibt es allerdings auch elliptische Kurven, für die dies nicht gilt. Nun untersuchen wir den Fall, daß kein solcher "guter" Punkt gefunden wurde. Da wir die Schleife des ersten Teils in Schritt 6 des Algorithmus nur dann abbrechen, wenn wir einen Punkt der Ordnung mindestens $\sqrt{p}-1$ gefunden haben, müssen wir die Existenz eines solchen Punktes garantieren, um keine Endlosschleife der Schritte 1 bis 6 zu erhalten. Wählen wir dann so einen Punkt, so verlassen wir den ersten Teil und beginnen mit dem zweiten Teil des Algorithmus. Um die Existenz eines solchen Punktes beweisen zu können, benötigen wir einige Informationen über den Isomorphismustyp der Punktgruppe einer elliptischen Kurve. Diese Informationen liefert der folgende Satz von Rück.

Satz 2.2 (Satz von Rück)

Falls die Ordnung der Gruppe der $\mathbb{F}_p$-rationalen Punkte $E(\mathbb{F}_p)$ einer elliptischen Kurve E über dem Körper $\mathbb{F}_p$ die Primzerlegung

$$\#E(\mathbb{F}_p) \;=\; \prod_{l\in\mathbb{P}} l^{\nu_l}$$

besitzt, so besitzt die Gruppe den Isomorphismustyp

$$E(\mathbb{F}_p) \;\cong\; \bigoplus_{l\in\mathbb{P}} (\mathbb{Z}/l^{\alpha_l}\mathbb{Z} \oplus \mathbb{Z}/l^{\nu_l-\alpha_l}\mathbb{Z}) \tag{2.1}$$

mit

$$0 \le \alpha_l \le \min\left(w_l(p-1), \left\lfloor \frac{\nu_l}{2} \right\rfloor\right).$$

Dabei ist $w_l(x) = \max\{k;\ l^k|x\}$.

Beweis: [Rü87] ∎

Bemerkung 2.3 *Wir geben hier eine Spezialisierung des Satzes von Rück für den Fall $K = \mathbb{F}_p$ wider. Der Originalsatz behandelt noch den allgemeineren Fall, daß K ein endlicher Erweiterungskörper von $\mathbb{F}_p$ ist. Desweiteren sollte man beachten, daß die Summe in (2.1) endlich ist, denn in der Primfaktorzerlegung von $\#E(\mathbb{F}_p)$ kommen nur endlich viele Primzahlen mit Potenz größer als Null vor.*

Korollar 2.1 *Für den Isomorphismustyp einer elliptischen Kurve E über dem Primkörper $\mathbb{F}_p$ gilt*

$$E(\mathbb{F}_p) \;\cong\; \mathbb{Z}/n\mathbb{Z} \oplus \mathbb{Z}/m\mathbb{Z} \qquad \textit{mit } m, n \ge 1. \tag{2.2}$$

Dabei ist n sowohl ein Teiler vom m als auch von $p-1$.

Beweis: Man beachte, daß in Satz 2.2 $\alpha_l \le \lfloor \frac{\nu_l}{2} \rfloor$ ist und damit $\alpha_l \le \nu_l - \alpha_l$ gilt. Außerdem gilt $\alpha_l \le w_l(p-1)$. Setzen wir nun

$$n = \prod_{l\in\mathbb{P}} l^{\alpha_l} \qquad \text{und} \qquad m = \prod_{l\in\mathbb{P}} l^{\nu_l-\alpha_l},$$

so ist die Behauptung des Korollars offensichtlich. ∎

Bemerkung 2.4 *Diese Typaussage bedeutet, daß es einen Punkt P der Ordnung m und einen Punkt Q der Ordnung n gibt, so daß*

$$E(\mathbb{F}_p) = \{ i \cdot Q + j \cdot P \; ; \; 0 \leq i < n \, , \; 0 \leq j < m \} .$$

Zusätzlich gilt noch $\langle P \rangle \cap \langle Q \rangle = \{\mathcal{O}\}$.

Diese Bemerkung werden wir im folgenden noch häufig benutzen. Eine erste Anwendung erfolgt nun direkt, denn mit Hilfe von Bemerkung 2.4 können wir beweisen, daß es immer einen Punkt der Ordnung mindestens $\sqrt{p} - 1$ gibt. Dies geschieht im folgenden Lemma.

Lemma 2.4 *Sei E eine elliptische Kurve über dem Primkörper $\mathbb{F}_p$. Dann besitzt $E(\mathbb{F}_p)$ eine zyklische Untergruppe mit Ordnung mindestens $\sqrt{p} - 1$, d.h. insbesondere gibt es einen Punkt $P \in E(\mathbb{F}_p)$, der mindestens Ordnung $\sqrt{p} - 1$ besitzt.*

Beweis: Nach Bemerkung 2.4 wissen wir, daß

$$E(\mathbb{F}_p) = \langle Q \rangle \oplus \langle P \rangle$$

mit zwei Punkten P und Q. Da wegen $\langle P \rangle \cap \langle Q \rangle = \{\mathcal{O}\}$ die Gleichheit $\#E(\mathbb{F}_p) = \mathrm{ord}\,(Q) \cdot \mathrm{ord}\,(P)$ und o.B.d.A. $\mathrm{ord}\,(Q) \leq \mathrm{ord}\,(P)$ gilt, erhalten wir

$$\mathrm{ord}\,(P) \geq \sqrt{\#E(\mathbb{F}_p)}.$$

Nach dem Satz von Hasse 1.2 wissen wir, daß die Ordnung der Punktgruppe $E(\mathbb{F}_p)$ nach unten durch $p + 1 - 2\sqrt{p}$ beschränkt ist. Folglich erhalten wir

$$\mathrm{ord}\,(P) \geq \sqrt{\#E(\mathbb{F}_p)} \geq \sqrt{p + 1 - 2\sqrt{p}} = \sqrt{(\sqrt{p} - 1)^2} = \sqrt{p} - 1.$$

Daher besitzt die von P erzeugte Untergruppe mindestens $\sqrt{p} - 1$ Elemente und ist somit die gewünschte zyklische Untergruppe von $E(\mathbb{F}_p)$. ■

Mit diesem Lemma 2.4 haben wir nur die Existenz mindestens eines Punktes bewiesen, dessen Ordnung mindestens $\sqrt{p} - 1$ ist. Damit wird der erste Teil des Shanks Algorithmus 2.1 bei Wahl des "richtigen" Punktes verlassen. Aussagen über die Anzahl solcher "richtiger" Punkte und damit über die Wahrscheinlichkeit, einen solchen Punkt in Schritt 1 zu wählen, werden wir im folgenden Kapitel herleiten. Diese Informationen werden dann einen großen Einfluß auf die zu erwartende Laufzeit besitzen.

2.3.2 Korrektheit des zweiten Teils

Sei die Situation nun die folgende: wir befinden uns vor Schritt 7 des Shanks Algorithmus und haben den ersten Teil des Algorithmus verlassen, aber die Gruppenordnung $\#E(\mathbb{F}_p)$

ist noch nicht bestimmt. Im folgenden werden wir beweisen, daß im zweiten Teil des Algorithmus – d.h. in den Schritte 7 bis 11 – dann die Ordnung der Punktgruppe korrekt berechnet wird.

Wenn wir in den zweiten Teil des Shanks Algorithmus gelangt sind, kennen wir einen Punkt P, dessen Ordnung die Eigenschaft

$$\sqrt{p} - 1 \leq \operatorname{ord}(P) < 4\sqrt{p}$$

besitzen muß. Wäre die Ordnung von P nämlich größer als $4\sqrt{p}$, so wäre nur ein Vielfaches der Punktordnung im Hasse-Intervall und folglich wäre der Algorithmus schon in Schritt 5 terminiert. Zur Verdeutlichung der Vorgehensweise erklären wir nun die Aktionen der einzelnen Schritte des zweiten Teils:

- In Schritt 8 und 9 wird die Ordnung eines in Schritt 7 zufällig gewählten Punktes P' bestimmt.
- In Schritt 10 berechnen wir daraus die Ordnung von P' in der Faktorgruppe $E(\mathbb{F}_p)/\langle P \rangle$.
- In Schritt 11 wird getestet, ob diese neue Information ausreicht, um die Gruppenordnung $\#E(\mathbb{F}_p)$ eindeutig zu bestimmen.

Wir werden nun die einzelnen Schritte genauer untersuchen und ihre Korrektheit beweisen. Dabei müssen wir beachten, daß der zweite Teil nicht immer vollständig durchlaufen werden muß, sondern daß möglicherweise auch schon in Schritt 9 die Gruppenordnung korrekt berechnet sein kann.

Lemma 2.5 *In den Schritten 8 und 9 des Shanks Algorithmus 2.1 wird entweder die Ordnung des Punktes P' korrekt bestimmt oder der Algorithmus terminiert mit der korrekt berechneten Lösung $\#E(\mathbb{F}_p)$.*

Beweis: Sei ein Punkt $P \in E(\mathbb{F}_p)$ mit Ordnung l bekannt. Dann wissen wir nach Satz 2.1 schon, daß die Gruppenordnung $\#E(\mathbb{F}_p)$ ein Vielfaches der Ordnung l des Punktes P im Hasse-Intervall ist. Zusätzlich gilt für die Gruppenordnung, daß sie auch Vielfaches der zu bestimmenden Ordnung von P' ist. Damit muß die Gruppenordnung eine Zahl x des Hasse-Intervalls sein, die von l geteilt wird und für die $x \cdot P' = \mathcal{O}$ gilt; also ist die Gruppenordnung ein Element der Menge

$$L = \{x \,;\, l \mid x,\ p + 1 - 2\sqrt{p} \leq x \leq p + 1 + 2\sqrt{p},\ x \cdot P' = \mathcal{O}\}\,.$$

Nach Satz 2.1 ist die Menge L nicht leer. Falls L genau ein Element enthält, so muß dies aus den gleichen Gründen wie in Proposition 2.1 die Gruppenordnung sein. Folglich terminiert der Algorithmus dann mit der korrekten Lösung.

Ansonsten wählen wir einen Wert x zufällig aus der Menge L aus. Da $x \cdot P' = \mathcal{O}$ ist, ist x ein Vielfaches der Ordnung von P'. Wenn wir also den kleinsten Teiler l' von x finden, so daß $l' \cdot P' = \mathcal{O}$ ist, dann haben wir die Ordnung von P' bestimmt. Genau diese Berechnung wird in Schritt 9 durchgeführt. ■

Damit kennen wir nach Schritt 9 die Ordnung von P' in der Punktgruppe $E(\mathbb{F}_p)$. Weiterhin können wir nach Schritt 9 annehmen, daß die Ordnung von P' maximal so groß wie die Ordnung von P ist. Dies wird durch die Abfrage im zweiten Teil von Schritt 9 bewirkt. Unser Ziel ist es nun, mit Hilfe dieser Ordnung die Ordnung von P' in der Faktorgruppe $E(\mathbb{F}_p)/\langle P\rangle$ (abgekürzt mit $\mathrm{ord}_{E(\mathbb{F}_p)/\langle P\rangle}(P')$) zu bestimmen. Zuerst werden wir nun beweisen, daß für diese Ordnung $\mathrm{ord}_{E(\mathbb{F}_p)/\langle P\rangle}(P')$ nur bestimmte Werte in Frage kommen.

Lemma 2.6 *Sei $E(\mathbb{F}_p)$ die Punktgruppe einer elliptischen Kurve über dem Primkörper $\mathbb{F}_p$, sei $\langle P\rangle$ die von dem Punkt $P \in E(\mathbb{F}_p)$ erzeugte Untergruppe. Sei $Q \in E(\mathbb{F}_p)$ ein beliebiger Punkt.*

Dann gilt: Falls $k > 0$ minimal mit $k \cdot Q \in \langle P\rangle$, d.h. $k = ord_{E(\mathbb{F}_p)/\langle P\rangle}(Q)$, so ist k ein Teiler der Ordnung von Q in der Punktgruppe $E(\mathbb{F}_p)$.

Beweis: Sei $\mathrm{ord}\,(Q) = l$ und $k \cdot Q \in \langle P\rangle$ mit minimalem $k > 0$. Wir wollen annehmen, daß k kein Teiler von l ist. Dann existieren natürliche Zahlen i und j mit $0 < i < k$ und

$$l = j \cdot k + i.$$

Nach unserer Annahme gilt $k \cdot Q = h \cdot P$ für ein $0 \leq h < \mathrm{ord}\,(P)$. Daraus folgt

$$\begin{aligned} \mathcal{O} = l \cdot Q &= (j \cdot k + i) \cdot Q \\ &= j \cdot (k \cdot Q) + i \cdot Q \\ &= j \cdot (h \cdot P) + i \cdot Q \\ &= j \cdot h \cdot P + i \cdot Q. \end{aligned}$$

Durch Umformung dieser Gleichung erhalten wir

$$i \cdot Q = -j \cdot h \cdot P,$$

d.h. daß schon $i \cdot Q \in \langle P\rangle$ ist. Da $0 < i < k$ gilt, ist dies ein Widerspruch zu unserer Annahme, daß k minimal gewählt war. Also muß k ein Teiler von l sein. ■

Damit können wir nun die Korrektheit des Schritts 10 des Shanks Algorithmus 2.1 beweisen.

Lemma 2.7 *Seien $P, P' \in E(\mathbb{F}_p)$ Punkte mit Ordnung l bzw. l'. Dann wird in Schritt 10 des Shanks Algorithmus 2.1 die Ordnung von P' in der Faktorgruppe $E(\mathbb{F}_p)/\langle P\rangle$ bestimmt.*

Beweis: Wir können nach Lemma 2.6 die Ordnung von P' in $E(\mathbb{F}_p)/\langle P\rangle$ bestimmen, indem wir den kleinsten Teiler d' von l' finden, so daß $d'\cdot P' \in \langle P\rangle$ ist. Dazu müssen wir für festes d' für alle Werte $i = 0,\ldots,\mathrm{ord}(P)-1$ überprüfen, ob $i\cdot P = d'\cdot P'$ ist. Finden wir ein solches i und ist d' minimal mit dieser Eigenschaft, so ist die Ordnung von P' in der Faktorgruppe $\mathrm{ord}_{E(\mathbb{F}_p)/\langle P\rangle}(Q) = d'$.

Für festes d' führen wir diesen Test wiederum mit einer Babystep-Giantstep Strategie durch. Da das zu durchsuchende "Lösungsintervall" $\{0,\ldots,\mathrm{ord}\,(P)-1\}$ diesmal die Größe ord (P) besitzt, wählen wir in diesem Fall die Anzahl der Babysteps als $\lceil\sqrt{\mathrm{ord}\,(P)}\,\rceil = \lceil\sqrt{l}\,\rceil$. Die Korrektheit dieses Babystep-Giantstep Teils folgt direkt aus den Bemerkungen zum ersten Teil und sollte damit klar sein. Allerdings sollte man beachten, daß in diesem Fall keine Lösung existieren muß. Existiert diese jedoch, so wird sie mit der Babystep-Giantstep Strategie gefunden.

Wenn wir Schritt 10 des Shanks Algorithmus 2.1 betrachten, erkennen wir, daß wir mit dieser Babystep-Giantstep Strategie für alle Teiler d von l' in aufsteigender Reihenfolge überprüfen, ob $d\cdot Q \in \langle P\rangle$ ist. Den Babystep-Giantstep Teil, der diesen Test durchführt, befindet sich dabei in den Teilschritten 10.1 bis 10.4 . Falls dieser Test erfolgreich ist und wir damit $\mathrm{ord}_{E(\mathbb{F}_p)/\langle P\rangle}(P')$ bestimmt haben, setzen wir die Variable $d' = \mathrm{ord}_{E(\mathbb{F}_p)/\langle P\rangle}(Q)$ auf den entsprechenden Wert und gehen weiter zu Schritt 11. ■

Die damit neu gewonnene Information hilft uns bei der Bestimmung der Gruppenordnung $\#E(\mathbb{F}_p)$. Die Ordnung von P' in der Faktorgruppe $E(\mathbb{F}_p)/\langle P\rangle$ liefert nämlich auch zusätzliche Informationen über die möglichen Werte der Gruppenordnung $\#E(\mathbb{F}_p)$. Dies folgt aus folgendem Lemma.

Lemma 2.8 *Sei E eine über $\mathbb{F}_p$ definierte elliptische Kurve und P und P' zwei Punkte aus $E(\mathbb{F}_p)$. Falls P' die Ordnung d' in der Faktorgruppe $E(\mathbb{F}_p)/\langle P\rangle$ besitzt, so ist $ord(P)\cdot d'$ ein Teiler von $\#E(\mathbb{F}_p)$.*

Beweis: Nach Satz 2.1 gilt, daß die Ordnung von P' in der Faktorgruppe $E(\mathbb{F}_p)/\langle P\rangle$ ein Teiler der Faktorgruppenordnung ist. Weiterhin gilt folgende Aussage für die Ordnung der Faktorgruppe ([Hun74, Th.5.4, Seite 42]):

$$\#\,(E(\mathbb{F}_p)/\langle P\rangle) \quad = \quad \frac{\#E(\mathbb{F}_p)}{\#\langle P\rangle}.$$

Da $\#\langle P\rangle = \mathrm{ord}\,(P)$ ist, folgt somit

$$d' \;\Big|\; \frac{\#E(\mathbb{F}_p)}{\mathrm{ord}\,(P)}$$

und damit $d'\cdot\mathrm{ord}\,(P) \mid \#E(\mathbb{F}_p)$. ■

Somit liefert die Bestimmung der Ordnung d' des Punktes P' in der Faktorgruppe $E(\mathbb{F}_p)/\langle P\rangle$ weitere Information über die Gruppenordnung, denn – falls $d' > 1$ – kennen wir nun einen größeren Teiler der Gruppenordnung $\#E(\mathbb{F}_p)$. Folglich können wir jetzt wiederum testen, ob diese neue Information ausreicht, um die Gruppenordnung $\#E(\mathbb{F}_p)$ eindeutig zu bestimmen. Gibt es nun nur noch ein Vielfaches von $\mathrm{ord}\,(P) \cdot d'$ in dem Hasse-Intervall, so ist die Gruppenordnung eindeutig bestimmt. Dann gibt der Algorithmus die korrekte Lösung aus und terminiert. Ansonsten müssen wir den zweiten Teil des Algorithmus mit einem anderen zufälligen Punkt P' noch einmal durchführen. Wenn wir nun noch gezeigt haben, daß es immer einen Punkt P' gibt, der zur Terminierung des Algorithmus führt, haben wir das Ziel erreicht und die Korrektheit des Shanks Algorithmus 2.1 vollständig bewiesen.

Lemma 2.9 *Es gibt mindestens einen Punkt P', so daß der zweite Teil des Shanks Algorithmus mit der korrekten Gruppenordnung $\#E(\mathbb{F}_p)$ terminiert.*

Beweis: Aus der Typaussage $E(\mathbb{F}_p) \cong \mathbb{Z}/n\mathbb{Z} \oplus \mathbb{Z}/m\mathbb{Z}$ folgt, daß es sicherlich einen Punkt P' gibt, der zur Terminierung des Shanks Algorithmus führt, wenn die Ordnung des Punktes P gleich m ist. Ist die Ordnung des Punktes P kleiner als m, so führt die zufällige Wahl eines Punktes P' mit größerer Ordnung dazu, daß in Schritt 9 der Punkt P als P' gewählt wird. Daher können wir annehmen, daß nach Wahl eines Punktes P' mit $\mathrm{ord}\,(P') = m$ der Punkt P ab der nächsten Iteration ebenfalls die Ordnung m besitzt. Dann folgt aber die Existenz eines zur Terminierung führenden Punktes schon aus obigen Bemerkungen. Falls der Algorithmus aber terminiert, so terminiert er immer nur mit der korrekten Lösung, wie wir schon gezeigt haben. ■

Damit haben wir den folgenden Satz bewiesen.

Satz 2.3 *Der Shanks Algorithmus 2.1 berechnet bei Eingabe einer elliptischen Kurve E über einem Primkörper $\mathbb{F}_p$ mit $p \geq 37$ die Ordnung der Punktgruppe $E(\mathbb{F}_p)$ korrekt.*

Kapitel 3

Erwartete Laufzeit des Shanks Algorithmus

Im vorherigen Kapitel haben wir die Korrektheit des Shanks Algorithmus zur Berechnung der Punktanzahl einer elliptischen Kurve E über einem Primkörper $\mathbb{F}_p$ gezeigt. Wir haben dabei allerdings noch keine Aussagen über die zu erwartende Laufzeit dieses Algorithmus gemacht. Dies wird in diesem Kapitel geschehen. Als Maß zur Bestimmmung der Komplexität werden wir dabei die Anzahl der Bitoperationen verwenden, die wir für eine bestimmte Operation brauchen.

Sei im folgenden $E : y^2 = x^3 + ax + b$ die Gleichung einer festen, über dem Primkörper $\mathbb{F}_p$ definierten elliptischen Kurve E.

3.1 Laufzeit der Operationen auf Punkten

Bevor wir uns mit der Analyse des eigentlichen Shanks Algorithmus 2.1 beschäftigen, werden wir in diesem Abschnitt die Komplexität der Punktaddition und der Berechnung von Vielfachen von Punkten bestimmen. Dieses sind Grundoperationen bei Rechnungen mit elliptischen Kurven und werden demnach häufig benötigt. Wie wir aus Satz 1.1 ersehen, benötigen wir bei der Punktaddition die Invertierung von Elementen aus dem Körper $\mathbb{F}_p$. Deshalb nun zuerst zur Komplexität des **erweiterten euklidischen Algorithmus**, mit dessen Hilfe wir eine solche Invertierung durchführen können.

Satz 3.1 *Der erweiterte euklidische Algorithmus zur Berechnung der Darstellung des größten gemeinsamen Teilers zweier Zahlen $a \geq b$ benötigt Zeit $O(\log(a)^2)$.*

Beweis: Zuerst geben wir den erweiterten euklidischen Algorithmus an. Dabei stützen wir uns auf die Beschreibung aus [Co91, Seite 13].

Algorithmus 3.1 (Erweiterter Euklidischer Algorithmus)

Eingabe: *positive Zahlen* $a \geq b > 0$.
Ausgabe: *Darstellung* $ggt(a,b) = u \cdot a + v \cdot b$.

1. *Setze* $i = 0$, $a_0 = a$, $a_1 = b$, $v = 0$, $u = 1$.
2. *Falls* $a_{i+1} = 0$, *dann setze* $v = (a_i - a \cdot u)/b$ *und terminiere mit* $ggt(a,b) = a_i$.
3. *Bestimme* $a_i = q_i \cdot a_{i+1} + a_{i+2}$, $t = u - q_i \cdot v$, $u = v$, $v = t$, $i = i + 1$ *und gehe zu 2.*

Zur Korrektheit dieses Algorithmus beachte man [Co91, Seite 13]. Nun zur Bestimmung der Komplexität dieses Algorithmus. Dabei wollen wir ausnutzen, daß die Zahlen a_i in dem Schritt 3 immer kleiner werden. Daher verwenden wir folgende Ergebnisse aus [Kn73, Abschnitt 4.3]:

- Das Produkt $q_i \cdot v$ kann in $O(\log(q_i) \cdot \log(|v|))$ vielen Bitoperationen berechnet werden.
- Die Division mit Rest $a_i = q_i \cdot a_{i+1} + a_{i+2}$ kann in $O(\log(a_{i+1}) \cdot (\log(q_i) + 1))$ vielen Bitoperationen erfolgen.

Sei l der kleinste Index mit $a_{l+1} = 0$, d.h. a_l sei der größte gemeinsame Teiler von a und b. Dann erhalten wir als Laufzeit der Schleife in den Schritten 2 und 3

$$\begin{aligned} T &= \sum_{i=0}^{l-1} O\left(\log(a_{i+1}) \cdot (\log(q_i) + 1)\right) + O\left(\log(q_i) \cdot \log(|v|)\right) \\ &\leq \log(a) \cdot O\left(\sum_{i=0}^{l-1} 2 \cdot \log(q_i) + 1\right). \end{aligned}$$

Dabei beachte man, daß während aller Schleifendurchläufe die Abschätzung $|v| \leq a$ gilt. Dies erkennt man, wenn man mit vollständiger Induktion beweist, daß nach dem $(i+2)$-ten Schleifendurchlauf $|v| \leq q_0 \cdot \ldots \cdot q_i$ gilt und dann dies mit dem nun folgenden Trick abschätzt. Schätzen wir nun $\sum_{i=0}^{l-1} \log(q_i)$ ab. Im Algorithmus wird immer

$$q_i = \left\lfloor \frac{a_i}{a_{i+1}} \right\rfloor \leq \frac{a_i}{a_{i+1}}$$

gewählt. Also gilt

$$\begin{aligned}\sum_{i=0}^{l-1}\log(q_i) &\leq \sum_{i=0}^{l-1}\log\left(\frac{a_i}{a_{i+1}}\right)\\ &= \sum_{i=0}^{l-1}\log(a_i)-\log(a_{i+1})\\ &= \log(a_0)-\log(a_l)\\ &\leq \log(a).\end{aligned}$$

Weiterhin kann man leicht mit Hilfe vollständiger Induktion zeigen, daß $a_{i+2} \leq \frac{1}{2}\cdot a_i$ gilt. Damit gilt auch für die Anzahl l der Schleifendurchläufe $l = O(\log(a))$. Setzen wir diese Ergebnisse zusammen, so erhalten wir leicht die im Satz behauptete Laufzeit. ■

Damit können wir nun die Komplexität der Addition zweier Punkte der elliptischen Kurve bestimmen.

Proposition 3.1 *Gegeben zwei Punkte P und Q der elliptischen Kurve E über dem Primkörper $\mathbb{F}_p$. Dann kann die Summe $P+Q$ in $O\left(\log(p)^2\right)$ vielen Bitoperationen berechnet werden.*

Beweis: Zur Berechnung der Summe zweier Punkte können wir die Formeln aus Satz 1.1 verwenden. Dabei benötigen wir an Operationen über dem Körper $\mathbb{F}_p$ einige Additionen und Multiplikationen sowie eine Invertierung einer Zahl aus $\mathbb{F}_p$. Additionen und Multiplikationen in $\mathbb{F}_p$ können wir mit anschließender Reduktion in $O(\log(p)^2)$ vielen Bitoperationen durchführen. Die Invertierung einer Zahl in $\mathbb{F}_p$ kann mit Hilfe des erweiterten euklidischen Algorithmus nach Satz 3.1 ebenfalls in Zeit $O(\log(p)^2)$ durchgeführt werden. Da nur eine konstante Anzahl von Additionen und Multiplikationen benötigt wird, ist somit auch die Komplexität der Addition zweier Punkte auf einer elliptischen Kurve $O(\log(p)^2)$. ■

Anschließend untersuchen wir die Komplexität des Problems, für einen Punkt $P \in E(\mathbb{F}_p)$ und eine Zahl $x \in \mathbb{Z}$ den Punkt $x\cdot P \in E(\mathbb{F}_p)$ zu berechnen. Zur Berechnung dieses Vielfachen verwenden wir einen Algorithmus ähnlich der schnellen Exponentiation. Näheres liefert die folgende Proposition.

Proposition 3.2 *Sei $P \in E(\mathbb{F}_p)$ und $x \in \mathbb{Z}$. Dann kann der Punkt $x \cdot P$ in*

$$O\left(\log(|x|)\cdot\log(p)^2\right)$$

vielen Bitoperationen berechnet werden.

Beweis: Zum Beweis dieser Proposition geben wir einen Algorithmus zur Bestimmung des Vielfachen an:

Algorithmus 3.2 (Vielfache eines Punktes)

Eingabe: *$P \in E(\mathbb{F}_p)$ und $x \in \mathbb{Z}$.*
Ausgabe: *$x \cdot P$.*

1. *Setze $H = \mathcal{O}$.*

2. *Falls $x < 0$, setze $x = |x|$; $P = -P$.*

3. *Solange $x > 0$ tue*
   1. *Falls $x \equiv 1 \bmod 2$, setze $H = H + P$.*
   2. *Setze $P = P + P$ und $x = \lfloor \frac{x}{2} \rfloor$.*

4. *Gib H als Ergebnis zurück.*

Dieser Algorithmus ist eine direkte Übertragung des Algorithmus zur schnellen Exponentiation in der Gruppe $(\mathbb{F}_p^\times, \cdot)$ auf die additive Gruppe $(E(\mathbb{F}_p), +)$. Damit folgt seine Korrektheit direkt aus der Korrektheit der schnellen Exponentiation. Zur Bestimmung der Laufzeit beachte man, daß x in jedem Schleifendurchlauf mindestens halbiert wird. Damit ist direkt klar, daß die Schleife in Schritt 3 maximal $O(\log(|x|))$-mal durchlaufen wird. Innerhalb der Schleife werden nur Additionen von Punkten sowie Division mit Rest von ganzen Zahlen benötigt. Beides kann in $O(\log(p)^2)$ vielen Bitoperationen durchgeführt werden. Dies liefert das Ergebnis der Proposition. ■

Bemerkung 3.1 *In dem Shanks Algorithmus gilt für die Zahl x immer, daß sie betragsmäßig kleiner als die obere Hasse-Schranke $p + 1 + 2\sqrt{p}$ ist. Daher können wir $\log(|x|)$ durch $\log(p + 1 + 2\sqrt{p}) = O(\log(p))$ ersetzen und erhalten dann als Laufzeit von Algorithmus 3.2 $O(\log(p)^3)$. Diese obere Abschätzung der Laufzeit werden wir im folgenden verwenden, wenn wir die Laufzeit der Berechnung von Vielfachen eines Punktes benötigen.*

3.2 Der Punktsuche-Algorithmus

In Schritt 1 und 7 des Shanks Algorithmus 2.1 wird vorgeschrieben, einen Punkt auf einer elliptischen Kurve E über einem Primkörper $\mathbb{F}_p$ zufällig zu wählen. Wir stellen hier einen probabilistischen Algorithmus zur Lösung dieses Problems vor.

Algorithmus 3.3 (Punktsuche-Algorithmus)

Eingabe: *elliptische Kurve $E : y^2 = x^3 + ax + b$ über Primkörper $\mathbb{F}_p$.*
Ausgabe: *zufälliger Punkt $P \in E(\mathbb{F}_p)$ ungleich $\mathcal{O}$.*

1. *Wähle zufällig eine Zahl $\tilde{x} \in \mathbb{F}_p$.*

2. *Teste, ob $\tilde{x}^3 + a\tilde{x} + b$ ein Quadrat modulo p ist.*
   Falls nein, gehe zurück zu 1.

3. *Berechne eine Quadratwurzel $\tilde{y}$ modulo p von $\tilde{x}^3 + a\tilde{x} + b$ und gib den Punkt $P = (\tilde{x}, \tilde{y})$ als Ergebnis zurück.*

Dieser Algorithmus enthält noch zwei wesentliche Lücken, denn es ist noch nicht klar, wie wir den Test in Schritt 2 und die Berechnung der Quadratwurzel modulo der Primzahl p in Schritt 3 durchführen können. Falls wir Lösungen für die zwei Lücken finden, folgt die Korrektheit von Algorithmus 3.3 direkt aus der Definition von Punkten einer elliptischen Kurve. Wenden wir uns daher nun den Lücken zu:

- Der Test, ob $\tilde{x}^3 + a\tilde{x} + b$ ein Quadrat modulo p ist, kann durch Berechnung des Legendre-Symbols
$$\left(\frac{\tilde{x}^3 + a\tilde{x} + b}{p}\right)$$
durchgeführt werden. Ist dieses Legendre-Symbol ungleich -1, so ist $\tilde{x}^3 + a\tilde{x} + b$ entweder Null oder ein Quadrat modulo p. In beiden Fällen gibt es eine Zahl $\tilde{y}$, so daß $\tilde{y}^2 = \tilde{x}^3 + a\tilde{x} + b$ ist.

  Folgender Algorithmus berechnet das Legendre-Symbol zu zwei Zahlen a' und p.

  Algorithmus 3.4 (Legendre-Symbol)

  Eingabe: *$a', p \in \mathbb{Z}$, p eine ungerade Primzahl, $0 \leq a' < p$.*
  Ausgabe: *Legendre-Symbol $\left(\frac{a'}{p}\right)$.*

  1. *Setze $k = 1$.*
  2. *Falls $a' = 0$, so terminiere mit Ergebnis* $\begin{cases} k, & \text{falls } p = 1; \\ 0, & \text{sonst}; \end{cases}$
     ansonsten setze $v = 0$.
  3. *Solange a' gerade, setze $v = v + 1$ und $a' = \frac{a'}{2}$.*
  4. *Falls v ungerade, setze $k = (-1)^{(p^2-1)/8} \cdot k$.*
  5. *Setze $k = (-1)^{(a'-1)(p-1)/4} \cdot k$, $r = a'$, $a' = p \bmod r$, $p = r$ und gehe zu 2.*

  Genaue Analysen dieses Algorithmus – insbesondere über die zugrundeliegende Theorie und ein Beweis der Korrektheit des Algorithmus – findet man in [Co91, Seite 22]. Dort findet man auch einen Beweis, daß die Laufzeit von Algorithmus 3.4 $O(\log(p)^2)$ ist. Dabei wird wie in Satz 3.1 berücksichtigt, daß die Zwischenergebnisse innerhalb der Schleife immer kleiner werden und daß Division durch 2 einem Shift entspricht.

- Die Berechnung einer Quadratwurzel zu einem gegebenen quadratischen Rest $\tilde{x}$ modulo einer Primzahl p kann mit Hilfe des RESSOL-Algorithmus von D. Shanks ([Sh72]) durchgeführt werden. Wir geben hier nur eine kurze Beschreibung dieses probabilistischen Algorithmus. Weitere Angaben – insbesondere über den theoretischen Hintergrund des Algorithmus und seine Korrektheit – findet man wiederum in [Co91, Seite 25].

  Algorithmus 3.5 (RESSOL-Algorithmus)

  Eingabe: *Primzahl* $p \geq 3$ *und quadratischer Rest* $a \bmod p$.
  Ausgabe: x *mit* $x^2 \equiv a \bmod p$.

  1. *Bestimme* l *und* q *so, daß* $p - 1 = 2^l q$ *und* q *ungerade.*
  2. *Wähle zufällig quadratischen Nichtrest* $n \in \mathbb{F}_p$.
  3. *Setze* $z = n^q \bmod p$, $y = z$, $r = l$, $x = a^{(q-1)/2} \bmod p$, $b = a \cdot x^2 \bmod p$, $x = a \cdot x \bmod p$.
  4. *Falls* $b \equiv 1 \bmod p$, *terminiere mit Ergebnis* x*;* *sonst finde minimales* $m \geq 1$ *mit* $b^{2^m} \equiv 1 \bmod p$.
  5. *Setze* $t = y^{2^{r-m-1}} \bmod p$, $y = t^2 \bmod p$, $r = m$, $x = x \cdot t \bmod p$, $b = b \cdot y \bmod p$ *und gehe zu Schritt 4.*

  Wir wollen nun die Laufzeit des RESSOL-Algorithmus 3.5 bestimmen. Da die Schleife in Schritt 4 maximal $O(\log(p))$-mal durchlaufen wird ([Co91, Seite 26]), beträgt die Laufzeit der Schritte 3 bis 5 $O(\log(p)^4)$. Wenn wir annehmen, daß wir in Schritt 2 k-mal zufällig eine Zahl $\tilde{x} \in \mathbb{F}_p$ wählen müssen, bis wir einen quadratischen Nichtrest gefunden haben, erhalten wir für Schritt 2 die Laufzeit $O\left(k \cdot \log(p)^2\right)$, da wir für jede gewählte Zahl $\tilde{x}$ durch Berechnung des Legendre-Symbols testen müssen, ob $\tilde{x}$ ein quadratischer Nichtrest ist. Also erhalten wir für den RESSOL-Algorithmus 3.5 insgesamt die Laufzeit $O(k \cdot \log(p)^2 + \log(p)^4)$, wobei k die Anzahl der Durchläufe von Schritt 2 ist.

  Unter Annahme der **Verallgemeinerten Riemannschen Vermutung** kann man sogar beweisen, daß die Wahl des quadratischen Nichtrests in Schritt 2 deterministisch in $O(\log(p)^4)$ Bitoperationen durchgeführt werden kann. Damit wird der RESSOL-Algorithmus 3.5 deterministisch und besitzt Laufzeit $O(\log(p)^4)$.

Um später die erwartete Laufzeit des Shanks Algorithmus herleiten zu können, müssen wir noch die Wahrscheinlichkeit bestimmen, mit der in Schritt 2 des RESSOL-Algorithmus 3.5 ein quadratischer Nichtrest gewählt wird. Dazu formulieren wir folgendes Lemma.

Lemma 3.1 *Die Wahrscheinlichkeit, daß eine zufällige Zahl* $x \in \mathbb{F}_p$ *ein quadratischer Nichtrest ist, ist* $\frac{p-1}{2p}$.

Beweis: Das Lemma folgt aus der Beobachtung, daß es genauso viele quadratische Nichtreste wie Reste gibt ([Ke82, Th. 1.1, Seite 35]). Damit erhalten wir unter Berücksichtigung der Null als Anzahl der quadratischen Nichtreste den Wert $\frac{p-1}{2}$. Hieraus folgt dann die Behauptung. ■

Als nächstes müssen wir die Wahrscheinlichkeit dafür bestimmen, daß bei zufälliger Wahl von $\tilde{x}$ der Wert $\tilde{x}^3 + a\tilde{x} + b$ ein Quadrat modulo p ist. Diese Wahrscheinlichkeit wird im folgenden Lemma abgeschätzt.

Lemma 3.2 *Die Wahrscheinlichkeit, eine Zahl $\tilde{x} \in \mathbb{F}_p$ zufällig so zu wählen, daß $\left(\frac{\tilde{x}^3+a\tilde{x}+b}{p}\right) \neq -1$ ist, ist mindestens*

$$\frac{p-3-2\sqrt{p}}{2p}.$$

Beweis: Nach Lemma 1.2 ist jede Zahl $\tilde{x} \in \mathbb{F}_p$ mit $\left(\frac{\tilde{x}^3+a\tilde{x}+b}{p}\right) = 1$ die x-Koordinante von genau zwei Punkten aus $E(\mathbb{F}_p)$. Da das Polynom $x^3 + ax + b$ höchstens drei Nullstellen modulo p besitzt, kann es weiterhin höchstens 3 Punkte mit y-Koordinate 0 geben. Weil Algorithmus 3.3 nach Konstruktion nie den Punkt $\mathcal{O}$ liefert und da nach dem Satz von Hasse 1.2 die Anzahl der Punkte der elliptischen Kurve mindestens $p + 1 - 2\sqrt{p}$ ist, gibt es also mindestens $p - 3 - 2\sqrt{p}$ Punkte mit y-Koordinate ungleich 0. Somit existieren mindestens

$$\frac{p-3-2\sqrt{p}}{2}$$

viele Zahlen $\tilde{x} \in \mathbb{F}_p$, die zu so einem Punkt führen. Da $\tilde{x}$ zufällig aus dem Körper $\mathbb{F}_p$ gewählt wird, ist die Wahrscheinlichlichkeit, einen solchen Wert $\tilde{x}$ zu finden, somit mindestens

$$\frac{p-3-2\sqrt{p}}{2p}.$$

■

Lemma 3.3 *Für alle Primzahlen $p \geq 53$ ist $\frac{p-3-2\sqrt{p}}{2p} \geq \frac{1}{3}$.*

Beweis: Wir betrachten die Funktion $f(x) = \frac{x-3-2\sqrt{x}}{2x}$. Dann verwenden wir, daß $f(53) = 0,3343375492$ und daß die Ableitung

$$f'(x) = \frac{3}{2x^2} + \frac{1}{2x^{3/2}}$$

für $x > 0$ positiv ist, d.h. die Funktion f monoton wachsend ist. Daraus folgt dann die Behauptung insbesondere für alle Primzahlen ≥ 53. ■

Bemerkung 3.2 *Die Voraussetzung $p \geq 53$ ist keine echte Einschränkung, denn wir sind nur an der Berechnung von $\#E(\mathbb{F}_p)$ für wirklich große Primzahlen p interessiert. Für Primzahlen kleiner als 100 kann die Ordnung der Punktgruppe schnell auch mit dem deterministischen Algorithmus 1.1 aus Kapitel 1 berechnet werden.*

Damit erhalten wir die folgende Laufzeit des Punktsuche-Algorithmus 3.3.

Lemma 3.4 *Die Punktsuche kann in Laufzeit $O\left((k_1 + k_2) \cdot \log(p)^2 + \log(p)^4\right)$ erfolgen, wobei k_1 die Anzahl der Schleifendurchläufe der Schritte 1-2 des Punktsuche-Algorithmus 3.3 und k_2 die Anzahl der Durchläufe des Schritts 2 des RESSOL-Algorithmus ist.*

Beweis: Da wir in jedem Durchlauf der Schritte 1-2 von Algorithmus 3.3 ein Legendre-Symbol berechnen müssen, benötigt dieser Teil des Algorithmus Zeit $O\left(k_1 \cdot \log(p)^2\right)$. Nach obiger Bemerkung benötigt der RESSOL-Algorithmus zusätzlich die Laufzeit $O\left(k_2 \cdot \log(p)^2 + \log(p)^4\right)$. Durch Addition dieser beiden Ergebnisse folgt die Behauptung des Lemmas. ■

3.3 Erwartete Laufzeit des ersten Teils

Nun werden wir die erwartete Laufzeit des ersten Teils des Shanks Algorithmus genauer untersuchen. Dabei klammern wir im Moment die Punktsuche aus und gehen davon aus, daß wir einen zufälligen Punkt $P \in E(\mathbb{F}_p)$ kennen. Der erste Teil des Algorithmus bestimmt dann in den Schritten 2-6 Vielfache der Punktordnung innerhalb des Hasse-Intervalls. Ist diese Punktordnung mindestens $\sqrt{p} - 1$, so wird der erste Teil auf alle Fälle verlassen, denn entweder haben wir die Gruppenordnung bestimmt oder wir fahren im zweiten Teil des Shanks Algorithmus fort.

Wir haben im vorigen Kapitel schon gezeigt, daß diese Schleife im ersten Teil des Shanks Algorithmus keine Endlosschleife ist, d.h. daß es auf jeder elliptischen Kurve einen Punkt der Ordnung mindestens $\sqrt{p} - 1$ geben muß. Zur Bestimmung der erwarteten Laufzeit des ersten Teils ist es natürlich von großem Interesse, wieviele solcher Punkte existieren. Denn damit können wir die Wahrscheinlichkeit bestimmen, mit der ein solcher Punkt in Schritt 1 gewählt wird und somit eine Aussage über die erwartete Anzahl der Schleifendurchläufe des ersten Teils machen. Dazu formulieren wir nun den folgenden Satz.

Satz 3.2 *Sei $E(\mathbb{F}_p) \cong \mathbb{Z}/n\mathbb{Z} \oplus \mathbb{Z}/m\mathbb{Z}$, wobei o.B.d.A. n ein Teiler von m ist. Dann ist die Wahrscheinlichkeit, bei zufälliger Wahl einen Punkt aus $E(\mathbb{F}_p)$ der Ordnung m*

zu erhalten, mindestens

$$\frac{\varphi(m)}{m},$$

wobei $\varphi(\ )$ die Eulersche - Phifunktion ist.

Beweis: Aus der Voraussetzung über den Typ der Punktgruppe $E(\mathbb{F}_p)$ und Bemerkung 2.4 folgt, daß es einen Punkt $P \in E(\mathbb{F}_p)$ mit Ordnung m und einen Punkt $Q \in E(\mathbb{F}_p)$ mit Ordnung n gibt, so daß

$$E(\mathbb{F}_p) \;=\; \{i \cdot Q + j \cdot P\ ;\ 0 \leq i < n, 0 \leq j < m\}$$

und $\langle P \rangle \cap \langle Q \rangle = \{\mathcal{O}\}$ ist.

Sei $H = k \cdot Q + l \cdot P$ ein beliebiger Punkt. Falls $\operatorname{ggt}(l, m) = 1$ gilt, so besitzt der Punkt H ebenfalls die Ordnung m, denn

$$\operatorname{ord}(H) = \operatorname{kgV}(\operatorname{ord}(k \cdot Q), \operatorname{ord}(l \cdot P)) = m.$$

Dabei beachte man, daß die Ordnung des Punktes $k \cdot Q$ nach Satz 2.1 ein Teiler der Ordnung n von Q ist. Da nach Theorem 3.6 in [Hun74, Seite 36] wegen $\operatorname{ggt}(l, m) = 1$ die Ordnung des Punktes $l \cdot P$ gleich m ist, ist das kleinste gemeinsame Vielfache der beiden Ordnungen genau m.

Damit erhalten wir immer dann einen Punkt $H = k \cdot Q + l \cdot P$ der Ordnung m, wenn wir k aus der Menge $\{0, 1, \ldots, n-1\}$ und l aus der Menge $\{i\ ;\ 1 \leq i < m,\ \operatorname{ggt}(i, m) = 1\}$ beliebig wählen. Aus Bemerkung 2.4 folgt weiterhin, daß alle diese Punkte verschieden sind. Zählen wir also die Anzahl der "guten" Kombinationen von k und l, so erhalten wir eine untere Abschätzung für die Anzahl der Punkte mit Ordnung m. Damit erhalten wir mindestens $\varphi(m) \cdot n$ viele solche Punkte, denn es gibt $\varphi(m)$ viele zu m teilerfremde Zahlen. Da die Punktgruppe der elliptischen Kurve E aber $\#E(\mathbb{F}_p) = m \cdot n$ viele Punkte enthält, erhalten wir als Wahrscheinlichkeit, daß ein zufällig gewählter Punkt die Ordnung m besitzt, somit mindestens

$$\frac{\varphi(m) \cdot n}{m \cdot n} = \frac{\varphi(m)}{m}.$$

■

Da wir nach Lemma 2.4 schon wissen, daß m sicherlich größer als $\sqrt{p} - 1$ ist, liefert uns obiger Satz 3.2 schon die gewünschte Wahrscheinlichkeit, mit der ein zufälliger Punkt zum Verlassen des ersten Teils des Shanks Algorithmus führt. Falls m sogar größer als die Länge des Hasse-Intervalls, also $4\sqrt{p}$, sein sollte, kennen wir sogar die Wahrscheinlichkeit, mit der ein Durchlauf der Schritte 1-6 zur Bestimmung der Gruppenordnung führt. Leider kennen wir i.a. jedoch nicht die Größe von m. Daher wollen wir den Wert $\frac{\varphi(m)}{m}$ nun in Abhängigkeit von der Charakteristik des Primkörpers abschätzen. Dazu formulieren wir den folgenden Satz.

Satz 3.3 *Für hinreichend großes x gibt es eine Konstante $c > 0$, so daß*

$$\frac{\varphi(x)}{x} \geq c \cdot \frac{1}{\log\log(x)}.$$

Beweis: Sei $x \in \mathbb{N}$ beliebig. Nach [Ke82, Seite 109] gilt dann für $\varphi(x)$

$$\varphi(x) \; = \; x \cdot \prod_{p|x,\, p\in\mathbb{P}} \left(1 - \frac{1}{p}\right). \tag{3.1}$$

Dabei sei $\mathbb{P}$ die Menge aller Primzahlen. Da nur endlich viele Primzahlen x teilen, können wir (3.1) mit Hilfe der i-ten Primzahl p_i abschätzen als

$$\prod_{p|x, p\in\mathbb{P}} \left(1 - \frac{1}{p}\right) \quad \geq \quad \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right),$$

wobei k so gewählt ist, daß $\prod_{i=1}^{k} p_i \geq x$ ist. Da $\prod_{i=1}^{k} p_i \geq p_1^k = 2^k$ ist, können wir k als $\lceil \log(x) \rceil$ wählen. Damit ergibt sich

$$\frac{\varphi(x)}{x} \quad \geq \quad \prod_{i=1}^{\lceil\log(x)\rceil} \left(1 - \frac{1}{p_i}\right). \tag{3.2}$$

Zur Bestimmung dieses Produkts benutzen wir den folgenden Satz.

Satz 3.4 *Für $y > 1$ existiert eine Konstante C, so daß*

$$\prod_{2\leq p\leq y} \left(1 - \frac{1}{p}\right) \; > \; \frac{e^{-C}}{\log(y)} \left(1 - \frac{1}{\log\log(y)}\right).$$

Beweis: [RoSc62, (3.27)] ■

Verwenden wir diesen Satz, so erhalten wir aus (3.2)

$$\frac{\varphi(x)}{x} \geq \prod_{i=1}^{\lceil\log(x)\rceil} \left(1 - \frac{1}{p_i}\right) > \frac{e^{-C'}}{\log\log(x)} \left(1 - \frac{1}{\log\log\log(x)}\right).$$

Da für hinreichend große Werte von x der Wert $\left(1 - \frac{1}{\log\log\log(x)}\right) > c'$ für eine Konstante $c' > 0$ ist, erhalten wir hieraus die Aussage des Satzes. ■

Bemerkung 3.3 *Bei allen Logarithmen in diesem Satz handelt es sich immer um den Logarithmus zur Basis zwei. Dies ist keine Einschränkung, da es für jede Basis b eine Konstante c gibt, so daß*

$$\log_b(x) = c \cdot \log_2(x).$$

Korollar 3.1 *Sei $E(\mathbb{F}_p) \cong \mathbb{Z}/n\mathbb{Z} \oplus \mathbb{Z}/m\mathbb{Z}$ mit $n \mid m$. Für hinreichend großes p gibt es eine Konstante $c > 0$, so daß wir bei zufälliger Wahl eines Punktes mit mindestens der Wahrscheinlichkeit*

$$\frac{c}{\log\log(p)}$$

einen Punkt der Ordnung m finden.

Beweis: Nach Satz 3.2 wissen wir, daß die Wahrscheinlichkeit, bei zufälliger Wahl einen Punkt der Ordnung m zu treffen, mindestens

$$\frac{\varphi(m)}{m}$$

ist. Weiterhin liefert Satz 3.3 die Abschätzung

$$\frac{\varphi(m)}{m} \geq \frac{c}{\log\log(m)}.$$

Außerdem – da m sicherlich höchstens so groß wie die Gruppenordnung ist – gilt nach dem Satz von Hasse, daß

$$m \leq p + 1 + 2\sqrt{p} \leq c' \cdot p$$

für eine Konstante c' ist. Damit erhalten wir

$$\frac{\varphi(m)}{m} \geq \frac{c}{\log\log(c' \cdot p)}.$$

Beachten wir, daß es Konstanten c'' und c''' gibt, so daß

$$\begin{aligned} \log\log(c' \cdot p) = \log\left(\log(c') + \log(p)\right) &\leq \log(c'' \cdot \log(p)) \\ &= \log(c'') + \log\log(p) \\ &\leq c''' \cdot \log\log(p) \end{aligned}$$

ist, so erhalten wir das gewünschte Ergebnis

$$\frac{\varphi(m)}{m} \geq \frac{\tilde{c}}{\log\log(p)}.$$

■

Damit haben wir die Voraussetzungen geschaffen, um die erwartete Laufzeit des ersten Teils des Shanks Algorithmus zu bestimmen.

Satz 3.5 *Die erwartete Laufzeit des ersten Teils des Shanks Algorithmus 2.1 ist für hinreichend großes p*

$$O\left(\sqrt[4]{p} \cdot \log(p)^2 \cdot (\log\log(p))^2\right).$$

Beweis: Als erstes wollen wir die erwartete Laufzeit des ersten Teils des Shanks Algorithmus 2.1 bestimmen, wenn wir insgesamt genau n Schleifendurchläufe benötigen. Dabei zählen wir die Anzahl i der Durchläufe der Schritte 1-2 im Punktsuche-Algorithmus 3.3, die Anzahl j der Durchläufe des Schritts 2 im RESSOL-Algorithmus 3.5 und die Anzahl k der Durchläufe der Schritte 2-6 des eigentlichen Shanks Algorithmus zusammen, d.h. $i + j + k$ soll genau n sein. Nun wollen wir die einzelnen Mißerfolgswahrscheinlichkeiten bestimmen.

Nach Lemma 3.1 ist für $p \geq 3$ die Wahrscheinlichkeit, einen quadratischen Nichtrest zu finden, mindestens $\frac{p-1}{2p} \geq \frac{1}{3}$. Außerdem wissen wir nach Lemma 3.3 schon, daß wir auch einen "guten" x-Wert mit Wahrscheinlichkeit ebenfalls mindestens $\frac{1}{3}$ finden (jedenfalls für hinreichend großes p). Die Wahrscheinlichkeit, daß wir genau i Schleifendurchläufe im Punktsuche-Algorithmus und genau j Durchläufe im RESSOL-Teil benötigen, ist also maximal

$$\left(\frac{2}{3}\right)^{i-1} \cdot \left(\frac{2}{3}\right)^{j-1} = \left(\frac{2}{3}\right)^{i+j-2}.$$

Da wir bei k-maligem Durchlaufen der Schritte 2-6 des Shanks Algorithmus insgesamt k-mal den Rumpf des RESSOL-Algorithmus (d.h. den RESSOL-Algorithmus ohne Schritt 2) durchlaufen, brauchen wir nach Lemma 3.4 für die Punktsuche, d.h. Schritt 1 des Shanks Algorithmus, dann insgesamt Zeit $O((i + j) \cdot \log(p)^2 + k \cdot \log(p)^4)$.

Nach Korollar 3.1 ist die Wahrscheinlichkeit, die Schleife 2-6 im Shanks Algorithmus 2.1 genau k-mal zu durchlaufen, maximal

$$\left(1 - \frac{c'}{\log\log(p)}\right)^{k-1}.$$

Sei T_S die Zeit, die wir für einen Durchlauf der Schritte 2-6 benötigen.

Damit können wir nun die erwartete Laufzeit bei genau n Schleifendurchläufen bestimmen. Offensichtlich müssen noch die Bedingungen $i, j \geq k$ und $i, j, k \geq 1$ gelten. Damit erhalten wir als Erwartungswert

$$E_n \leq \tag{3.3}$$
$$c \cdot \sum_{\substack{i+j+k=n;\\ i,j\geq k; i,j,k\geq 1}} \left(\frac{2}{3}\right)^{i+j-2} \cdot \left(1 - \frac{c'}{\log\log(p)}\right)^{k-1} \cdot \Big((i + j) \cdot \log(p)^2 + k \cdot \log(p)^4 + k \cdot T_S\Big).$$

Durch Umformen und Zusammenfassen können wir dies abschätzen als

$$E_n \leq c \cdot \sum_{\substack{i'+k=n; i'\geq 2;\\ k\geq 1; i'\geq k/2}} \left(\frac{2}{3}\right)^{i'-2} \cdot \left(1 - \frac{c'}{\log\log(p)}\right)^{k-1} \cdot \Big(i' \cdot \log(p)^2 + k \cdot (T_S + \log(p)^4)\Big).$$

Durch Ausmultiplizieren der Klammer erhalten wir daraus

$$E_n \leq c \cdot \log(p)^2 \cdot \sum_{i=2}^{n} i \cdot \left(\frac{2}{3}\right)^{i-2} + c \cdot (T_S + \log(p)^4) \cdot \sum_{k=1}^{n} k \cdot \left(1 - \frac{c'}{\log\log(p)}\right)^{k-1}. \tag{3.4}$$

Die erwartete Laufzeit des ersten Teils des Shanks Algorithmus 2.1 erhalten wir nun, indem wir den Grenzübergang $n \to \infty$ durchführen. Es ist jedoch nicht klar, ob wir dabei wirklich ein endliches Ergebnis erhalten.

Lemma 3.5 *Für $|x| < 1$ konvergiert die Summe $\sum_{i=0}^{\infty}(i+1) \cdot x^i$ und es gilt*

$$\sum_{i=0}^{\infty}(i+1) \cdot x^i = \frac{1}{(1-x)^2}.$$

Beweis: (Lemma)
Die Konvergenz der Summe kann man leicht mit Hilfe des Quotientenkriteriums zeigen. Zur Berechnung des Summenwertes beachte man, daß für die n-te Partialsumme S_n gilt:

$$\begin{aligned} S_n &= \sum_{i=0}^{n}(i+1) \cdot x^i \\ &= \sum_{i=0}^{n} \frac{d}{dx} x^{i+1} \\ &= \frac{d}{dx} \sum_{i=1}^{n+1} x^i. \end{aligned}$$

Rechnen wir diese Partialsumme nach der Formel $\sum_{i=0}^{n} x^i = \frac{1-x^{n+1}}{1-x}$ aus und leiten den entstehenden Term ab, so erhalten wir nach dem Grenzübergang $n \to \infty$ die Behauptung.
■ (Lemma)

Wenden wir dieses Lemma nun auf (3.4) an. Dann gilt für die erste Summe in (3.4), daß sie einen endlichen Wert besitzt. Für die zweite Summe erhalten wir mit diesem Lemma den Wert

$$\sum_{k=1}^{\infty} k \cdot \left(1 - \frac{c'}{\log\log(p)}\right)^{k-1} = \frac{1}{\left(1 - \left(1 - \frac{c}{\log\log(p)}\right)\right)^2} = \frac{(\log\log(p))^2}{c^2}.$$

Also erhalten wir als erwartete Laufzeit des ersten Teils des Shanks Algorithmus

$$c' \cdot \log(p)^2 + c'' \cdot (\log\log(p))^2 \cdot (T_S + \log(p)^4) \tag{3.5}$$

mit Konstanten c' und c''.

Nun bleibt noch die Bestimmung von T_S, d.h. der Laufzeit eines Durchlaufs der Schritte 2-6 des Shanks Algorithmus 2.1. Dazu untersuchen wir die einzelnen Schritte getrennt:

- In Schritt 2 führen wir $\lceil 2\sqrt[4]{p}\,\rceil$ viele Punktadditionen durch. Da nach Proposition 3.1 die Punktaddition $O(\log(p)^2)$ viele Bitoperationen benötigt, können wir diesen Schritt in Zeit $O(\sqrt[4]{p} \cdot \log(p)^2)$ ausführen (dabei nehmen wir an, daß das Abspeichern in der Tabelle T in Zeit $O(1)$ durchgeführt werden kann).

- Die Berechnung von $k \cdot Q$ in Schritt 3 kann nach Proposition 3.2 in Zeit
$$O(\log(\sqrt[4]{p}) \cdot \log(p)^2) = O(\log(p)^3)$$
vielen Bitoperationen erfolgen.

- Die Berechnung des Punktes in Schritt 4 kann jeweils durch eine Punktaddition aus dem im vorherigen Schleifendurchlauf berechneten Punkt erfolgen. Damit erhalten wir wie in Schritt 2 Laufzeit $O(\sqrt[4]{p} \cdot \log(p)^2)$ (damit berücksichtigt man auch das Suchen in der Tabelle, was bei linearer Suche Zeit $O(\sqrt[4]{p})$ benötigt).

- Die Schritte 5 und 6 bestehen nur aus einigen Multiplikationen und Additionen von Zahlen der Größe $O(\log(p))$. Sie können also in $O(\log(p)^2)$ vielen Bitoperationen durchgeführt werden.

Insgesamt erhalten wir also als Laufzeit eines Durchlaufs der Schritte 2 bis 6 den Wert $T_S = O(\sqrt[4]{p} \cdot \log(p)^2)$. Damit ergibt sich als erwartete Laufzeit des ersten Teil
$$O(\sqrt[4]{p} \cdot \log(p)^2 \cdot (\log\log(p))^2).$$

■

Bemerkung 3.4 *In der Praxis führen wir in Schritt* 4 *natürlich keine lineare Suche durch. Wir sortieren die Tabelle und können dann viel schneller suchen. Bei diesen theoretischen Betrachtungen spielt dies jedoch keine Rolle. Die praktische Implementierung wird genauer im folgenden Kapitel beschrieben.*

Bemerkung 3.5 *Wir haben schon vorher bemerkt, daß in fast allen Fällen der erste Teil des Algorithmus ausreicht, um die Gruppenordnung zu bestimmen. Experimentell wurde außerdem festgestellt, daß fast immer schon die Wahl des ersten Punktes ausreicht, um die Gruppenordnung zu bestimmen. Dies wird einsichtig, wenn man die Verteilung der verschiedenen Typen betrachtet. Für kleine Primzahlen p wurde der Typ aller möglichen elliptischen Kurven in dem Primkörper $\mathbb{F}_p$ berechnet. Dabei wurde beispielsweise festgestellt, daß von den 1640 elliptischen Kurven über dem Primkörper $\mathbb{F}_{41}$ 1370 eine zyklische Punktgruppe besitzen. Weiterhin besitzen noch 230 weitere elliptische Kurven den Typ $(x, 2)$. Also existieren für 97.5 Prozent aller elliptischen Kurven über dem Primkörper $\mathbb{F}_{41}$ Punkte mit sehr großer Ordnung.*

Jedoch sollte man immer beachten, daß dies nicht immer der Fall ist. Als Beispiel können wir die Kurve
$$E : y^2 = x^3 + x \qquad \textit{im Primkörper } \mathbb{F}_{65537}$$
angeben. Diese Kurve besitzt den Typ
$$E(\mathbb{F}_{65537}) \cong \mathbb{Z}/256\mathbb{Z} \oplus \mathbb{Z}/256\mathbb{Z}$$

und daher kann man alleine mit Hilfe des ersten Teils die Gruppenordnung nicht eindeutig bestimmen. Dies ist also ein Beispiel für eine Kurve, für die man den zweiten Teil zwingend benötigt.

3.4 Erwartete Laufzeit des zweiten Teils

Im folgenden werden wir die erwartete Laufzeit des zweiten Teils des Shanks Algorithmus 2.1 herleiten.

Sei also im folgenden ein Punkt P vorgegeben, dessen Ordnung l die Bedingung $\sqrt{p} - 1 \leq l < 4\sqrt{p}$ erfüllt und der die Gruppenordnung $\#E(\mathbb{F}_p)$ noch nicht eindeutig bestimmt. Wenn wir in den zweiten Teil des Shanks Algorithmus 2.1 gelangen, ist immer sichergestellt, daß wir so einen Punkt kennen. Wir wollen jetzt eine Schranke für die Wahrscheinlichkeit herleiten, mit der in Schritt 7 ein anderer Punkt P' gewählt wird, der nicht zu einer Bestimmung der Gruppenordnung führt. Als Hilfsmittel zur Lösung dieser Aufgabe benötigen wir folgenden Satz.

Satz 3.6 *Sei P ein Punkt der elliptischen Kurve E über dem Primkörper $\mathbb{F}_p$ und sei $k \in \mathbb{N}$. Dann ist die Anzahl der Punkte $Q \in E(\mathbb{F}_p)$ mit $k \cdot Q \in \langle P \rangle$ höchstens $k^2 \cdot ord(P)$.*

Beweis: Aus Bemerkung 2.4 wissen wir, daß die Punktgruppe $E(\mathbb{F}_p)$ von zwei Punkten P' und Q' erzeugt wird, daß also

$$E(\mathbb{F}_p) = \langle P' \rangle \oplus \langle Q' \rangle$$

ist. Daher gibt es Zahlen k_P und l_P, so daß

$$P = k_P \cdot P' + l_P \cdot Q'.$$

Für festes i wollen wir nun die Anzahl der Punkte Q abschätzen, für die $k \cdot Q = i \cdot P$ gilt. Dazu setzen wir Q allgemein an als

$$Q = k_Q \cdot P' + l_Q \cdot Q'$$

mit beliebigen ganzen Zahlen $0 \leq k_Q < \text{ord}\,(P')$ und $0 \leq l_Q < \text{ord}\,(Q')$. Da wir annehmen, daß $k \cdot Q = i \cdot P$ ist, müssen die folgenden Kongruenzen für k_Q bzw. l_Q gelten:

$$k \cdot k_Q \equiv i \cdot k_P \bmod \text{ord}\,(P') \tag{3.6}$$
$$k \cdot l_Q \equiv i \cdot l_P \bmod \text{ord}\,(Q') \tag{3.7}$$

Dabei beachte man, daß $i \cdot k_P$ bzw. $i \cdot l_P$ eine Konstante ist. Nach Satz 6.1 aus [Ke82, Seite 29] gilt für eine Kongruenz der Form $k \cdot x \equiv c \bmod n$, daß es höchstens k verschiedene

Lösungen für x gibt. Damit gibt es also maximal k viele Lösungen k_Q mit (3.6) und k viele Lösungen l_Q mit (3.7). Da die "Parameter" k_Q und l_Q eines Punktes Q mit $k \cdot Q = i \cdot P$ beide Bedingungen (3.6) und (3.7) erfüllen müssen, kann es für festes i maximal k^2 viele solche Punkte Q geben. Die Zahl k^2 erhalten wir dabei als die Anzahl der Möglichkeiten, die verschiedenen Lösungen k_Q und l_Q zu kombinieren. Jede dieser Kombinationen führt zu einem Punkt Q.

Nun betrachten wir alle möglichen Werte für i. Es ist direkt einsichtig, daß es maximal $\mathrm{ord}\,(P)$ viele Möglichkeiten für i gibt (da wir ohne Einschränkung i aus der Menge $\{0, \ldots, \mathrm{ord}\,(P) - 1\}$ wählen können). Da es für jedes i maximal k^2 viele Punkte Q mit $k \cdot Q = i \cdot P$ gibt, erhalten wir also insgesamt maximal $k^2 \cdot \mathrm{ord}\,(P)$ viele Punkte Q mit $k \cdot Q \in \langle P \rangle$ und die Behauptung ist bewiesen. ∎

Bemerkung 3.6 *Diese Proposition wird im folgenden benutzt, um die Anzahl der Punkte Q mit $ord_{E(\mathbb{F}_p)/\langle P \rangle}(Q) = k$ abzuschätzen, denn für diese Punkte gilt insbesondere, daß $k \cdot Q \in \langle P \rangle$ ist. Allerdings gilt noch die schärfere Bedingung, daß für Q mit $ord_{E(\mathbb{F}_p)/\langle P \rangle}(Q) = k$ auch $i \cdot Q \notin \langle P \rangle$ für alle $0 < i < k$ gelten muß. Dies wird nicht berücksichtigt, da wir nur an einer oberen Abschätzung der Anzahl der Punkte Q mit $ord_{E(\mathbb{F}_p)/\langle P \rangle}(Q) = k$ interessiert sind.*

Damit können wir nun eine Aussage über die Wahrscheinlichkeit herzuleiten, daß wir in Schritt 7 einen Punkt P' wählen, der nicht zur Bestimmung der Gruppenordnung $\#E(\mathbb{F}_p)$ führt. Dies geschieht in der folgenden Proposition.

Proposition 3.3 *Sei $P \in E(\mathbb{F}_p)$ ein Punkt, dessen Ordnung l die Bedingung $\sqrt{p} - 1 \leq l < 4\sqrt{p}$ erfüllt. Dann ist die Wahrscheinlichkeit, daß ein zufällig gewählter Punkt Q nicht zu der Bestimmung der Gruppenordnung $\#E(\mathbb{F}_p)$ führt, maximal*

$$c \cdot \frac{1}{\sqrt{p}}$$

mit einer Konstanten c.

Beweis: Wir haben schon in Lemma 2.8 gesehen, daß $x = \mathrm{ord}_{E(\mathbb{F}_p)/\langle P \rangle}(Q) \cdot l$ ein Teiler der Gruppenordnung ist. Außerdem folgt nach dem Satz von Hasse 1.2, daß die Gruppenordnung eindeutig bestimmt ist, wenn $x > 4\sqrt{p}$ und damit größer als die Intervallänge ist. Wie man leicht einsieht, führt wegen $l \geq \sqrt{p} - 1$ ein Punkt Q mit $\mathrm{ord}_{E(\mathbb{F}_p)/\langle P \rangle}(Q) \geq 5$ zur Bestimmung der Gruppenordnung.

In Satz 3.6 haben wir gezeigt, daß die Anzahl der Punkte Q mit $\mathrm{ord}_{E(\mathbb{F}_p)/\langle P \rangle}(Q) = k$ maximal $k^2 \cdot \mathrm{ord}\,(P)$ sein kann. Also kann man die Anzahl S der "schlechten" Punkte, d.h. der Punkte Q mit ord $_{E(\mathbb{F}_p)/\langle P\rangle}(Q) < 5$, abschätzen durch

$$S \leq (1+4+9+16)\cdot \mathrm{ord}\,(P) = 30\cdot \mathrm{ord}\,(P).$$

Da wir uns im zweiten Teil des Shanks Algorithmus befinden und demnach $\mathrm{ord}(P) < 4\sqrt{p}$ ist, erhalten wir

$$S < 120\sqrt{p}.$$

Damit gilt für die Wahrscheinlichkeit h, einen solchen "schlechten" Punkt zu wählen, die Abschätzung

$$h < \frac{120\sqrt{p}}{p+1-2\sqrt{p}} < \frac{120\sqrt{p}}{\sqrt{p}(\sqrt{p}-2)}$$

(dabei wird die Anzahl der Punkte der Punktgruppe $E(\mathbb{F}_p)$ wieder mit der unteren Hasse-Grenze $p+1-2\sqrt{p}$ abgeschätzt). Kürzen wir diese Gleichung, so erhalten wir

$$h < \frac{120}{\sqrt{p}-2} < c\cdot\frac{1}{\sqrt{p}}$$

für eine Konstante c. Daraus folgt die Behauptung. ■

Damit können wir nun auch eine erwartete Laufzeit des zweiten Teils des Shanks Algorithmus 2.1 bestimmen. Dabei ist die Vorgehensweise analog zu der in Satz 3.5.

Satz 3.7 *Die erwartete Laufzeit des zweiten Teils des Shanks Algorithmus 2.1 ist*

$$O(\sqrt[4]{p}\cdot\log(p)^3).$$

Beweis: Sei diesmal T_S die Laufzeit eines Durchlaufs der Schritte 8-11 im Shanks Algorithmus 2.1. Dann bestimmen wir analog zum Beweis von Satz 3.5 die erwartete Laufzeit bei genau n Schleifendurchläufen. Da wir auch im zweiten Teil wieder einen Punkt zufällig wählen müssen und demnach wieder auf den probabilistischen Algorithmus 3.3 angewiesen sind, betrachten wir n wieder als die Summe der Anzahl der Durchläufe der Schritte 1-2 in Algorithmus 3.3, des Schritts 2 des RESSOL-Algorithmus 3.5 und der Schritte 8-11 des Shanks Algorithmus 2.1. Dann erhalten wir analog zum Beweis von Satz 3.5, daß die erwartete Laufzeit bei genau n Schleifendurchläufen abgeschätzt werden kann durch

$$E_n \quad\leq\quad c'\cdot \sum_{\substack{i+k=n;i\geq 2;\\ k\geq 1;i\geq k/2}} \left(\frac{2}{3}\right)^{i-2}\cdot\left(\frac{c}{\sqrt{p}}\right)^{k-1}\cdot\left(i\cdot\log(p)^2+k\cdot(T_S+\log(p)^4)\right). \qquad (3.8)$$

Durch Ausmultiplizieren und den Grenzübergang $n\to\infty$ erhalten wir mit Hilfe von Lemma 3.5

$$E_\infty \leq \tilde{c}\cdot\log(p)^2 + \frac{c'}{\left(1-\frac{c}{\sqrt{p}}\right)^2}\cdot(T_S+\log(p)^4).$$

Wenn wir den Term $\frac{c'}{(1-c/\sqrt{p})^2}$ berechnen, so erhalten wir

$$\frac{c'}{\left(1-\frac{c}{\sqrt{p}}\right)^2} \leq \frac{c' \cdot \sqrt{p}}{\sqrt{p}-2c},$$

was für hinreichend große Primzahlen p kleiner als die Konstante $\tilde{c}$ ist. Also erhalten wir als erwartete Laufzeit des zweiten Teils

$$O\left(\log(p)^2 + \log(p)^4 + T_S\right).$$

Nun bleibt noch die Aufgabe, die Laufzeit T_S eines Durchlaufs der Schritte 8-11 des Shanks Algorithmus 2.1 zu bestimmen. Dazu gehen wir wieder die einzelnen Schritte durch.

- In Schritt 8 berechnen wir die Menge L. Da wegen $l \geq \sqrt{p}-1$ die Menge L maximal 5 Elemente enthalten kann, ist es offensichtlich, daß dies in Zeit $O(\log(p)^3)$ erfolgen kann (wir müssen maximal für fünf verschiedene Zahlen $y \leq p+1+2\sqrt{p}$ testen, ob $y \cdot P' = \mathcal{O}$ ist).
- Da die ausgewählte Zahl x nach oben durch die obere Hasse-Grenze beschränkt ist, kann sie maximal $O(\log(p))$ viele Teiler besitzen. Also berechnen wir höchstens $O(\log(p))$ viele Vielfache eines Punktes. Damit erhalten wir die Laufzeit $O(\log(p)^4)$ für Schritt 9.
- In Schritt 10 wird wieder eine Babystep-Giantstep Strategie benutzt. Diesmal ist die Größe der Tabelle T' und somit die Anzahl der Baby- bzw. Giantsteps gegeben durch $\sqrt{l}$. Dies ist von der Größenordnung $O(\sqrt[4]{p})$, denn wir kennen die obere Abschätzung $l < 4\sqrt{p}$. Analog zum ersten Teil des Shanks Algorithmus 2.1 erkennen wir leicht, daß daher die Schritte 10.1 bis 10.4 die Laufzeit $O(\sqrt[4]{p}\cdot\log(p)^2)$ benötigen. Da wir diese Babystep-Giantstep Strategie maximal für jeden Teiler der Ordnung des Punktes P' anwenden müssen und da wir damit für die Anzahl der Durchläufe der Schleife 10.1-10.4 die obere Schranke $O(\log(p))$ kennen, erhalten wir für Schritt 10 die Laufzeit $O(\sqrt[4]{p} \cdot \log(p)^3)$.
- Der letzte Schritt des zweiten Teils besteht wiederum nur aus einigen Multiplikationen bzw. Additionen von Zahlen der Größe $O(\log(p))$ und besitzt deshalb Laufzeit $O(\log(p)^2)$.

Damit erhalten wir als Laufzeit eines Durchlaufs der Schritte 8 bis 11

$$T_S = O(\sqrt[4]{p} \cdot \log(p)^3).$$

Somit können wir nun die erwartete Laufzeit des zweiten Teils des Shanks Algorithmus 2.1 berechnen als

$$O(\log(p)^2 + \log(p)^4 + \sqrt[4]{p} \cdot \log(p)^3) = O(\sqrt[4]{p} \cdot \log(p)^3).$$

■

3.5 Erwartete Laufzeit des Shanks Algorithmus

In diesem Abschnitt werden wir die Ergebnisse der beiden vorherigen Abschnitte zusammensetzen und so eine erwartete Laufzeit für den gesamten Shanks Algorithmus 2.1 bestimmen.

Satz 3.8 *Der Shanks Algorithmus 2.1 besitzt erwartete Laufzeit* $O(\sqrt[4]{p} \cdot \log(p)^3)$.

Beweis: Dieser Satz folgt direkt aus den Ergebnissen von Satz 3.5 und 3.7. Dort haben wir die erwarteten Laufzeiten $O(\sqrt[4]{p} \cdot \log(p)^2 \cdot (\log\log(p))^2)$ für den ersten Teil und $O(\sqrt[4]{p} \cdot \log(p)^3)$ für den zweiten Teil des Shanks Algorithmus bewiesen. Aus Kombination dieser beiden Ergebnisse folgt direkt die Behauptung des Satzes 3.8. ∎

Mit diesem Satz haben wir ein theoretisches Ergebnis für die zu erwartende Laufzeit des Shanks Algorithmus 2.1 bewiesen. Leider müssen wir dabei davon ausgehen, daß der zweite Teil des Shanks Algorithmus immer auch durchlaufen wird. Es sind zur Zeit keine Wahrscheinlichkeitsabschätzungen bekannt, für wie viele Kurven über einem festen Primkörper wir den zweiten Teil wirklich benötigen. Ein experimentelles Ergebnis ist die Tatsache, daß für "fast alle" elliptischen Kurven der erste Teil des Shanks Algorithmus völlig ausreicht, um die Gruppenordnung $\#E(\mathbb{F}_p)$ zu bestimmen. Betrachten wir hingegen eine feste elliptische Kurve über verschiedenen Primkörpern, so existiert der folgende Satz.

Satz 3.9 (Serre)
Sei E eine elliptische Kurve über $\mathbb{Q}$. Unter Annahme der Verallgemeinerten Riemannschen Vermutung gibt es eine Konstante c_E, so daß

$$\lim_{x\to\infty} \frac{f(x,\mathbb{Q})}{x/\log(x)} \quad = \quad c_E$$

gilt. Dabei ist $f(x,\mathbb{Q})$ die Anzahl der Primzahlen $p \leq x$, für die E gute Reduktion modulo p besitzt und für die $E(\mathbb{F}_p)$ zyklisch ist.

Beweis: [Mu83] ∎

Bemerkung 3.7 *Der Begriff* **gute Reduktion** *bedeutet, daß die Diskriminante der Kurve $\Delta = -16 \cdot (4a^3 + 27b^2)$ nicht von p geteilt wird. Dann ist die reduzierte Kurve E', die wir aus E durch Reduktion der Kurvenparameter a und b modulo p erhalten, wirklich eine elliptische Kurve über dem Primkörper $\mathbb{F}_p$.*

Dieser Satz ist ein Indiz, daß eine feste elliptische Kurve über "fast allen" Primkörpern zyklisch ist. Für solche Primkörper erhalten wir aber eine sehr hohe Wahrscheinlichkeit, einen Punkt genügend großer Ordnung auf der elliptischen Kurve zu finden, d.h. der Algorithmus bricht mit hoher Wahrscheinlichkeit schon im ersten Teil ab.

3.6 Platzbedarf des Shanks Algorithmus

In diesem letzten kleinen Abschlußabschnitt wollen wir noch die Platzkomplexität des Shanks Algorithmus bestimmen. Dies ist nach den Vorbetrachtungen kein großes Problem mehr und wird in folgendem Satz beschrieben.

Satz 3.10 *Der Shanks Algorithmus 2.1 benötigt* $O(\sqrt[4]{p} \cdot \log(p))$ *Bits zur Bestimmung der Gruppenordnung* $\#E(\mathbb{F}_p)$ *einer elliptischen Kurve* E *über einem Primkörper* $\mathbb{F}_p$.

Beweis: Es ist aus der Beschreibung des Shanks Algorithmus 2.1 direkt ersichtlich, daß der dominierende Platzanteil zur Speicherung der beiden Tabellen T und T' benötigt wird. Außer diesen Tabellen benötigen wir zusätzlich nur noch Platz für eine konstante Anzahl von Hilfswerten.

Von beiden Tabellen T und T' wissen wir schon, daß sie $O(\sqrt[4]{p})$ viele Punkte enthalten. Ein Punkt besteht aus einer x- und einer y-Koordinate, deren mögliche Einträge beide durch p beschränkt sind. Daher benötigen wir zur Abspeicherung eines Punktes Platz $O(\log(p))$. Durch Kombination dieser beiden Beobachtungen folgt die Behauptung. ■

Kapitel 4

Die Implementierung des Shanks Algorithmus

Wir haben den Shanks Algorithmus 2.1 implementiert und praktische Ergebnisse damit berechnet. Diese Implementierung und die dabei angewandten Tricks sollen in diesem Kapitel beschrieben werden. Unsere Implementierung des Shanks Algorithmus erfolgte in der Programmiersprache ‘C’. Da wir das Ziel hatten, in Primkörpern bis zu der Charakteristik ungefähr 10^{25} zu arbeiten, konnten wir als Datenstruktur für ganze Zahlen nicht den C-Datentyp ‘long integer’ benutzen. Stattdessen haben wir uns der Hilfe des Computeralgebrasystems PARI bedient. Wir bringen daher jetzt zuerst eine kurze Beschreibung von PARI.

4.1 Einführung in PARI

Das Computeralgebrasystem PARI aus Bordeaux kann benutzt werden, um mit langen Zahlen und Polynomen zu rechnen. Es stellt eine Bibliothek von Funktionen und Datentypen zur Verfügung, die zu jedem C-Programm dazugebunden werden kann. Dann kann man, nach einigen organisatorischen Vorbereitungen, die in dem C-Programm erledigt werden müssen, mit diesen PARI-Objekten rechnen.

PARI speichert alle eigenen Objekte auf einem großen Stack. Dieser Stack muß am Anfang des C-Programms mit Hilfe der Funktion “init()” initialisiert werden. Dabei wird Speicherplatz für diesen Stack reserviert (die Größe des Stacks ist ein Eingabewert von init) und es werden einige wichtige Konstanten auf den Stack abgelegt. Bei späteren Rechnungen werden auch alle Objekte des Benutzers wie etwa Polynome oder lange Zahlen auf diesem Stack gespeichert. Der Benutzer kann auf solche Objekte durch Variablen vom Typ GEN zugreifen. Dabei handelt es sich um nichts anderes als Pointer auf eine Adresse im Stack. Nur intern unterscheidet PARI mit Hilfe einer internen Typnummer zwischen Integern, modularen Zahlen, Polynomen und den anderen zugelassenen Typen.

Im folgenden werden wir die für uns wichtigen internen Typen vorstellen. PARI garantiert, daß bei Operationen zwischen erlaubten Typen das Ergebnis die Typkonventionen erfüllt. Beispielsweise wird nach Operationen auf modularen Zahlen das Ergebnis automatisch reduziert. Ein wichtiges Prinzip von PARI ist der rekursive Aufbau von Datentypen. Der Rekursionsanfang wird dabei durch die Darstellung von langen Zahlen gegeben. Lange Zahlen werden in PARI als konsekutive Folge von 32-Bit Speicherstellen auf dem Stack dargestellt. In den ersten beiden Speicherzellen befinden sich dabei organisatorische Daten wie die Anzahl der benutzten Speicherzellen oder das Vorzeichen. Lange Zahlen besitzen die interne Typ-Nummer 1.

Ein höherwertiger Typ von PARI ist der Typ "Vektor". Ein Vektor besteht ebenfalls aus einer Folge von Speicherzellen. Dabei enthält die erste Zelle wiederum organisatorische Daten. Alle anderen Zellen sind jedoch Pointer auf andere PARI-Objekte, die Einträge des Vektors. An diesem Beispiel wird der rekursive Aufbau von PARI gut deutlich. Mit Hilfe von Vektoren werden beispielsweise modulare Zahlen dargestellt. Dabei enthält die zweite Zelle einen Verweis auf den Modul und die dritte Zelle einen Verweis auf den kleinsten positiven Rest der darzustellenden Zahl. Eine Übersicht über alle anderen Datentypen findet man in dem PARI-Handbuch [BBCO91].

Besonders wichtig ist für uns noch die Darstellung von elliptischen Kurven. Eine elliptische Kurve wird in PARI als Vektor der Länge fünf, also als $e = [e_1, e_2, e_3, e_4, e_5]$, dargestellt. Diese Darstellung entspricht der allgemeinen Weiterstraß-Form einer elliptischen Kurve. Da wir nur die kurze Weierstraß-Normalform verwenden, benutzen wir nur die letzten zwei Komponenten des Vektors, d.h wir stellen eine elliptische Kurve $E : y^2 = x^3 + ax + b$ dar als $e = [0, 0, 0, a, b]$. Dabei entsprechen a und b natürlich Verweisen auf lange Zahlen des Werts a bzw. b.

Punkte der elliptischen Kurve werden in PARI als Vektoren der Länge zwei dargestellt. Dabei enthält die erste Komponente des Vektors einen Verweis auf die x-Koordinate des Punktes und die zweite Komponente einen Verweis auf die y-Koordinate. Eine Ausnahme macht dabei nur der Punkt $\mathcal{O}$; er wird dargestellt als Vektor [0]. Da wir über einem Primkörper arbeiten, zeigen die Verweise in diesen Vektoren auf GEN-Objekte vom Typ "modulare Zahl". Diese werden dann wiederum als Vektoren dargestellt. Dieses Beispiel macht wiederum die ausgeprägte Rekursivität von PARI deutlich.

Für alle PARI-Typen stellt PARI eine Menge von Funktionen zur Verfügung. Diese Funktionen sorgen dafür, daß die Typbedingungen nicht verletzt werden. Unter den zur Verfügung gestellten Funktionen befinden sich die am meisten benötigten elementaren Operationen, aber auch Funktionen wie Berechnung des Legendre-Symbols und andere mathematische Operationen existieren. Ein Überblick über diese Funktionen und ihre Eingabeparameter entnehme man dem Handbuch [BBCO91].

Eine "Spezialität" von PARI ist es, keine automatische Speicherbereinigung auf dem

Stack durchzuführen. Dies wird als Aufgabe des Benutzers angesehen. Ohne vom Benutzer programmierte Speicherbereinigung führt dies dazu, daß PARI sehr schnell mit dem Fehler "stack overflow" abbricht. Eine weitere Fehlerquelle ergibt sich, wenn der Benutzer auf Objekte zugreift, die sich zwar im Stack, aber oberhalb des Stack pointers befinden. Dies kann aber bei unsachgemäßer Speicherbereinigung leicht passieren. Dies zeigt schon, daß das Problem der Beseitigung alter, nicht mehr benötigter Ergebnisse eine der größten Fehlerquellen bei der Benutzung von PARI aufwirft. Durchführen kann der Benutzer die Speicherbereinigung entweder durch Hochsetzen des globalen Stack pointers "avma" (was der schnellste und einfachste Weg ist, aber oft nicht möglich ist) oder mit Hilfe einer speziellen Funktion "gerepile()". Details zur Benutzung dieser ziemlich komplizierten Funktion findet man wiederum in [BBCO91].

4.2 Beschreibung unserer praktischen Implementierung

In diesem Abschnitt werden wir nun unsere praktische Implementierung der einzelnen Schritte des Shanks Algorithmus 2.1 beschreiben.

4.2.1 Berechnung von Vielfachen eines Punktes

Wir haben im vorherigen Kapitel schon einen Algorithmus zur Berechnung von Vielfachen von Punkten angegeben. Dieser Algorithmus 3.2 beruhte auf einer Verallgemeinerung der schnellen Exponentiation auf den Fall der additiven Gruppe $E(\mathbb{F}_p)$. In unserer praktischen Implementierung verwenden wir eine Erweiterung dieses Algorithmus nach François Morain ([MoOl90]). Die Idee seines Algorithmus basiert auf der Beobachtung, daß man lange Ketten von Einsen in der Binärzerlegung des Multiplikators besser behandeln kann als im Falle von Algorithmus 3.2. Als Beispiel betrachte man den Fall

$$15 \cdot P = 2 \cdot (2 \cdot (2 \cdot P + P) + P) + P = (2 \cdot (2 \cdot (2 \cdot (2 \cdot P)))) - P.$$

In diesem Beispiel benötigt man im ersten Fall 6 Punktadditionen, im zweiten Fall jedoch nur 5 Punktadditionen (da die Subtraktion eines Punktes einer Punktaddition entspricht). An diesem praktischen Beispiel erkennt man schon die Vorgehensweise: man ersetzt einen binären Block '1^a' im Multiplikator durch den Block '$10^{a-1}(-1)$' (wobei (-1) für eine Subtraktion steht). Führt man diese Idee weiter aus, so erkennt man, daß man auch binäre Blöcke der Form '1^a01^b' besser behandeln kann. Diese kann man durch den äquivalenten Block '$10^a(-1)0^{b-1}(-1)$' ersetzen. Die Sonderbehandlung dieser Fälle führt zu einer Beschleunigung gegenüber Algorithmus 3.2. Dieser neue Algorithmus wird beschrieben durch den endlichen Automaten aus Abbildung 4.1.

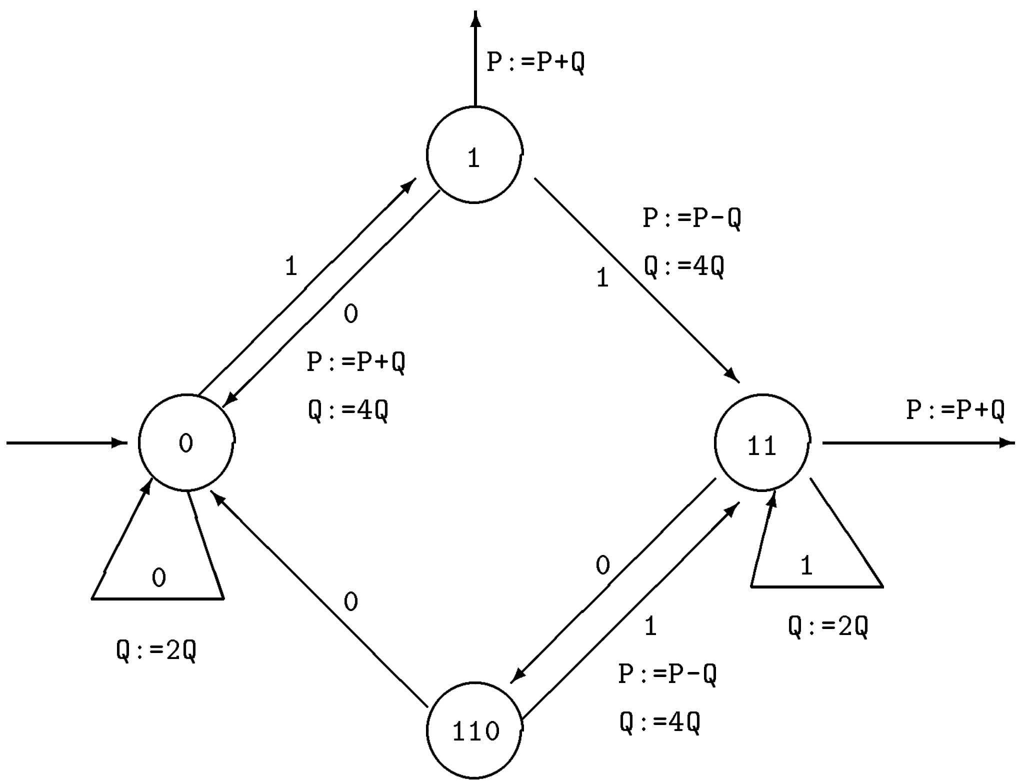

Abbildung 4.1: Morains Algorithmus zur Berechnung von $P = x \cdot Q$

Wie Morain in [MoOl90] ausführt, bringt dieser neue Algorithmus eine relative Laufzeitverbesserung von ungefähr 11 Prozent, wenn wir die Anzahl aller Operationen betrachten. Obwohl der Algorithmus größenordnungsmäßig nicht besser ist als Algorithmus 3.2, stellt er in der Praxis damit doch eine Verbesserung dar. Eine ausführliche Beschreibung der Ideen und eine Analyse findet man in [MoOl90].

4.2.2 Die Punktsuche

Für die Punktsuche wurde genau der im vorherigen Kapitel vorgestellte Punktsuche-Algorithmus 3.3 implementiert. Da wir auf einem Rechner keine "echten" Zufallszahlen erzeugen können, benutzen wir stattdessen die Pseudozufallszahlen eines Pseudozufallszahlengenerators. Da PARI leider keine Funktion besitzt, die lange Pseudozufallszahlen liefert, haben wir uns dabei darauf beschränkt, Pseudozufallszahlen bis 2^{32} zu benutzen. Diese Zufallszahlen werden von dem in der Programmiersprache C zur Verfügung gestellten Pseudozufallszahlengenerator (gegeben durch die Funktion "random()") erzeugt. Dies stellt natürlich eine deutliche Einschränkung für die Menge der tatsächlich von Algorithmus 3.3 erzeugten Punkte dar, was aber in der Praxis keine Probleme bereitet. Die so erzeugten Punkte sind immer noch zufällig "genug", damit der Shanks Algorithmus ein Ergebnis liefert. Man findet die Funktion, die die Punktsuche durchführt, unter dem Namen "find_point()" im Programmlisting.

Zur Berechnung des Legendre-Symbols benutzen wir dabei die von PARI zur Verfügung gestellte Funktion "kronecker()". Diese Funktion berechnet bei Eingabe von zwei Zahlen das Jacobi-Symbol zu diesen Zahlen. Das Jacobi-Symbol ist eine Verallgemeinerung des Legendre-Symbols und stimmt für Primzahlen mit dem Legendre-Symbol überein.

Zur Berechnung einer Quadratwurzel modulo der Primzahl p wurde der schon im vorherigen Kapitel beschriebene RESSOL-Algorithmus von D. Shanks implementiert. Zwar bietet auch PARI eine entsprechende Funktion an, doch diese Funktion löst das Problem der Berechnung einer Quadratwurzel durch sukzessives Ausprobieren. Dies ist aber für große Primzahlmoduln, wie wir sie verwenden, ein im allgemeinen extrem schlechter Algorithmus. Um den RESSOL-Algorithmus möglichst schnell zu machen, haben wir – wenn möglich – Multiplikationen und Divisionen durch Shifts ersetzt. Im Programmlisting findet man diese Funktion unter "ressol()".

4.2.3 Die Babysteps

Alle Babysteps des Shanks Algorithmus 2.1, also die Schritte 2 und 10.2, können mit einer Funktion "babysteps()" durchgeführt werden. Diese Funktion erhält als Eingabe zwei Punkte P und Q und den maximalen Suchbereich. Dieser wird gegeben durch eine untere Schranke x und die Größe des Bereichs y. Damit soll für alle $x' \in \{x + i \ ; \ 0 \leq i \leq y\}$ überprüft werden, ob $x' \cdot P = Q$ ist. In dieser Funktion werden die Babysteps einer solchen Überprüfung mit dem Babystep-Giantstep Algorithmus durchgeführt. Am Ende der Funktion "babysteps()" wird automatisch die Funktion "giantsteps()" zur Durchführung der Giantsteps aufgerufen. Weiterhin gibt es noch ein Eingabebit, mit dem entschieden wird, ob nach Durchführung der Baby- und Giantsteps die Tabelle gelöscht werden soll oder nicht. Im zweiten Teil des Shanks Algorithmus, also in Schritt 10, muß man nämlich eventuell mehrere Babystep-Giantstep Durchläufe mit dem gleichen Punkt P machen. Dabei sollte man die im vorherigen Durchlauf berechnete Tabelle nicht löschen, sondern wiederverwenden. Existiert eine gültige Tabelle, dann können die Babysteps einfach durch Suchen in dieser Tabelle durchgeführt werden.

Aber auch bei Durchführung der Babysteps kann man noch einige Tricks anwenden. Eine erste Beobachtung ist die, daß wir bei der Berechnung von $i \cdot P$ den im vorherigen Schleifendurchlauf berechneten Punkt $(i-1) \cdot P$ benutzen können. Deshalb können wir in jedem Babystep den aktuellen Punkt $i \cdot P$ durch genau eine Punktaddition berechnen.

Die Tabelle der Punkte legen wir auf dem PARI-Stack ab. Leider stellt PARI keine Möglichkeit zur Verfügung, auf Tabellen auf dem Stack zuzugreifen. Daher haben wir eine eigene Addressierung entworfen. In einer globalen Variablen "*top_tab*" merken wir uns die Addresse des ersten Tabelleneintrags. Weiterhin sorgen wir dafür, daß alle Tabelleneinträge die gleiche Anzahl von Speicherzellen benötigen ; wir allokieren nämlich zuerst ausreichend Platz (fester Größe) und kopieren dann den Tabelleneintrag dorthin. Da wir vor dem Abspeichern eines neuen Tabellenelements weiterhin dafür sorgen, daß der PARI-Stack oberhalb des letzten aktuellen Tabelleneintrags geleert wird, werden alle Tabelleneinträge ohne "Löcher" nacheinander auf den PARI-Stack geschrieben. Dann erhalten wir die Addresse des i-ten Tabelleneintrags als

$$top\_tab + i \cdot ZL \cdot 4.$$

Dabei ist ZL die Anzahl der 32-Bit Worte, die zur Abspeicherung eines Tabellenelements verwendet wurde.

Eine weitere Verbesserung erhalten wir, wenn wir in der Tabelle nicht ganze Punkte, sondern nur die x-Koordinaten dieser Punkte abspeichern. Dann speichern wir nämlich implizit mit dem Punkt $i \cdot P$ auch den Punkt $-i \cdot P$ ab. Man sollte nämlich beachten, daß diese Punkte die gleiche x-Koordinate besitzen. Außerdem benötigen wir für einen Tabelleneintrag nur noch die Hälfte des Platzes, den wir bei Abspeicherung von ganzen Punkten benötigt hätten. Damit braucht ein Tabellenelement nur noch $k+1$ Speicherstellen, wobei k die Anzahl der Speicherzellen ist, die man zur Abspeicherung der Charakteristik des Körpers benötigt. In der einen weiteren Speicherzelle merken wir uns den Index i jedes Eintrags $i \cdot P$.

Speichern wir wie gerade beschrieben nur die x-Koordinaten von Punkten in unserer Tabelle, so vermindert sich auch die Anzahl der Baby- und Giantsteps, denn wir speichern ja implizit statt einem zwei Punkte ab. Da wir schon früher bemerkt hatten, daß das Verfahren optimal ist, wenn die Anzahl der Babysteps und die Anzahl der Giantsteps gleich sind, kann man die neue optimale Anzahl k von Schleifendurchläufen berechnen als Lösung von

$$x = \frac{4\sqrt{p}}{2x}.$$

Sei nämlich x die Anzahl der Babysteps. Dann besitzt die Tabelle die implizite Größe $2 \cdot x$ und die Anzahl der noch benötigten Giantsteps ergibt sich als $\frac{4\sqrt{p}}{2x}$. Diese Gleichung besitzt die Lösung $\sqrt{2} \cdot \sqrt[4]{p}$. Also können wir in diesem Fall die Anzahl der Babysteps als $k = \lceil \sqrt{2} \cdot \sqrt[4]{p} \rceil$ wählen. Da die Schrittweite, mit der wir mit dem Punkt Q durch das Lösungsintervall gehen, abhängig ist von der Tabellengröße, sollte man den Punkt Q (vgl. Schritt 3 des Shanks Algorithmus) nun wählen als $Q = (2k) \cdot P$.

Beim Abspeichern in der Tabelle werden die x-Koordinaten der Punkte $i \cdot P$ unsortiert in der Tabelle abgelegt. Damit ist es aber nicht möglich, bei der Durchführung der Giantsteps effizient nach diesen Einträgen zu suchen. Deshalb sortieren wir die Tabelle nach Ablauf aller Babysteps nach der Größe der x-Koordinaten (die wir hier als ganze Zahlen auffassen). Dies geschieht mit Hilfe des Quicksort-Algorithmus. Die Addressierung der Tabellenelemente erfolgt dabei auf die gleiche Art und Weise wie schon beschrieben. Die dabei möglichen Vertauschungen von Tabelleneinträgen können durch Kopieren der einzelnen Speicherzellen geschehen. Eine Beschreibung des Sortieralgorithmus Quicksort findet man in [Me84].

4.2.4 Die Giantsteps

Auch bei den Giantsteps sollte man natürlich die Punkte $j \cdot Q$ nicht immer neu berechnen. Hier kann man den gleichen Trick anwenden wie bei den Babysteps und durch Abspeichern der Zwischenpunkte $(j-1) \cdot Q$ den aktuellen Giantstep-Punkt $j \cdot Q$ durch genau eine Punktaddition berechnen.

Weiterhin stellen wir sicher, daß bei Beginn der Giantsteps die Tabelle mit den x-Koordinaten der Punkte der Babysteps nach Größe sortiert ist. Zusätzlich merken wir uns noch in einer globalen Variablen "tab_size" die Anzahl der Einträge der Tabelle. Benutzen wir wieder die schon erwähnte Addressierung, so können wir mit binärer Suche (vgl. [Me84]) dann schnell in der Tabelle suchen. Diese binäre Suche findet man in der Implementierung in der Funktion "bin_search()".

Das alleinige Abspeichern der x-Koordinaten der Punkte hat nun bei den Giantsteps kleine Auswirkungen. Angenommen, wir haben für festes j bei der binären Suche entdeckt, daß die x-Koordinaten von $j \cdot Q$ und $i \cdot P$ für ein $0 \leq i \leq k$ gleich sind. Dabei beachte man, daß der Wert von i in dem Tabelleneintrag (und zwar in der letzten Speicherstelle des Eintrags für $i \cdot P$) verfügbar ist. Dann gilt im Gegensatz zur Beschreibung des Shanks Algorithmus 2.1 die Gleichheit

$$(\lfloor p + 1 - 2\sqrt{p} \rfloor + j \cdot 2 \cdot k) \cdot P = \pm i \cdot P.$$

Somit müssen wir beiden Seiten dieser Gleichung berechnen und dann entscheiden, welcher Fall vorliegt.

Im Gegensatz zur Beschreibung des Shanks Algorithmus 2.1 beenden wir in unserer praktischen Implementierung die Durchführung der Giantsteps, wenn wir eine Lösung gefunden haben. Diese Lösung muß ja ein Vielfaches der Ordnung unseres Punktes P sein. Da für Primzahlen bis etwa 10^{30} die PARI-Funktion "divisors()" recht schnell alle Teiler einer Zahl berechnet (also die Zahl faktorisiert), können wir daraus schnell die Ordnung von P ausrechnen. Wir bestimmen einfach den kleinsten Teiler d, so daß $d \cdot P = \mathcal{O}$ ist. Dies muß dann die Ordnung von P sein.

Danach testen wir in "order_found()", ob genau ein Vielfaches der Punktordnung im Hasse-Intervall liegt. Ist dies der Fall, so haben wir die Gruppenordnung bestimmt, ansonsten beginnt ein neuer Durchlauf.

4.2.5 Das Hauptprogramm

Im Hauptprogramm werden im großen und ganzen nur organisatorische Aufgaben erfüllt. Die wichtigste Aufgabe besteht noch in der Initialisierung des PARI-Stacks mit Hilfe der PARI-Funktion "init()". Dabei muß insbesondere die PARI-Stackgröße für den kompletten Programmdurchlauf festgelegt werden, denn es ist nicht möglich, später noch zusätzlichen Platz zu allokieren oder freizugeben. Wir realisieren dies, indem wir nach Eingabe der Körpercharakteristik nach obiger Formel die Anzahl der Babysteps und damit die Anzahl der Tabelleneinträge berechnen. Um den Rechner nicht zu überlasten, legen wir eine maximale Tabellengröße fest (bei uns zur Zeit 500000 Einträge). Wird diese Grenze überschritten, so muß die Funktion Babysteps die Tabellengröße beschränken. Da die Tabellengröße in der globalen Variablen "tab_size" immer bewahrt wird, kann sich die Funktion "giantsteps()" dann die korrekte Anzahl der Giantsteps ausrechnen (diese ist nicht mehr notwendig optimal). Steht somit die Anzahl der Tabelleneinträge fest, dann können wir den PARI-Stack initialisieren. Da jeder Eintrag eine feste Größe besitzt, läßt sich leicht die benötigte Stack-Größe berechnen. Man sollte jedoch darauf achten, daß der PARI-Stack nicht zu klein wird, weil auch noch Platz für automatisch berechnete Konstanten benötigt wird. Daher allokieren wir immer mindestens 2000000 Bits als Stack.

Zusätzliche Aufgaben des Hauptprogramms bestehen im wesentlichen nur aus Aufrufen der einzelnen Unterfunktionen, die die zufälligen Punkte suchen und dann die Babysteps durchführen. Am Ende des Programms ist das Hauptprogramm auch noch für die Ausgabe der berechneten Ergebnisse zuständig. Weiterhin werden im Hauptprogramm noch einige Variablen initialisiert und einige Konstanten wie die Hasse-Grenzen berechnet.

Für weitere Informationen schaue man sich im Programmlisting die Funktion "main()" an; diese Funktion ist ziemlich selbsterklärend.

Kapitel 5

Der Schoof-Algorithmus

In diesem Kapitel werden wir die Idee von Schoof ([Scho85]) zur Berechnung der Punktanzahl einer elliptischen Kurve über einem Primkörper vorstellen. Durch Rechnungen mit bestimmten Polynomen können wir dabei die Gruppenordnung bestimmen. Der Algorithmus von Schoof ist dabei vollständig deterministisch und benötigt nur polynomiell viel Zeit. Allerdings ist er in der Praxis zu zeitaufwendig, um für Primkörper großer Charakteristik die Punktanzahl einer elliptischen Kurve zu berechnen. Im übernächsten Kapitel werden wir jedoch eine Idee vorstellen, wie wir Teile des Schoof-Algorithmus benutzen können, um den Shanks Algorithmus zu beschleunigen.

Im folgenden ersten Abschnitt werden wir die theoretischen Grundlagen des Schoof-Algorithmus vorstellen. Dabei werden wir wie im Einführungskapitel größtenteils auf die Beweise der grundlegenden Sätze verzichten und dazu auf die Literatur verweisen. Sei im folgenden $E : y^2 = x^3 + ax + b$ wieder eine feste elliptische Kurve über dem Primkörper $\mathbb{F}_p$, von der wir die Ordnung der Punktgruppe $E(\mathbb{F}_p)$ bestimmen wollen.

5.1 Theorie des Schoof-Algorithmus

Der Schoof-Algorithmus basiert auf den Eigenschaften von speziellen Polynome, den sogenannten Divisionspolynomen. Diese Polynome werden wir nun als erstes vorstellen.

Definition 5.1 Für $n \geq -1$ ist das **n-te Divisionspolynom** $\psi_n \in \mathbb{F}_p[x, y]$ definiert als

- $\psi_{-1}(x, y) = -1$
- $\psi_0(x, y) = 0$

- $\psi_1(x,y) = 1$
- $\psi_2(x,y) = 2y$
- $\psi_3(x,y) = 3x^4 + 6ax^2 + 12bx - a^2$
- $\psi_4(x,y) = 4y(x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3)$
- $\psi_{2n}(x,y) = \frac{\psi_n(x,y)}{2y} \cdot \left(\psi_{n+2}(x,y)\psi_{n-1}^2(x,y) - \psi_{n-2}(x,y)\psi_{n+1}^2(x,y)\right)$ für $n \geq 3$.
- $\psi_{2n+1}(x,y) = \psi_{n+2}(x,y)\psi_n^3(x,y) - \psi_{n+1}^3(x,y)\psi_{n-1}(x,y)$ für $n \geq 2$.

Dabei sind a und b die Konstanten aus der Kurvengleichung $E : y^2 = x^3 + ax + b$.

Durch diese rekursive Definition besitzen die Divisionspolynome eine besondere Form. Wir erhalten nämlich

Lemma 5.1 *Für das n-te Divisionspolynom gilt, daß*

$$\psi_n(x,y) \in \begin{cases} y \cdot \mathbb{F}_p[x,y^2] & \textit{falls } n \textit{ gerade ist.} \\ \mathbb{F}_p[x,y^2] & \textit{falls } n \textit{ ungerade ist.} \end{cases}$$

Beweis: Durch Induktion über n und Benutzung der rekursiven Definition der Divisionspolynome zeigt man leicht die Behauptung des Lemmas. Dabei muß man in der Induktion die Fälle $n = 2 \cdot m$ oder $n = 2 \cdot m + 1$ (d.h. gerade oder ungerade) und zusätzlich m gerade oder ungerade unterscheiden. ■

Die Divisionspolynome sind nach Definition Polynome in den zwei Variablen x und y. Wir wollen jedoch möglichst oft vermeiden, mit Polynomen in zwei Variablen zu rechnen. Um möglichst nur mit einer Variablen x zu rechnen, beachte man, daß für die Variable y die Kurvengleichung $y^2 = x^3 + ax + b$ erfüllt ist. Demnach können wir alle y^2-Terme eines Divisionspolynoms durch $x^3 + ax + b$ ersetzen. Dann erhalten wir

Definition 5.2 Die **reduzierten Divisionspolynome** $f_n(x) \in \mathbb{F}_p[x]$ sind definiert als

$$f_n(x) = \begin{cases} \frac{1}{y}\left(\psi_n(x,y) \big/ (y^2 - x^3 - ax - b)\right) & \text{falls } n \text{ gerade.} \\ \psi_n(x,y) \big/ (y^2 - x^3 - ax - b) & \text{falls } n \text{ ungerade.} \end{cases}$$

Bemerkung 5.1 *Nach Lemma 5.1 ist es offensichtlich, daß f_n wirklich ein Polynom in nur noch einer Veränderlichen x ist.*

Um später die Komplexität des Schoof-Algorithmus herleiten zu können, müssen wir Informationen über den Grad der reduzierten Divisionspolynome besitzen. Dies liefert

Lemma 5.2 *Falls p kein Teiler von n ist, so gilt für den Grad des reduzierten Divisionspolynoms $f_n(x)$*

$$grad\,(f_n(x))) = \begin{cases} \frac{1}{2}(n^2-1) & \text{falls } n \text{ ungerade ist.} \\ \frac{1}{2}(n^2-4) & \text{falls } n \text{ gerade ist.} \end{cases}$$

Beweis: Den Beweis erhält man wiederum durch Induktion über n und Unterscheidung der einzelnen Fälle in der rekursiven Definition von f_n bzw. ψ_n. ■

Nun wollen wir uns mit den besonderen Eigenschaften der Divisionspolynome beschäftigen, die den Gebrauch dieser Polynome zur Berechnung der Ordnung der Punktgruppe $E(\mathbb{F}_p)$ ermöglichen. Das n-te Divisionspolynom kann nämlich dazu benutzt werden, Aussagen über das Vielfache $n \cdot P$ für einen beliebigen Punkt $P \in E(\overline{\mathbb{F}_p})$ zu machen. Nähere Informationen dazu liefern die beiden folgenden Sätze.

Satz 5.1 *Sei $P = (x', y') \in E(\overline{\mathbb{F}_p})$ ein Punkt der elliptischen Kurve E mit $2 \cdot P \neq \mathcal{O}$ und sei $n \geq -1$. Dann ist genau dann $n \cdot P = \mathcal{O}$, wenn $f_n(x') = 0$ ist.*

Beweis: [La78] ■

Satz 5.2 *Sei $n \geq 1$ und $P = (x', y') \in E(\overline{\mathbb{F}_p})$ ein Punkt der elliptischen Kurve E mit $n \cdot P \neq \mathcal{O}$. Dann gilt*

$$n \cdot P = \left(x' - \frac{\psi_{n-1}\psi_{n+1}}{\psi_n^2}, \frac{\psi_{n+2}\psi_{n-1}^2 - \psi_{n-2}\psi_{n+1}^2}{4y'\psi_n^3} \right).$$

Dabei fasse man ψ_k immer als $\psi_k(x', y')$ auf.

Beweis: [La78] ■

Diese Eigenschaften der Divisionspolynome werden wir im Schoof-Algorithmus ausnutzen. Um die Verbindung zum Problem der Bestimmung der Ordnung der Punktgruppe $E(\mathbb{F}_p)$ herzustellen, beschäftigen wir uns mit folgender Abbildung.

Lemma 5.3 *Sei E eine elliptische Kurve über dem Primkörper $\mathbb{F}_p$. Dann ist*

$$\begin{aligned} \phi : E(\overline{\mathbb{F}_p}) &\longrightarrow E(\overline{\mathbb{F}_p}) \\ (x, y) &\longmapsto (x^p, y^p) \end{aligned} \tag{5.1}$$

ein Endomorphismus der Gruppe $E(\overline{\mathbb{F}_p})$.

Beweis: Zum Beweis der Abgeschlossenheit der Abbildung ϕ beachte man, daß in einem Körper der Charakteristik p die Gleichungen $(x+y)^p = x^p + y^p$ und $a^p = a$ für jedes Element $a \in \mathbb{F}_p$ gelten (siehe [Hun74, Seite 280]). Damit erhalten wir

$$\begin{aligned} (y^p)^2 &= (y^2)^p \\ &= (x^3 + ax + b)^p \\ &= (x^3)^p + a^p x^p + b^p \\ &= (x^p)^3 + ax^p + b \end{aligned}$$

und mit (x, y) ist auch (x^p, y^p) ein Punkt aus $E(\overline{\mathbb{F}_p})$.

Die Homomorphieeigenschaft der Abbildung ϕ kann man mit Hilfe der Additionsformeln aus Satz 1.1 auf die gleiche Art beweisen. ∎

Definition 5.3 Der Endomorphismus ϕ aus (5.1) heißt der **Frobeniusendomorphismus**.

Der Frobeniusendomorphismus stellt nun die fehlende Beziehung zu unserem Problem der Berechnung der Gruppenordnung $\#E(\mathbb{F}_p)$ her. Um dies zu erkennen, müssen wir noch eine Eigenschaft des Frobeniusendomorphismus angeben.

Satz 5.3 *Der Frobeniusendomorphismus ϕ erfüllt im Endomorphismenring $End(E(\overline{\mathbb{F}_p}))$ die Relation*

$$\phi^2 - t'\phi + p = 0$$

mit $|t'| \leq 2\sqrt{p}$, d.h. für alle Punkte $P \in E(\overline{\mathbb{F}_p})$ gilt

$$(\phi^2 - t'\phi + p)(P) = \mathcal{O}. \tag{5.2}$$

Dabei sei $i : E(\overline{\mathbb{F}_p}) \rightarrow E(\overline{\mathbb{F}_p})$ die Abbildung, die jedem Punkt $P \in E(\overline{\mathbb{F}_p})$ den Punkt $i \cdot P$ zuordnet.

Beweis: [ChRo88, Theorem 12.16, Seite 71] ∎

Bemerkung 5.2 *Man kann leicht zeigen, daß die Abbildung i aus obigem Satz 5.3 ein Endomorphismus von $E(\overline{\mathbb{F}_p})$ ist. Diese Abbildung läßt sich auch als ganze Zahl auffassen. Im folgenden werden wir häufig von der Zahl i sprechen, wenn wir die Abbildung $i : E(\overline{\mathbb{F}_p}) \rightarrow E(\overline{\mathbb{F}_p})$ meinen.*

Nach dem Satz von Hasse 1.2 wissen wir, daß $\#E(\mathbb{F}_p) = 1 + p - t$ für eine natürliche Zahl t mit $|t| \leq 2\sqrt{p}$ ist. Man kann nun vermuten, daß diese Zahl t und die Zahl t' aus Satz 5.3 in einer Beziehung zueinander stehen. Dies wird erhärtet durch das folgende Lemma.

Lemma 5.4 *Die Zahl t' aus Satz 5.3 und die Zahl t aus der Hasse-Gleichung $\#E(\mathbb{F}_p) = p+1-t$ sind gleich.*

Beweis: Zum Beweis beachte man, daß der Frobeniusendomorphismus eingeschränkt auf die $\mathbb{F}_p$-rationalen Punkte der Punktgruppe $E(\overline{\mathbb{F}_p})$ die Identität ist. Da die Gruppenordnung $\#E(\mathbb{F}_p)$ eindeutig bestimmt ist, folgt damit schon die Behauptung des Lemmas. ∎

Dieser Satz legt nun schon die Vorgehensweise des Schoof-Algorithmus nahe. Wir können die Ordnung der Punktgruppe $E(\mathbb{F}_p)$ berechnen, indem wir die sogenannte **Spur** t' des Frobeniusendomorphismus berechnen. Dazu müssen wir für alle Punkte aus $E(\overline{\mathbb{F}_p})$ testen, mit welchem Wert für t' die Gleichung (5.2) erfüllt ist. Leider ist die Anzahl der möglichen Werte für t' von der Größenordnung $O(\sqrt{p})$ und wir haben keine Hinweise über den "wahrscheinlichen" Wert von t'. Deshalb versuchen wir, den Wert von t' nur modulo kleiner Primzahlen zu bestimmen. Zur Bestimmung von $t' \bmod n$ für eine ungerade Primzahl n betrachten wir nicht mehr alle Punkte aus $E(\overline{\mathbb{F}_p})$. Wir können uns stattdessen auf Punkte aus der folgenden Untergruppe beschränken.

Definition 5.4 Die Menge aller Punkte aus $E(\overline{\mathbb{F}_p})$ mit Ordnung n heißt die **n-Torsionsgruppe** $E[n]$.

Bemerkung 5.3 *Man kann leicht beweisen, daß die Menge der n-Torsionspunkte wirklich eine Untergruppe von $E(\overline{\mathbb{F}_p})$ ist.*

Eine wichtige Eigenschaft der n-Torsionsgruppe wird gegeben in

Satz 5.4 *Falls p nicht n teilt, so ist $E[n] \cong \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$, d.h. insbesondere ist $E[n]$ nicht die triviale Untergruppe $\{\mathcal{O}\}$.*

Beweis: [Si86, Cor. 6.4, Seite 89] ∎

Wir wollen nun die Gleichung (5.2) nur noch für Punkte aus der n-Torsionsgruppe überprüfen. Nach obigem Satz 5.4 wissen wir, daß diese Gruppe für im Vergleich zu p kleine Primzahlen n nicht trivial und endlich ist. Wir werden im folgenden Satz beweisen, daß wir damit wirklich das gewünschte Ergebnis $t' \bmod n$ berechnen können.

Satz 5.5 *Für alle n-Torsionspunkte $P \in E[n]$ erfüllt der Frobeniusendomorphismus ϕ die Gleichung*

$$(\phi^2 - t_n\phi + p)(P) = \mathcal{O} \tag{5.3}$$

mit $t_n \equiv t' \bmod n$.

Beweis: Sei ϕ_n die Einschränkung von ϕ auf $E[n]$. Dann ist ϕ_n abgeschlossen, denn wegen der Homomorphieeigenschaft von ϕ (und somit auch von ϕ_n) ist für alle Punkte $P \in E[n]$ auch $\phi_n(P) \in E[n]$:

$$\mathcal{O} = \phi_n(\mathcal{O}) = \phi_n(n \cdot P) = n \cdot \phi_n(P).$$

Da ϕ_n nur eine Einschränkung des Frobeniusendomorphismus ist, gilt wegen (5.2) insbesondere für alle n-Torsionspunkte P

$$(\phi_n^2 - t'\phi_n + p)(P) = \mathcal{O}.$$

Gilt nun andererseits auch $(\phi_n^2 - t_n\phi_n + p)(P) = \mathcal{O}$ für alle $P \in E[n]$, so folgt damit

$$((t_n - t')\phi_n)(P) = \mathcal{O}.$$

Da nach Definition von ϕ_n und Satz 5.4 ein Punkt $P \in E[n]$ existiert, so daß $Q = \phi_n(P) \neq \mathcal{O}$ ist, gilt für diesen Punkt

$$(t_n - t')(Q) = \mathcal{O}.$$

Da Q ebenfalls ein n-Torsionspunkt ist, muß folglich $t_n \equiv t \bmod n$ sein. ■

Dieser Satz legt nun ein Verfahren zur Berechnung der Spur t des Frobeniusendomorphismus nahe. Wir berechnen $t \bmod n$, indem wir testen, für welchen kleinsten positiven Rest t_n modulo n die Relation (5.3) für alle Punkte von $E[n]$ gilt. Die Rechnungen in $E[n]$ können wir dabei mit Hilfe der am Anfang des Kapitels eingeführten Divisionspolynome durchführen. Haben wir $t \bmod n$ für "genügend" viele n berechnet, so können wir mit dem Chinesischen Restsatz (vgl. dazu [Co91, Seite 16]) den Wert von t berechnen. Damit kennen wir dann auch die Gruppenordnung $\#E(\mathbb{F}_p)$.

5.2 Der Algorithmus

5.2.1 Berechnung von $t \bmod n$

Bevor wir den gesamten Schoof-Algorithmus in Formeln angeben, beschreiben wir zuerst die einzelnen Schritte zur Berechnung von $t \bmod n$. Sei dazu im folgenden n eine ungerade feste Primzahl. Die Einschränkung $n \geq 3$ benötigen wir, um die Formeln aus Satz 5.1 und 5.2 benutzen zu können.

Nach Satz 5.5 können wir $t \bmod n$ berechnen, indem wir testen, für welches $t_n \in \mathbb{Z}/n\mathbb{Z}$ die Relation $\phi_n^2 + p = t_n\phi_n$ für alle n-Torsionspunkte gilt. Weil für einen n-Torsionspunkt

$P \in E[n]$ die Gleichheit $p \cdot P = q \cdot P$ mit dem kleinsten positiven Rest $q \equiv p \bmod n$ gilt, können wir die Gleichung (5.3) umformen zu

$$\phi_n^2 + q = t_n \phi_n.$$

Wir müssen nun testen, für welchen Wert von t_n diese Gleichung für alle n-Torsionspunkte erfüllt ist. Dazu zerlegen wir den ganzen Test in zwei Teile:

1. Teste, ob ein n-Torsionspunkt die Gleichung $\phi_n^2 = \pm q$ erfüllt. Dann können wir leicht den Wert von t_n berechnen.
2. Teste für $t_n = 1, \ldots, n-1$, ob $\phi_n^2(P) + q \cdot P = t_n \cdot \phi_n(P)$ für alle n-Torsionspunkte P gilt. Dabei müssen wir alle Werte für t_n sukzessive testen, bis wir den "richtigen" Wert gefunden haben.

Wir werden nun die beiden Teile einzeln genauer untersuchen. Dabei werden wir auch die Formeln herleiten, um diese Tests durchzuführen. Zuerst nun also zum ersten Teil.

Erster Teil der Berechnung von $t \bmod n$

Im ersten Teil wollen wir testen, ob ein n-Torsionspunkt $P \in E[n]$ existiert, der die Gleichung

$$\phi_n^2(P) = \pm q \cdot P \tag{5.4}$$

erfüllt (dabei sei q wiederum der kleinste positive Rest von $p \bmod n$). Setzen wir dazu einen beliebigen n-Torsionspunkt $P \neq \mathcal{O}$ allgemein an als $P = (x, y)$. Dann lautet die linke Seite von (5.4)

$$\phi_n^2(P) = \phi_n((x^p, y^p)) = (x^{p^2}, y^{p^2}).$$

Weiterhin können wir mit Hilfe von Satz 5.2 die rechte Seite von (5.4) berechnen als

$$\pm q \cdot P = \left(x - \frac{\psi_{q-1}(x,y)\psi_{q+1}(x,y)}{\psi_q^2(x,y)}, \pm \frac{\psi_{q+2}(x,y)\psi_{q-1}^2(x,y) - \psi_{q-2}(x,y)\psi_{q+1}^2(x,y)}{4y\psi_q^3(x,y))} \right).$$

Dabei sollte man beachten, daß die x-Koordinaten der Punkte $q \cdot P$ und $-q \cdot P$ gleich sind. Folglich erhalten wir für einen n-Torsionspunkt $\mathcal{O} \neq P = (x, y)$, der (5.4) erfüllt, durch Vergleich der x-Koordinaten die Gleichung

$$x^{p^2} - x + \frac{\psi_{q-1}(x,y)\psi_{q+1}(x,y)}{\psi_q^2(x,y)} = 0. \tag{5.5}$$

Formen wir (5.5) um, so erhalten wir daraus

$$(x^{p^2} - x) \cdot \psi_q^2(x, y) + \psi_{q-1}(x, y)\psi_{q+1}(x, y) = 0. \tag{5.6}$$

Mit derselben Begründung wie im ersten Abschnitt können wir jetzt in (5.6) alle y^2-Terme durch den rechten Teil der Kurvengleichung (also $x^3 + ax + b$) ersetzen. Dann ist das entstehende Polynom ein Polynom nur noch einer Variablen x. Dies sieht man leicht ein, wenn man die Fälle q gerade oder ungerade unterscheidet und Lemma 5.1 benutzt. Nennen wir das dann entstehende Polynom $g(x)$, d.h.

$$g(x) \quad := \quad (x^{p^2} - x) \cdot \psi_q^2(x, y) + \psi_{q-1}(x, y)\psi_{q+1}(x, y)\Big/(y^2 - x^3 - ax - b).$$

Wie können wir aber prüfen, ob wirklich ein n-Torsionspunkt $P = (x, y) \neq \mathcal{O}$ existiert, dessen x-Koordinate die Gleichung $g(x) = 0$ erfüllt? Die Antwort liefert Satz 5.1. Damit wissen wir, daß jede x-Koordinate eines n-Torsionspunktes eine Nullstelle des Polynoms $f_n(x)$ ist. Also können wir durch Berechnung des größten gemeinsamen Teilers von $g(x)$ und $f_n(x)$ überprüfen, ob ein n-Torsionspunkt existiert, der (5.4) erfüllt. Besitzt dieser größte gemeinsame Teiler einen Grad größer als Null, so existiert mindestens ein n-Torsionspunkt $P \neq \mathcal{O}$, der (5.4) erfüllt. Ansonsten sind wir im ersten Fall erfolglos geblieben und und wir beginnen mit dem zweiten Teil der Berechnung von $t \bmod n$.

Sei nun bekannt, daß ein n-Torsionspunkt P existiert, der die Gleichung $\phi_n^2(P) = \pm q \cdot P$ erfüllt. Wie entscheiden wir nun, welcher der zwei möglichen Fälle $\phi_n^2 = +q$ und $\phi_n^2 = -q$ vorliegt und wie bestimmen wir in beiden Fällen den Wert von t_n?

Nehmen wir dazu einmal an, es existierte ein n-Torsionspunkt $P \neq \mathcal{O}$ mit $\phi_n^2(P) = q \cdot P$. Durch Einsetzen in Gleichung (5.3) erhalten wir dann

$$(2q - t_n\phi_n)(P) = \mathcal{O} \qquad \text{und damit} \qquad \phi_n(P) = \frac{2q}{t_n} \cdot P.$$

Man beachte dabei, daß in diesem Fall $t_n \not\equiv 0 \bmod n$ ist (zum Beweis setze man einfach $\phi_n^2(P) = q \cdot P$ in die Frobeniusgleichung ein, was bei $t_n \equiv 0 \bmod n$ sofort zu einem Widerspruch führt) und daß man daher t_n modulo der Primzahl n invertieren kann. Quadrieren wir die letzte Gleichung, so erhalten wir

$$\phi_n^2(P) = q \cdot P = \frac{4q^2}{t_n^2} \cdot P$$

und damit

$$t_n^2 \equiv 4q \bmod n.$$

Da 4 ein Quadrat modulo n ist, muß also auch q ein Quadrat modulo n sein. Erhalten wir also bei der Berechnung des Legendre-Symbol $\left(\frac{q}{n}\right)$ den Wert -1, so können wir folgern, daß nur die Gleichung $\phi_n^2(P) = -q \cdot P$ die korrekte Gleichung sein kann. Falls wir herausgefunden haben, daß $\phi_n^2(P) = -q \cdot P$ gilt, so erhalten wir durch einfaches Umformen der Frobeniusgleichung (5.3) direkt, daß $t_n \equiv 0 \bmod n$ sein muß.

Ansonsten existiert – da $\left(\frac{q}{n}\right)$ sicher ungleich 0 ist – eine Quadratwurzel $0 < w < n$ mit $w^2 \equiv q \bmod n$. Durch Einsetzen erhalten wir dann $\phi_n^2(P) = w^2 \cdot P$ und damit die Gleichung

$$\phi_n(P) = \pm w \cdot P. \tag{5.7}$$

Falls also kein n-Torsionspunkt $P = (x, y) \neq \mathcal{O}$ existiert, der Gleichung (5.7) erfüllt, so wissen wir wiederum, daß $\phi_n^2 = -q$ und damit $t_n \equiv 0 \bmod n$ ist. Die Existenz eines Punktes, der Gleichung (5.7) erfüllt, können wir auf die gleiche Art und Weise wie oben überprüfen. Wiederum benutzen wir Satz 5.2 und berechnen damit für einen allgemein angesetzten n-Torsionspunkt $\mathcal{O} \neq P = (x, y)$ die linke und rechte Seite von (5.7). Nach Reduktion erhalten wir dann

$$h(x) \quad := \quad (x^p - x) \cdot \psi_w^2(x, y) + \psi_{w-1}(x, y)\psi_{w+1}(x, y) \big/ (y^2 - x^3 - ax - b).$$

Auch $h(x)$ ist ein Polynom in nur noch einer Veränderlichen x. Analog zu obigem Test berechnen wir den größten gemeinsamen Teiler von $h(x)$ und $f_n(x)$. Besitzt dieser größte gemeinsame Teiler den Grad Null, so wissen wir, daß kein n-Torsionspunkt ungleich $\mathcal{O}$ existieren kann, der (5.7) erfüllt. Dann muß sicherlich die Gleichung $\phi_n^2(P) = -q \cdot P$ für einen n-Torsionspunkt gelten und damit ist $t_n \equiv 0 \bmod n$.

Angenommen, wir wissen jetzt, daß $\phi_n(P) = \pm w \cdot P$ für einen n-Torsionspunkt $P = (x, y) \neq \mathcal{O}$ ist. Nun müssen wir noch unterscheiden, ob $\phi_n = w$ oder $\phi_n = -w$ ist. Im Falle $\phi_n = w$ erhalten wir aus der Gleichung $\phi_n = \frac{2q}{t_n} = \frac{2w^2}{t_n}$, daß $t_n \equiv 2w \bmod n$ sein muß. Analog erhalten wir für $\phi_n = -w$ die Lösung $t_n \equiv -2w \bmod n$.

Zur Unterscheidung der beiden Möglichkeiten $\phi_n = w$ und $\phi_n = -w$ verwenden wir noch einmal die gleiche Vorgehensweise. Doch jetzt vergleichen wir nicht die x-Koordinaten, sondern die y-Koordinaten der beiden Seiten. Damit können wir eindeutig feststellen, welcher Fall vorliegt. Um zu überprüfen, ob $\phi_n(P) = w \cdot P$ für einen n-Torsionspunkt $P = (x, y) \neq \mathcal{O}$ gilt, berechnen wir durch Vergleich der y-Koordinaten der Gleichung $\phi_n(P) = w \cdot P$ und mit Hilfe von Satz 5.2

$$k(x, y) = y^p - \left(\frac{\psi_{w+2}(x, y)\psi_{w-1}^2(x, y) - \psi_{w-2}(x, y)\psi_{w+1}^2(x, y)}{4y\psi_w^3(x, y)} \right) = 0.$$

Durch Umformung und Ersetzen der y^2-Terme erhalten wir daraus

$$\begin{aligned} k'(x, y) \quad = \quad & 4 \cdot (x^3 + ax + b)^{(p+1)/2} \psi_w^3(x, y) - \psi_{w+2}(x, y)\psi_{w-1}^2(x, y) + \\ & \psi_{w-2}(x, y)\psi_{w+1}^2(x, y) \big/ (y^2 - x^3 - ax - b). \end{aligned}$$

Durch Fallunterscheidung erkennt man leicht, daß $k'(x, y)$ entweder ein Polynom aus $\mathbb{F}_p[x]$ oder aus $y \cdot \mathbb{F}_p[x]$ ist. Zum Test der Existenz eines n-Torsionspunktes berechnet man analog zu oben den größten gemeinsamen Teiler von $k'(x, y)$ und $f_n(x)$. Besitzt dieses Ergebnis einen Grad größer als Null, so ist $\phi_n = w$, ansonsten ist $\phi_n = -w$. Die entsprechenden Werte für t_n haben wir für beide Fälle schon angegeben.

Zweiter Teil der Berechnung von $t \bmod n$

Nun beschreiben wir den zweiten Teil der Bestimmung von $t' \bmod n$. Wenn wir diesen Teil des Algorithmus erreichen, wissen wir, daß $\phi_n^2(P) \neq \pm q \cdot P$ für alle n-Torsionspunkte $P \in E[n]$ gilt. Nun wollen wir bestimmen, für welchen Wert von $1 \leq t_n < n$ die Gleichung

$$\phi_n^2(P) + q \cdot P = t_n \phi_n(P) \tag{5.8}$$

für alle n-Torsionspunkte P gilt. Sei dazu nun $1 \leq t_n < n$ fest. Wir leiten zuerst die Gleichungen zum Test her, ob (5.8) für alle n-Torsionspunkte für einen festen Wert t_n gilt. Danach können wir dann durch Durchführen dieser Tests für alle möglichen Werte von t_n den Wert von $t \bmod n$ bestimmen.

Wir gehen analog zum ersten Teil der Bestimmung von $t \bmod n$ vor und stellen Polynomgleichungen für die linke und rechte Seite von (5.8) auf. Dann vergleichen wir wiederum die x- und y-Koordinate der beiden Seiten. Dazu benutzen wir wie vorher wieder Satz 5.2. Sei $P = (x, y) \neq \mathcal{O}$ wieder ein beliebiger n-Torsionspunkt. Dann erhalten wir für die linke Seite von (5.8)

$$\phi_n^2(P) + q \cdot P =$$
$$(x^{p^2}, y^{p^2}) + \left(x - \frac{\psi_{q-1}(x,y)\psi_{q+1}(x,y)}{\psi_q^2(x,y)}, \frac{\psi_{q+2}(x,y)\psi_{q-1}^2(x,y) - \psi_{q-2}(x,y)\psi_{q+1}^2(x,y)}{4y\psi_q^3(x,y)}\right).$$

Dabei beachte man, daß das Zeichen '+' für die Punktaddition auf der Kurve E steht. Da wir wissen, daß $\phi_n(P) \neq \pm q \cdot P$ ist, können wir mit Hilfe der zweiten Formel aus Satz 1.1, d.h. der Formel für die Punktaddition von zwei Punkte P und $Q \neq \pm P$, dies berechnen. Dann erhalten wir die Gleichung

$$\phi_n^2(P) + q \cdot P = \tag{5.9}$$
$$\left(-x^{p^2} - x - \frac{\psi_{q-1}\psi_{q+1}}{\psi_q^2} + \lambda^2, -y^{p^2} - \lambda\left(-2x^{p^2} - x + \lambda^2 + \frac{\psi_{q-1}\psi_{q+1}}{\psi_q^2}\right)\right)$$

mit

$$\lambda = \frac{\psi_{q+2}\psi_{q-1}^2 - \psi_{q-2}\psi_{q+1}^2 - 4y^{p^2+1}\psi_q^3}{4\psi_q y((x - x^{p^2})\psi_q^2 - \psi_{q-1}\psi_{q+1}}.$$

Dabei fasse man ψ_k wieder als $\psi_k(x, y)$ auf.

Analog berechnen wir die rechte Seite von (5.8). Dann erhalten wir

$$t_n\phi_n(P) = \phi_n(t_n \cdot P) = \phi_n\left(x - \frac{\psi_{t_n+1}\psi_{t_n-1}}{\psi_{t_n}^2}, \frac{\psi_{t_n+2}\psi_{t_n-1}^2 - \psi_{t_n-2}\psi_{t_n+1}^2}{4y\psi_{t_n}^3}\right).$$

Benutzen wir nun die Definition von ϕ_n und die Tatsache, daß im Körper $\mathbb{F}_p$ die Gleichung $(a'+b')^p = a'^p + b'^p$ gilt, so erhalten wir

$$t_n\phi_n(P) = \left(x^p - \left(\frac{\psi_{t_n+1}\psi_{t_n-1}}{\psi_{t_n}^2}\right)^p, \left(\frac{\psi_{t_n+2}\psi_{t_n-1}^2 - \psi_{t_n-2}\psi_{t_n+1}^2}{4y\psi_{t_n}^3}\right)^p\right). \tag{5.10}$$

Mit Hilfe der beiden Formeln (5.9) und (5.10) können wir nun testen, ob die Gleichung $\phi_n^2 + q = t_n\phi_n$ für diese feste Zahl t_n und für alle n-Torsionspunkte gilt. Da wir (5.8) nur für alle n-Torsionspunkte testen wollen, können wir alle nachfolgenden Rechnungen nach Satz 5.1 modulo des Polynoms $f_n(x)$ durchführen. Vergleichen wir zuerst die x-Koordinaten von (5.9) und (5.10). Dann erhalten wir durch Umformen von (5.9) und (5.10) folgende Gleichung

$$f_x(x,y) = \left((\psi_{q-1}\psi_{q+1} - \psi_q^2(x^{p^2} + x^p + x))\cdot\beta^2 + \psi_q^2\cdot\alpha\right)\psi_{t_n}^{2p} + \psi_{t_n-1}^p\psi_{t_n+1}^p\beta^2\psi_q^2.$$

Dabei seien α und β so gewählt, daß $\lambda = \frac{\alpha}{\beta}$ ist, d.h.

$$\begin{aligned}\alpha &= \psi_{q+2}\psi_{q-1}^2 - \psi_{q-2}\psi_{q+1}^2 - 4y^{p^2+1}\psi_q^3\\ \beta &= 4\psi_q y((x - x^{p^2})\psi_q^2 - \psi_{q-1}\psi_{q+1}).\end{aligned}$$

Natürlich sollten wir auch in dem Polynom $f_x(x,y)$ alle y^2-Terme wieder reduzieren. Durch Fallunterscheidung kann man leicht zeigen, daß

$$f_x'(x) = f_x(x,y)\big/(y^2 - x^3 - ax - b) \in \mathbb{F}_p[x]$$

ist. Also müssen wir zum Vergleich der x-Koordinaten von (5.9) und (5.10) das Polynom f_x' mod $f_n(x)$ berechnen. Ist dieses Polynom ungleich 0, so wissen wir mit Sicherheit, daß für diesen festen Wert von t_n die Relation $\phi_n^2 + q = t_n\phi_n$ nicht für alle n-Torsionspunkte erfüllt sein kann. Dann war die Wahl von t_n falsch und wir wiederholen die Berechnungen des zweiten Teils mit einem anderen Wert für t_n. Erhalten wir jedoch $f_x'(x) \equiv 0$ mod $f_n(x)$, so können wir durch Vergleich der y-Koordinaten von (5.9) und (5.10) feststellen, ob t_n wirklich der korrekte Wert ist.

Durch Gleichsetzen der y-Koordinaten aus (5.9) und (5.10) und Umformen der dann entstehenden Gleichung erhalten wir

$$\begin{aligned}f_y(x,y) &= 4y^p\psi_{t_n}^{3p}\left(((2x^{p^2} + x)\psi_q^2 - \psi_{q-1}\psi_{q+1})\cdot\alpha - y^{p^2}\beta\psi_q^2\right)\\ &\quad -\beta\psi_q^2\left(\psi_{t_n+2}\psi_{t_n-1}^2 - \psi_{t_n-2}\psi_{t_n+1}^2\right)^p.\end{aligned}$$

Hierin stehen α und β für dieselben Terme wie im Fall des Tests der x-Koordinaten. Wiederum reduzieren wir die y^2-Terme und erhalten

$$f_y'(x,y) = f_y(x,y)\big/(y^2 - x^3 - ax - b).$$

Dann ist $f'_y(x, y)$ entweder ein Polynom aus $I\!F_p[x]$ oder aus $y \cdot I\!F_p[x]$. Aus den gleichen Gründen wie oben können wir auch diesmal die Berechnung von $f'_y(x, y)$ wieder modulo $f_n(x)$ ausführen. Ist auch dieses Polynom das Nullpolynom, dann war auch der Test der y-Koordinate erfolgreich und damit ist nach Satz 5.5 der Wert von $t \bmod n$ berechnet. Ansonsten war die Wahl für t_n nicht korrekt und wir müssen den zweiten Teil mit einem anderen Wert für t_n wiederholen.

Führen wir diese Tests für alle $1 \leq t_n < n$ durch, so finden wir mit Sicherheit den "korrekten Wert" t_n, für den $\phi_n^2 + q = t_n \phi_n$ für alle n-Torsionspunkte gilt. Sobald wir nämlich den Wert $t_n \equiv t \bmod n$ gewählt haben, sind sowohl der Test der x-Koordinate als auch der Test der y-Koordinate erfolgreich und wir wissen daher, daß dies der richtige Wert für t_n ist.

5.2.2 Bestimmung der Gruppenordnung

Wir haben jetzt beschrieben, wie wir durch Rechnungen im Polynomring $I\!F_p[x, y]$ den Wert von t_n berechnen kann, so daß $\phi_n^2 - t_n \phi_n + q = 0$ für alle n-Torsionspunkte gilt. Wir haben weiterhin schon bewiesen, daß $t_n \equiv t \bmod n$ ist, wobei t die Konstante aus der Gleichung $\#E(I\!F_p) = p + 1 - t$ für die Gruppenordnung ist. Wir wissen aus dem Satz von Hasse, daß $|t| \leq 2\sqrt{p}$ ist. Nun bestimmen wir eine Menge von ungeraden Primzahlen $\{p_1, \ldots, p_k\}$, so daß

$$\prod_{i=1}^{k} p_i \geq 4\sqrt{p}$$

ist. Dann berechnen wir, wie oben geschildert, die Werte $t_{p_i} \equiv t \bmod p_i$ für alle $1 \leq i \leq k$. Da die Primzahlen $p_1, \ldots, p_k$ paarweise teilerfremd sind, können wir dann mit Hilfe des Chinesischen Restsatzes ([Co91, Seite 16]) eine Lösung von

$$t' \equiv t \bmod \left(\prod_{i=1}^{k} p_i \right)$$

berechnen. Falls wir t' als absolut kleinsten Rest wählen, so erfüllt t' genau die Bedingung des Satzes von Hasse. Somit haben wir die Gruppenordnung $\#E(I\!F_p)$ berechnet als

$$\#E(I\!F_p) \quad = \quad p + 1 - t'.$$

5.3 Der Algorithmus in Formeln

In diesem Abschnitt werden wir den Algorithmus explizit angeben, so daß man ihn direkt implementieren kann. Dabei verwenden wir die Formeln aus dem vorherigen Abschnitt. Sei also nun eine elliptische Kurve $E : y^2 = x^3 + ax + b$ über dem Primkörper $I\!F_p$ fest vorgegeben.

Wir nehmen im weiteren an, daß es eine Funktion 'y_reduce' gibt, die in einem Polynom $f(x,y) \in \mathbb{F}_p[x,y]$ alle y^2-Terme durch x^3+ax+b ersetzt und das dann entstehende Polynom zurückgibt. Zuerst geben wir die Beschreibung des Vorgehens zur Berechnung von $t \bmod n$ für festes n an. Dabei setzen wir voraus, daß alle benötigten Divisionspolynome $\psi_i(x,y)$ und reduzierten Divisionspolynome $f_i(x)$ für die elliptische Kurve der Eingabe schon berechnet wurden und verfügbar sind.

Algorithmus 5.1 (Berechnung von $t \bmod n$)

Eingabe: *Primzahl* $n > 2$.
Ausgabe: $t \bmod n$.

1. /* *Fall 1* */
   (a) *Setze* $q = min\{q' \geq 0\ ;\ q' \equiv p \bmod n\}$.
   (b) *Berechne* $h(x) = y\_reduce\left((x^{p^2} - x)\psi_q^2 + \psi_{q-1}\psi_{q+1}\right)$.
   (c) *Berechne* $g(x) = ggt(h(x), f_n(x))$.
   (d) *Falls* $Grad(g(x)) = 0$, *gehe zu 2.* .
   (e) *Falls* $\left(\frac{q}{n}\right) = -1$, *so terminiere mit Ergebnis* $t \equiv 0 \bmod n$.
   (f) *Berechne* w *mit* $w^2 \equiv q \bmod n$.
   (g) *Berechne* $h'(x) = y\_reduce((x^p - x)\psi_w^2 + \psi_{w-1}\psi_{w+1})$.
   (h) *Berechne* $g'(x) = ggt(h'(x), f_n(x))$.
   (i) *Falls* $Grad(g'(x)) = 0$, *terminiere mit Ergebnis* $t \equiv 0 \bmod n$.
   (j) *Berechne* $h''(x,y) = y\_reduce\left(4y^{(p+1)}\psi_w^3 - \psi_{w+2}\psi_{w-1}^2 + \psi_{w-2}\psi_{w+1}^2\right)$.
   (k) *Berechne* $g''(x) = ggt(h''(x), f_n(x))$.
   (l) *Falls* $Grad(g''(x)) = 0$, *terminiere mit Ergebnis* $t \equiv -2w \bmod n$, *ansonsten terminiere mit Ergebnis* $t \equiv 2w \bmod n$.

2. /* *Fall 2* */
   (a) *Berechne*
   $$\alpha = y\_reduce\left(\psi_{q+2}\psi_{q-1}^2 - \psi_{q-2}\psi_{q+1}^2 - 4(x^3+ax+b)^{(p^2+1)/2}\psi_q^3 \bmod f_n(x)\right).$$
   (b) *Berechne*
   $$\beta = y\_reduce\left(4y\psi_q((x - x^{p^2})\psi_q^2 - \psi_{q-1}\psi_{q+1}) \bmod f_n(x)\right).$$

(c) *Berechne*

$$kl_1(x,y) = y\_reduce\left((\psi_{q-1}\psi_{q+1} - \psi_q^2(x^{p^2} + x^p + x))\beta^2 + \psi_q^2\alpha^2\right) \bmod f_n(x).$$

(d) *Berechne*

$$\begin{aligned} kl_2(x,y) &= y\_reduce\Big(4 \cdot y^p \left(\alpha((2x^{p^2} + x)\psi_q^2 - \psi_{q-1}\psi_{q+1})\right. \\ &\qquad \left.-y^{p^2}\beta\psi_q^2\right)\Big) \bmod f_n(x). \end{aligned}$$

(e) *Für alle Werte* $t_n = 1$ *bis* $n-1$ *tue*

i. *Berechne* $f_x(x) = y\_reduce\left(kl_1 \cdot \psi_{t_n}^{2p} + \psi_{t_n-1}^p\psi_{t_n+1}^p\beta^2\psi_q^2\right) \bmod f_n(x)$.

ii. *Falls* $f_x(x) \neq 0$, *fahre in Schleife fort.*

iii. *Berechne*

$$\begin{aligned} &f_y(x,y) = \\ &\quad y\_reduce\left(\psi_{t_n}^{3p} \cdot kl_2 - \beta\psi_q^2(\psi_{t_n+2}\psi_{t_n-1}^2 - \psi_{t_n-2}\psi_{t_n+1}^2)^p\right) \bmod f_n(x). \end{aligned}$$

iv. *Falls* $f_y(x,y) = 0$, *so terminiere mit Ergebnis* $t \equiv t_n \bmod n$.

In diesem Algorithmus fassen wir natürlich, wie immer in diesem Kapitel, ψ_k als $\psi_k(x,y)$ auf. Die Korrektheit des Algorithmus sollte nach vorherigem Abschnitt klar sein. Die Formeln wurden direkt von dort übernommen. Zusätzlich haben wir noch neue Polynome kl_1 und kl_2 eingeführt, um bestimmte Werte nicht mehrfach zu berechnen. Diese Polynome entsprechen den Formelteilen von $f_x(x,y)$ und $f_y(x,y)$, die nicht von dem aktuellen Wert von t_n abhängen.

Nachdem wir nun den "inneren Teil" des Schoof-Algorithmus angegeben haben, können wir damit den kompletten Schoof-Algorithmus zur Berechnung der Punktanzahl einer elliptischen Kurve über einem Primkörper leicht angeben.

Algorithmus 5.2 (Schoof-Algorithmus)

Eingabe: *elliptische Kurve* $E: y^2 = x^3 + ax + b$ *über Primkörper* $\mathbb{F}_p$.
Ausgabe: *Punktanzahl* $\#E(\mathbb{F}_p)$ *der elliptischen Kurve* E.

1. *Berechne eine Schranke* k, *so daß das Produkt* $\prod_{i=2}^k p_i \geq 4\sqrt{p}$ *ist (dabei ist* p_i *die* i*-te Primzahl).*

2. *Berechne und speichere die Divisionspolynome* $\psi_i(x,y)$ *und die reduzierten Divisionspolynome* $f_i(x)$ *für* $i = -1, \ldots, p_k + 1$.

3. *Berechne mit Hilfe von Algorithmus 5.1 den Wert* $t_i \equiv t \bmod p_i$ *für alle Primzahlen* $p_2, \ldots, p_k$.

4. *Berechne mit dem Chinesischen Restsatz eine Zahl* t, *so daß* $t \equiv t_i \bmod p_i$ *für alle* $i = 2, \ldots, k$. *Dabei sei* t *der absolut kleinste Rest modulo* $\prod_{i=2}^{k} p_i$.

5. *Gib Ergebnis* $p + 1 - t$ *zurück.*

Die Korrektheit dieses Algorithmus sollte ebenfalls nach den vorherigen Aussagen klar sein. Im wesentlichen werden hier nur noch ein paar organisatorische Vorberechnungen unternommen.

5.4 Berechnung von $\#E(\mathbb{F}_p) \bmod 2$

Wir haben in den vorherigen Abschnitten immer Wert darauf gelegt, daß wir t nur modulo ungerader Primzahlen berechnen. Dies war wegen den Voraussetzungen aus Satz 5.1 notwendig. Dennoch ist es auch möglich, den Wert $\#E(\mathbb{F}_p) \bmod 2$ auf elementare Weise zu berechnen. Grundlage dazu ist das folgende Lemma.

Lemma 5.5 *Sei* $P = (x, y) \in E(\mathbb{F}_p), p > 2$. *Dann ist genau dann* $2 \cdot P = \mathcal{O}$, *wenn* $y = 0$ *ist.*

Beweis: Man beachte, daß wir zu $P = (x, y)$ den negativen Punkt als $-P = (x, -y)$ erhalten. Außerdem ist die Gleichung $2 \cdot P = \mathcal{O}$ äquivalent zu $P = -P$. Daraus folgt direkt die Behauptung. ■

Damit können wir leicht das Verhalten von $\#E(\mathbb{F}_p)$ modulo 2 bestimmen. Nach dem Satz von Cauchy ([Hun74, Theorem 5.2, Seite 93]) ist die Gruppenordnung nämlich genau dann gerade, wenn es einen Punkt der Ordnung 2 gibt. Ein Punkt der Ordnung 2 besitzt aber nach obigem Lemma die y-Koordinate 0. Da die Koordinaten der Punkte aus $E(\mathbb{F}_p)$ selbst aus dem Primkörper $\mathbb{F}_p$ sind, kann es einen solchen Punkt also nur geben, wenn $x^3 + ax + b$ eine Nullstelle in $\mathbb{F}_p$ besitzt. Dies können wir durch Berechnung des größten gemeinsamen Teiler von $x^3 + ax + b$ mit dem charakteristischen Polynom des Körpers $\mathbb{F}_p$, also mit $x^p - x$, testen. Besitzt dieser größte gemeinsame Teiler den Grad größer Null, so gibt es mindestens einen Punkt der Ordnung zwei und damit ist die Gruppenordnung gerade. Andernfalls ist die Gruppenordnung ungerade.

Damit können wir auch den Wert von $t \bmod 2$ berechnen und mit im Schoof-Algorithmus verwenden.

Kapitel 6

Laufzeit des Schoof-Algorithmus

Wir haben im vorherigen Kapitel den Schoof-Algorithmus 5.2 zur Berechnung der Punktanzahl einer elliptischen Kurve über einem Primkörper vorgestellt. In diesem Kapitel wollen wir die Komplexität dieses Algorithmus bestimmen und damit die Aussage, daß der Schoof-Algorithmus nur polynomielle Laufzeit besitzt, rechtfertigen. Der Schoof-Algorithmus bestimmt die Gruppenordnung durch Berechnungen mit bestimmten Polynomen. Daher müssen wir zuerst die Komplexität der im Schoof-Algorithmus benutzten Polynomoperationen bestimmen.

Satz 6.1 *Seien f, g zwei Polynome vom Grad $d = d_f \geq d_g$ aus dem Polynomring $\mathbb{F}_p[x]$. Dann können wir*

- *die Summe bzw. Differenz $f \pm g$ in Zeit $O(d \cdot \log(p)^2)$ berechnen.*
- *das Produkt $f \cdot g$ in Zeit $O(d^2 \cdot \log(p)^2)$ bestimmen.*
- *Division mit Rest in Zeit $O((d_f - d_g + 1) \cdot d_g \cdot \log(p)^2)$ durchführen.*
- *den größten gemeinsamen Teiler der beiden Polynome in Zeit $O(d^2 \cdot \log(p)^2)$ berechnen.*

Beweis: Die ersten drei Aussagen dieses Satzes folgen direkt bei Betrachtung der Standardalgorithmen zur Lösung dieser Probleme. Diese findet man beispielsweise in [Kn73, Seite 399ff.].

Nun zur Komplexität der Berechnung des größten gemeinsamen Teilers. Bei Verwendung des Euklid-Algorithmus erhalten wir durch wiederholte Division mit Rest eine Iteration der Form $a_i(x) = q_i(x) \cdot a_{i+1}(x) + a_{i+2}(x)$ (zur Beschreibung des Euklid-Algorithmus vgl. [Kn73]). Sei k die Anzahl der benötigten Iterationen und d_i der Grad von $a_i(x)$. Da wir in jeder Iteration nur eine Division mit Rest durchführen, erhalten wir nach der dritten Behauptung dieses Satzes folgende Komplexität

$$\begin{aligned} T &= \sum_{i=0}^{k} O\left((d_i - d_{i+1} + 1) \cdot d_{i+1} \cdot \log(p)^2\right) \\ &= \sum_{i=0}^{k} O\left((d_i d_{i+1} - d_{i+1}^2 + d_{i+1}) \cdot \log(p)^2\right). \end{aligned}$$

Wenn wir beachten, daß bei der Division mit Rest der Grad des Ergebnisses fällt und damit $d_{i+1} \leq d_i - 1$ gilt, dann erhalten wir daraus

$$\begin{aligned} T &\leq \sum_{i=0}^{k} O\left((d_i^2 - d_i - d_{i+1}^2 + d_{i+1}) \cdot \log(p)^2\right) \\ &= O\left(\sum_{i=0}^{k} (d_i^2 - d_{i+1}^2 + d_{i+1} - d_i) \cdot \log(p)^2\right). \end{aligned}$$

Beachten wir, daß dies eine Teleskopsumme ist, so erhalten wir als Ergebnis

$$T \leq O\left((d_0^2 - d_0) \cdot \log(p)^2\right) = O\left(d_0^2 \cdot \log(p)^2\right).$$

Da bei Berechnung des größten gemeinsamen Teilers von f und g das Polynom $a_0(x) = f(x)$ und damit $d_0 = d$ ist, folgt damit das gewünschte Ergebnis. ■

Damit können wir mit der Analyse der Zeitkomplexität des Schoof-Algorithmus 5.2 beginnen. Als erstes wollen wir den Subalgorithmus 5.1 zur Bestimmung von $\#E(\mathbb{F}_p)$ mod n für festes n betrachten. Dabei sei n eine feste ungerade Primzahl. Algorithmus 5.1 besteht wiederum aus zwei Teilen. Das folgende Lemma beschreibt die Komplexität des ersten Teils dieses Subalgorithmus.

Lemma 6.1 *Der erste Fall zur Berechnung von t mod n in Algorithmus 5.1 benötigt für festes n die Laufzeit $O(n^4 \cdot \log(p)^3)$.*

Beweis: Zum Beweis beachten wir mehrere Beobachtungen:

1. Wenn wir Algorithmus 5.1 betrachten, so erkennen wir, daß wir nie die eigentlichen Divisionspolynome benötigen. Wir können immer schon die reduzierten Divisionspolynome benutzen, so daß der Grad der Variablen y maximal noch eins ist. Dann besteht eine Anwendung der Funktion 'y_reduce()' (d.h. eine Reduktion von evtl. neu entstandenen y^2-Termen) aus maximal einer Multiplikation. Dies gewinnt zusammen mit den beiden folgenden Beobachtungen Bedeutung.

2. Wir müssen die Zwischenergebnisse $h(x), h'(x)$ und $h''(x)$ immer nur modulo das reduzierte Divisionspolynom $f_n(x)$ kennen, denn es gilt

$$\text{ggt}\,(h(x)\,,\,f_n(x)) \quad = \quad \text{ggt}\,(h(x) \bmod f_n(x)\,,\,f_n(x))\,.$$

   Die Korrektheit dieser Gleichung kann man sich leicht überlegen. Damit besitzen nach Lemma 5.2 alle Zwischenergebnisse maximal den Grad $O(n^2)$.

3. Zusätzlich beachten wir, daß wir die Berechnung von $y^{(p+1)}$ in Schritt 1.(j) direkt durch $(x^3 + ax + b)^{(p+1)/2}$ ersetzen können. Da wir auch diesen Ausdruck modulo $f_n(x)$ berechnen können, besitzt auch dieses Polynom den maximalen Grad $O(n^2)$.

4. Damit kann die Berechnung der Polynome $h(x), h'(x)$ und $h''(x)$ mit schneller Exponentiation in Zeit $O(n^4 \cdot \log(p)^3)$ erfolgen. Dies folgt aus der Tatsache, daß alle Zwischenergebnisse den Grad $O(n^2)$ besitzen und wir bei schneller Exponentiation maximal $O\left(\log(p^2)\right) = O\left(\log(p)\right)$ viele Polynommultiplikationen benötigen.

5. Die Berechnung des größten gemeinsamen Teilers in den Schritten 1.(c), 1.(g) und 1.(k) kann nach Satz 6.1 in Zeit $O(n^4 \cdot \log(p)^2)$ erfolgen.

6. Die Berechnung einer Quadratwurzel w mod n in Schritt 1.(f) kann durch Probieren von $w = 2, 3, \ldots$ in Zeit $O\left(n \cdot \log(p)^2\right)$ erfolgen.

7. Alle anderen Schritte des ersten Teils können entweder in konstanter Zeit $O(1)$ oder – wie die Berechnung des Legendre-Symbols in Schritt 1.(e) – in Zeit $O(\log(p)^2)$ durchgeführt werden.

Damit haben wir alle Schritte des ersten Teils von Algorithmus 5.1 behandelt. Durch Vergleich der einzelnen Teilkomplexitäten erhalten wir als Gesamtkomplexität des ersten Teils $O(n^4 \cdot \log(p)^3)$. ∎

Bemerkung 6.1 *Diese Komplexität beschreibt einen kompletten Durchlauf des ganzen ersten Falls von Algorithmus 5.1. Wie wir wissen, ist es jedoch möglich, daß der erste Fall schon in Teilschritt 1.(d) verlassen wird. Da wir dann aber trotzdem die Berechnung von $h(x)$ in Schritt 1.(b) benötigen, erhalten wir auch in diesem Fall die Komplexität $O(n^4 \cdot \log(p)^3)$.*

Nun wollen wir auf die gleiche Weise die Komplexität des zweiten Teils von Algorithmus 5.1 bestimmen. Dabei können wir die gleichen Argumente wie im ersten Teil benutzen und erhalten damit

Lemma 6.2 *Der zweite Teil von Algorithmus 5.1 zur Berechnung von $\#E(\mathbb{F}_p) \bmod n$ besitzt die Zeitkomplexität $O\left(\max\{n^4 \cdot \log(p)^3\,,\, n^5 \cdot \log(p)^2\}\right)$.*

Beweis: Hier gehen wir auf dieselbe Art und Weise wie im Beweis zu Lemma 6.1 vor. Es gelten weiterhin noch die gleichen Aussagen wie im ersten Teil, d.h. wir müssen auch hier alle Zwischenergebnisse nur modulo $f_n(x)$ berechnen. Also treten auch im zweiten Teil nur Zwischenergebnisse mit maximalem Grad $O(n^2)$ auf. Weiterhin können wir auch hier alle y^2-Terme immer direkt reduzieren (siehe z.B. in Schritt 2.(d) und 2.(e)iii). Damit können wir mit derselben Begründung wie in Lemma 6.1 die Polynome α, β, kl_1 und kl_2 aus den Schritten 2.(a) - 2.(d) mit schneller Exponentiation in Zeit $O(n^4 \cdot \log(p)^3)$ bestimmen.

Die Schleife im darauffolgenden Schritt 2.(e) wird sicherlich maximal $O(n)$-mal durchlaufen. Bestimmen wir nun die Laufzeit eines solchen Schleifendurchlaufs. Dabei müssen wir die p-te Potenz eines Divisionspolynoms $\psi_j(x, y)$ bestimmen. Dies können wir effizient durchführen, wenn wir folgende Gleichheit beachten. Für $f(x) \in \mathbb{F}_p[x]$ gilt nämlich

$$f(x)^p \quad = \quad f(x^p).$$

Speichern wir also den schon im ersten Teil berechneten Wert $x^p \bmod f_n(x)$, so können wir das Horner-Schema benutzen, um $f(x)^p$ zu berechnen. Da wir wiederum nur $f(x)^p \bmod f_n(x)$ bestimmen müssen, können wir den maximal auftretenden Polynomgrad durch $O(n^2)$ abschätzen. Dieses Horner-Schema benötigt $Grad(f(x))$ viele Multiplikationen eines Elements aus $\mathbb{F}_p$ (nämlich des jeweiligen Koeffizienten von $f(x)$) mit dem Polynom $x^p \bmod f_n(x)$ sowie jeweils eine Addition von zwei Polynomen. Da diese Multiplikation und Addition in Zeit $O(n^2 \cdot \log(p)^2)$ erfolgen können und $Grad(f(x)) = O(n^2)$ ist, erhalten wir als Laufzeit zur Berechnung von $f(x)^p \bmod f_n(x)$ $O(n^4 \cdot \log(p)^2)$. Weiterhin sollte man beachten, daß für $f(x, y) \in y \cdot \mathbb{F}_p[x]$ das Polynom $f(x, y)^p$ gleich dem Polynom $y^p \cdot g(x)^p$ mit einem Polynom $g(x) \in \mathbb{F}_p[x]$ ist. Daher können wir in diesem Fall $y^p = y \cdot (x^3 + ax + b)^{(p-1)/2} \bmod f_n(x)$ speichern und dann mit einer zusätzlichen Polynommultiplikation das Polynom $f(x, y)^p$ berechnen. Dies ist insbesondere in den Schritten 2.(e)i. und 2.(e)iii. wichtig.

Damit können wir die Laufzeit der ganzen Schleife 2.(e) bestimmen. Ein Schleifendurchlauf braucht nach diesen Bemerkungen Zeit $O(n^4 \cdot \log(p)^2)$. Damit braucht der komplette Schritt 2.(e) Zeit $O(n^5 \cdot \log(p)^2)$.

Durch Vergleich der gerade berechneten Teilkomplexitäten folgt daraus das Lemma. ■

Aus diesen zwei Lemmata können wir nun leicht die Laufzeit der Bestimmung von $t \bmod n$ für festes n herleiten. Dies führt zu

Satz 6.2 *Für festes n berechnet Algorithmus 5.1 den Wert $\#E(\mathbb{F}_p)$ modulo n in Zeit*

$$O\left(\max\{n^4 \cdot \log(p)^3 \, , \, n^5 \cdot \log(p)^2\}\right).$$

Damit haben wir die Laufzeit für einen Durchlauf des inneren Teils des Schoof-Algorithmus 5.2 hergeleitet. Nun wollen wir daraus die Gesamtlaufzeit des Schoof-Algorithmus bestimmen. Dazu ist interessant, für wieviele Primzahlen wir den Subalgorithmus 5.1 verwenden müssen. Dies wird im folgenden Satz analysiert.

Satz 6.3 *Der Schoof-Algorithmus 5.2 berechnet die Punktanzahl einer elliptischen Kurve über dem Primkörper $\mathbb{F}_p$ in $O\left(\log(p)^8\right)$ Bitoperationen.*

Beweis: Um die Anzahl der benötigten Primzahlen abzuschätzen, benutzen wir folgendes Resultat aus [RoSc62]:

$$\text{Für} \quad \theta(x) := \log\left(\prod_{p \le x, p \in \mathbb{P}} p\right) \qquad \text{gilt} \qquad \theta(x) > x\left(1 - \frac{1}{\log(x)}\right).$$

Damit erhalten wir

$$\log\left(\prod_{i=2}^{k} p_i\right) > p_k \cdot \left(1 - \frac{1}{\log(p_k)}\right)$$

und da für $k \ge 3 \quad 1 - \frac{1}{\log(p_k)} > 0.5$ ist, ist sicherlich $\prod_{i=2}^{k} p_i \ge 4\sqrt{p}$, wenn

$$p_k > 2 \cdot \log(4\sqrt{p})$$

ist. Daraus folgt, daß p_k von der Größenordnung $O(\log(p))$ ist. Weiterhin gilt nach dem Primzahlsatz ([Ke82, Th. 5.1, Seite 231]), daß die i-te Primzahl größenordnungsmäßig gleich $i \cdot \log(i)$ ist. Damit ist der Index k der größten im Schoof-Algorithmus zu benutzenden Primzahl p_k ebenfalls von der Größenordnung $O(\log(p))$.

Somit müssen wir in Schritt 1 des Schoof-Algorithmus 5.2 $O(\log(p))$ viele Divisionspolynome berechnen. Nach Lemma 5.2 ist der Grad dieser (reduzierten) Polynome maximal $O\left(\log(p)^2\right)$. Anhand der rekursiven Gleichungen aus der Definition 5.1 der Divisionspolynome können wir das i-te Divisionspolynom mit Hilfe der vorherigen Divisionspolynome berechnen. Damit brauchen wir zur Berechnung eines Divisionspolynoms eine konstante Anzahl von Multiplikationen bzw. Additionen von vorher berechneten Divisionspolynomen. Da diese maximal den Grad $O\left(\log(p)^2\right)$ besitzen, benötigt nach Satz 6.1 die Berechnung eines Divisionspolynoms Zeit $O\left(\log(p)^6\right)$. Somit dauert die Berechnung aller Divisionspolynome in Schritt 2 des Schoof-Algorithmus 5.2 Zeit $O\left(\log(p)^7\right)$.

Danach benutzen wir für alle Primzahlen $p_i \le p_k$, also für insgesamt $O(\log(p))$ viele, den Subalgorithmus 5.1. Da die Beziehung $n \le p_k = O(\log(p))$ gilt, erhalten wir nach Satz 6.3 für eine Primzahl die Laufzeit $O\left(\log(p)^7\right)$. Da dieser Teilalgorithmus insgesamt für $O(\log(p))$ viele Primzahlen durchlaufen wird, erhalten wir als Laufzeit für Schritt 3 des Schoof-Algorithmus 5.2 $O\left(\log(p)^8\right)$.

Anschließend werden diese Teilergebnisse zu dem Gesamtergebnis zusammen. Dazu benutzen wir den Chinesischen Restsatz. Da wir dabei im wesentlichen $O(\log(p))$ viele Berechnungen des größten gemeinsamen Teilers zweier Zahlen der Größe $O(\log(p))$

durchführen, wird die Laufzeit der beiden Schritte 4 und 5 sicherlich von der Laufzeit des Schritts 3 dominiert.

Damit erhalten wir als Gesamtlaufzeit des Schoof-Algorithmus 5.2

$$O\left(\log(p)^8\right).$$

■

Bemerkung 6.2 *Der Schoof-Algorithmus war der erste bekannte Algorithmus zur Bestimmung der Ordnung der Punktgruppe einer elliptischen Kurve E, der eine polynomielle Laufzeit in der Länge der Eingabe besaß. Außerdem ist er vollständig deterministisch. Leider stellt sich in der Praxis heraus, daß der Schoof-Algorithmus für Primkörper großer Charakteristik zu zeitaufwendig ist. Eine Ursache dafür ist die Tatsache, daß der Grad der Divisionspolynome quadratisch wächst. Für Polynome großen Grades werden die Algorithmen der Polynomrechnung extrem zeitintensiv. So dauerte in einem von uns gerechneten Beispiel die Berechnung von $t \bmod 13$ schon ungefähr eine Stunde. Wenn wir aber $\#E(\mathbb{F}_p) \bmod p$ nur für alle Primzahlen bis 13 berechnen, können wir mit dem Schoof-Algorithmus nur in Primkörpern bis zur Charakteristik $\approx 5.6 \cdot 10^7$ rechnen. Daran erkennen wir schon, daß für Primkörper größerer Charakteristik die Berechnungsdauer sehr schnell übermäßig wächst. Allerdings gibt es erste Ansätze, die Divisionspolynome durch geeignete andere Polynome geringeren Grades zu ersetzen. In diese Richtung tendiert zur Zeit eine Richtung der Forschung nach neuen Algorithmen zur Bestimmung der Gruppenordnung $\#E(\mathbb{F}_p)$.*

Damit haben wir die Laufzeit des Schoof-Algorithmus bestimmt. Anschließend wollen wir auch noch die Platzkomplexität des Schoof-Algorithmus angeben. Dazu formulieren wir den folgenden Satz.

Satz 6.4 *Der Schoof-Algorithmus 5.2 besitzt die Platzkomplexität $O\left(\log(p)^4\right)$.*

Beweis: Wir haben im Beweis zu Satz 6.3 schon festgestellt, daß wir insgesamt $O\left(\log(p)\right)$ viele Divisionspolynome berechnen und dann speichern müssen. Jedes dieser Divisonspolynome besitzt dabei den maximalen Grad $O\left(\log(p)^2\right)$. Da die Koeffizienten der Divisionspolynome Elemente aus $\mathbb{F}_p$ sind und da daher ein Koeffizient Platz $O\left(\log(p)\right)$ benötigt, brauchen wir für ein Divisionspolynom Platz $O\left(\log(p)^3\right)$. Zur Abspeicherung aller Divisionspolynome benötigen wir somit den Platz $O(\log(p)^4)$.

Zusätzlich zu den Divisionspolynomen benötigt der Schoof-Algorithmus nur die Abspeicherung von konstant vielen Zwischenergebnissen. Der Platzbedarf für diese Zwischenergebnisse wird sicherlich von $O\left(\log(p)^4\right)$ dominiert und damit ist Satz 6.4 beweisen. ■

Kapitel 7

Kombination der beiden Algorithmen

Wir haben in den vorhergehenden Kapiteln die Algorithmen von Shanks und Schoof zur Berechnung der Punktanzahl einer elliptischen Kurve E über einem Primkörper $\mathbb{F}_p$ vorgestellt. Bei unserer Implementierung machten wir folgende Erfahrungen mit beiden Algorithmen:

- Der Shanks Algorithmus 2.1 ist in der Praxis für alle Primkörper bis zur Charakteristik ungefähr 10^{23} sehr gut geeignet. Wir werden im letzten Kapitel Laufzeitergebnisse angeben, wobei man erkennt, daß dieser Algorithmus für diese Primkörper sehr schnell ist. Für Körper größerer Charakteristik aber tritt das Problem auf, daß die Tabelle, die wir bei den Babysteps anlegen müssen, zuviel Platz belegt. Da wir die Tabelle immer im Hauptspeicher halten, muß der Rechner diese Tabelle häufig ein- und ausswappen. Dadurch tritt eine große Belastung des Rechners nur für organisatorische Arbeiten ein, was zu einer deutlichen Verlangsamung der Rechnung führt. Eine Auslagerung der Tabelle auf externe Speichermedien hat jedoch den Nachteil, daß durch das ständige Suchen in der Tabelle auf diesem externen Speichermedium ebenfalls eine starke Belastung des Rechners erfolgen würde.

- Der Schoof-Algorithmus 5.2 ist nur dazu geeignet, über Primkörpern mit "kleiner" Charakteristik zu arbeiten. Dies wird dadurch verursacht, daß für Polynome größeren Grades die Polynomrechnungen zu zeitaufwendig werden. Je mehr Primzahlen wir also im Schoof-Algorithmus benützen müssen, desto zeitaufwendiger wird der Schoof-Algorithmus. Andererseits können wir für kleine Primzahlen p_i relativ schnell $\#E(\mathbb{F}_p) \bmod p_i$ berechnen.

Wir suchen nun einen Weg, die Tabelle im Shanks Algorithmus zu verkleinern. Eine erste Möglichkeit, dies zu erreichen, erhalten wir einfach, indem wir die Babysteps nach einer bestimmten Anzahl (der gewünschten Tabellengröße) einfach beenden. Dann erhöht sich aber die Anzahl der Giantsteps und damit auch die Laufzeit deutlich, so daß dies keine sehr gute Möglichkeit ist. Eine bessere Variante erhalten wir, wenn wir Teile des Schoof-Algorithmus benutzen und die damit berechnete Information im Shanks Algorithmus benutzen. Die genaue Vorgehensweise, erstmals in [BuMü91] erwähnt, werden wir im folgenden vorstellen.

7.1 Die Idee

Wir gehen davon aus, daß wir die Gruppenordnung $\#E(\mathbb{F}_p)$ modulo einer Zahl t kennen, d.h. sei $\#E(\mathbb{F}_p) \equiv e \bmod t$ bekannt. Diese Information können wir, falls $t = p_1 \cdot \ldots \cdot p_{k'}$ ist, mit dem inneren Teil des Schoof-Algorithmus (also Algorithmus 5.1) berechnen. Wir verändern jetzt den Shanks Algorithmus 2.1 ein wenig, um diese Information zu benutzen. Dazu wählen wir die Konstante k des Schritts 2 – also die Schrittweite der Giantsteps – so, daß k die kleinste natürliche Zahl größer als $\lceil 2\sqrt[4]{p}\,\rceil$ ist, die von t geteilt wird.

Wie hilft uns dies weiter? Im Schritt 4 des Shanks Algorithmus 2.1 bestimmen wir Paare (j, i), so daß für einen zufälligen Punkt P

$$(\lfloor p + 1 - 2\sqrt{p}\,\rfloor + j \cdot k - i) \cdot P = \mathcal{O}$$

ist. Wir wissen schon, daß es mindestens ein solches Paar geben muß, nämlich das Paar (j', i'), das die Gruppenordnung $\#E(\mathbb{F}_p)$ darstellt. Daher erhalten wir

$$\lfloor p + 1 - 2\sqrt{p}\,\rfloor + j' \cdot k - i' \quad = \quad \#E(\mathbb{F}_p) \quad \equiv \quad e \bmod t.$$

Da wir $k \equiv 0 \bmod t$ gewählt haben, ist damit auch $j \cdot k \equiv 0 \bmod t$ für alle natürlichen j. Also muß

$$i' \equiv (\lfloor p + 1 - 2\sqrt{p}\,\rfloor) - e \bmod t$$

sein. Damit müssen wir nur noch bestimmte Punkte in der Tabelle abspeichern, nämlich die Punkte $i \cdot P$, für die $i \equiv (\lfloor p+1-2\sqrt{p}\,\rfloor) - e \bmod t$ ist. Dies vermindert die Größe der Tabelle T um den Faktor $\frac{1}{t}$. Damit haben wir unser erstes Ziel erreicht, wir haben die Tabelle verkleinert. Dies hat nun jedoch eventuell zur Folge, daß wir eine extrem kleine Tabelle (und damit sehr wenige Babysteps) haben, daß aber die Anzahl der Giantsteps sich nicht verändert hat. Wir wissen jedoch schon, daß der Shanks Algorithmus optimal ist, wenn die Anzahl der Babysteps und der Giantsteps gleich sind. Damit sollten wir die Anzahl der Babysteps und der Giantsteps auch hier angleichen. Dazu müssen wir die folgende Gleichung lösen:

$$k \quad = \quad \frac{4\sqrt{p}}{kt}.$$

Darin sei k die Anzahl der Babysteps. Dann haben wir nach Durchführung der Babysteps ein Intervall der Länge $k \cdot t$ durchsucht und demnach beschreibt die rechte Seite dieser Gleichung die Anzahl der dann noch benötigten Giantsteps. Durch Lösen dieser Gleichung erhalten wir

$$k = \frac{2\sqrt[4]{p}}{\sqrt{t}}.$$

Da die Schrittweite der Giantsteps dann $k \cdot t \equiv 0 \bmod t$ ist, können wir k optimal als $\left\lceil \frac{2\sqrt[4]{p}}{\sqrt{t}} \right\rceil$ wählen.

Sei nun c der kleinste positive Rest von $\lfloor p+1-2\sqrt{p} \rfloor - e \bmod t$. Dann müssen wir nach obigen Bemerkungen also die Punkte $(c+i\cdot t)\cdot P$ für $i = 0, \ldots, k-1$ mit dem gerade berechneten Wert für k in der Tabelle abspeichern.

7.2 Der Kombinations-Algorithmus

Wir formulieren jetzt den Kombinations-Algorithmus, der sich aus den Änderungen des Shanks Algorithmus 2.1 zur Benutzung der Information aus einem vorherigen Schoof-Teil ergibt. Dabei beschränken wir uns auf den ersten Teil des Shanks Algorithmus. Wie wir im Kapitel über die erwartete Laufzeit gesehen haben, sind die im zweiten Teil des Shanks Algorithmus angelegten Tabellen von der gleichen Größenordnung wie die Tabellen des ersten Teils. Leider gibt es allerdings keine offensichtliche Möglichkeit, diese Tabellen ebenfalls zu verkleinern. Liegt eine Kurve E vor, bei der den zweiten Teil des Shanks Algorithmus 2.1 benötigt haben, so können wir nun nicht mehr garantieren, daß der Kombinations-Algorithmus die Gruppenordnung korrekt berechnet. Dann können wir lediglich eine Menge von möglichen Gruppenordnungen bestimmen. Diese Menge besitzt allerdings höchstens die Ordnung vier, so daß wir mit einem Algorithmus, der entscheidet, ob eine Lösung korrekt ist, dann die Gruppenordnung leicht bestimmen können. Einen solchen Algorithmus werden wir im weiteren Verlauf dieses Kapitels noch beschreiben. In der Praxis stellt sich aber auch bei dem Kombinations-Algorithmus heraus, daß der zweite Teil nie benötigt wird.

Algorithmus 7.1 (Kombinations-Algorithmus)

Eingabe: *elliptische Kurve E über Primkörper $\mathbb{F}_p$.*
Ausgabe: *Menge von maximal 4 möglichen Werten für die Gruppenordnung $\#E(\mathbb{F}_p)$.*

Schritt 1: *Bestimme mit Hilfe von Algorithmus 5.1 den Wert $\#E(\mathbb{F}_p) \bmod p_i$ für die Primzahlen $p_1, \ldots, p_{k'}$ und eine Konstante k'.*

Schritt 2: *Berechne daraus mit Hilfe des Chinesischen Restsatzes den Wert $e \equiv \#E(\mathbb{F}_p) \bmod t$, wobei $t = \prod_{i=1}^{k'} p_i$.*

Schritt 3: *Wähle einen zufälligen Punkt P aus $E(\mathbb{F}_p)$, setze*

$$k \;=\; \left\lceil \frac{2\sqrt[4]{p}}{\sqrt{t}} \right\rceil \qquad \textit{und} \qquad c \;=\; \min\{x \geq 0;\; x \equiv \lfloor p+1-2\sqrt{p} \rfloor - e \bmod t\}.$$

Schritt 4: *Für $i = 0, \ldots, k-1$ tue*

Falls $(c + i \cdot t) \cdot P = \mathcal{O}$, gehe zurück zu Schritt 3,
ansonsten speichere $((c + i \cdot t) \cdot P, c + i \cdot t)$ in einer Tabelle T.

Schritt 5: *Setze $Q = (k \cdot t) \cdot P$.*

Schritt 6: *Für $j = 1, \ldots, k$ tue*

1. *Berechne den Punkt $H_j = \lfloor\, p+1-2\sqrt{p}\, \rfloor \cdot P + j \cdot Q$.*
2. *Falls $H_j = i \cdot P$ für einen Tabelleneintrag $(i \cdot P, i)$, so speichere das Paar (j, i) ab.*

Schritt 7: *Falls nur ein Paar (j, i) abgespeichert wurde, so gilt*

$$\#E(\mathbb{F}_p) \quad = \quad \lfloor\, p+1-2\sqrt{p}\, \rfloor + k \cdot t \cdot j - i$$

und der Algorithmus terminiert.

Ansonsten berechne mit zwei abgespeicherten Paaren (j, i) und (j', i')

$$l \;=\; ord(P) \;=\; \min\{d\,;\; d > 0,\; d \cdot P = \mathcal{O},\; d \mid k \cdot t \cdot (j - j') - (i - i')\,\}\,.$$

Schritt 8: *Falls $l \geq \sqrt{p} - 1$, so gib die Menge*

$$\{x \in [\, p+1-2\sqrt{p}, p+1+2\sqrt{p}\,],\; x \equiv 0 \bmod l,\; x \equiv e \bmod t\}$$

als Ergebnis zurück, ansonsten gehe zurück zu Schritt 3.

Die Korrektheit von Algorithmus 7.1 sollte nach den Bemerkungen am Anfang dieses Kapitels und den vorherigen Kapiteln klar sein. Diese folgt direkt aus der Korrektheit des Shanks und des Schoof-Algorithmus.

Wir haben am Anfang dieses Abschnitts schon erwähnt, daß wir diesmal den zweiten Teil des Shanks Algorithmus nicht übernehmen. Wie schon zu Beginn dieses Abschnitts erklärt, bestimmt Algorithmus 7.1 die Gruppenordnung korrekt, wenn auch der Shanks Algorithmus 2.1 im ersten Teil terminiert wäre. Sei nun E eine elliptische Kurve, für die der Shanks Algorithmus den zweiten Teil zwingend benötigte. Dann können wir nicht mehr garantieren, daß der Kombinations-Algorithmus 7.1 immer genau die Gruppenordnung bestimmt. Es könnte zum Beispiel sein, daß die Ordnung des Punktes P und die Gruppenordnung $\#E(\mathbb{F}_p)$ beide von t geteilt werden. Damit sind auch alle Vielfachen der Punktordnung im Hasse-Intervall Kandidaten für die Gruppenordnung. Da diese Punktordnung aber mindestens die Größe $\sqrt{p}-1$ besitzt, wenn der Algorithmus in Schritt 8 terminiert, kann es maximal vier Werte im Hasse-Intervall geben, die diese Bedingung erfüllen. Andererseits kann Algorithmus 7.1 aber auch für solche Kurven die Gruppenordnung $\#E(\mathbb{F}_p)$ eindeutig berechnen, wenn beispielsweise mehrere Vielfache der Ordnung des Punktes P im Hasse-Intervall liegen, aber nur ein Vielfaches davon kongruent zu $e \bmod t$ ist.

7.3 Die Implementierung

Die Implementierung des Kombinationsalgorithmus 7.1 stützt sich im wesentlichen auf die schon in Kapitel 4 vorgestellte Implementierung des Shanks Algorithmus. Als neuer Teil kommt die Implementierung des Subalgorithmus 5.1 des Schoof-Algorithmus hinzu. Deshalb beschreiben wir zuerst unsere Implementierung des Algorithmus 5.1 .

7.3.1 Die Implementierung der Berechnung von $\#E(\mathbb{F}_p) \bmod n$

In Algorithmus 5.1 wird durch Rechnung mit Polynomen das Verhalten der Gruppenordnung $\#E(\mathbb{F}_p)$ modulo einer ungeraden Primzahl n berechnet. Zur dazu benötigten Polynomarithmetik können wir wiederum die Hilfe des Computeralgebrasystems PARI benutzen.

Ein Polynom $f(x,y) \in \mathbb{F}_p[x,y]$ wird in PARI dargestellt als Vektor der Länge $\mathrm{Grad}_x(f)+1$. Dabei enthält der nullte Vektoreintrag wie bei den langen Zahlen statistische Daten. Alle anderen Vektoreinträge verweisen auf die Koeffizienten der Variablen x^i, aufsteigend sortiert. Ist $f(x,y)$ ein Polynom in den zwei Variablen x und y, so verweisen die anderen Vektoreinträge auf Polynome nur noch einer Variablen y. Hier wird wiederum der rekursive Aufbau von PARI deutlich.

In Algorithmus 5.1 benötigen wir zuerst die Divisionspolynome für die spezielle elliptische Kurve der Eingabe. Daher legen wir im Hauptprogramm zuerst eine obere Schranke l für die Primzahlen n fest, für die wir den Wert $\#E(\mathbb{F}_p) \bmod n$ berechnen wollen. Danach berechnen wir alle benötigten Divisionspolynome (also die Polynome $\psi_{-1},\ldots,\psi_{l+1}$) und speichern sie auf dem PARI-Stack. Zur Berechnung verwenden wir die rekursiven Formeln aus Definition 5.1. Nach der Berechnung reduzieren wir noch vor dem Abspeichern alle y^2-Terme. Die Berechnung der Divisionpolynome findet man im Listing in der Funktion “div_pol()”.

Zur Reduktion der y^2-Terme eines Polynoms durch die rechte Seite $x^3 + ax + b$ der Kurvengleichung haben wir eine eigene Funktion "y_reduce()" geschrieben. In dieser Funktion überprüfen wir als erstes, ob das Eingabepolynom $f(x,y) \in \mathbb{F}_p[x,y]$ von der speziellen Form $f(x,y) = g(x)$ oder $f(x,y) = y \cdot g'(x)$ ist. In diesen beiden Fällen sind wir direkt fertig und können das Eingabepolynom zurückgeben. Andernfalls ordnen wir das Polynom $f(x,y)$ zuerst nach y-Koordinaten. Dies ist in PARI relativ kompliziert, da die Variablen in PARI eine Priorität besitzen und die Priorität der Variablen x größer als die der Variablen y ist, d.h. Polynome der Variablen x und y sind immer nach x-Koordinaten geordnet. Deshalb lösen wir dieses Problem durch Umbenennung der Variablen x in z. Dann können wir das Hornerschema benutzen und damit alle y^2-Terme reduzieren. Anschließend müssen wir wieder eine Rücktransformierung der Variablen z in die Variable x durchführen.

Aus der Beschreibung von Algorithmus 5.1 auf Seite 70 wird weiterhin deutlich, daß wir eine Funktion benötigen, die eine Funktion $f(x,y) \in \mathbb{F}_p[x,y]$ potenziert. Diese Aufgabe wird bei uns von der Funktion "polmod()" erledigt. Dabei untersuchen wir $f(x,y)$ zuerst wiederum, ob es ein Polynom in einer Variablen x, ein Polynom aus $y \cdot \mathbb{F}_p[x]$ oder ein allgemeines Polynom aus $\mathbb{F}_p[x,y]$ ist. Die beiden ersten Arten von Polynomen können wir besser behandeln als in der schnellen Exponentiation. Dazu verwenden wir den schon in Kapitel 6 beschriebenen Trick. Wir können $f(x)^p \bmod f_n(x)$ schnell berechnen als $f(x^p) \bmod f_n(x)$. Daher speichern wir nach seiner Berechnung das Polynom $x^p \bmod f_n(x)$ und benutzen es später immer wieder bei der Potenzierung anderer Polynome. Ist das Eingabepolynom $f(x,y)$ von der Form $y \cdot g(x)$, dann können wir diesen Trick genauso verwenden. Wir speichern nach der ersten Berechnung das Polynom $y^p = y \cdot (x^3 + ax + b)^{(p-1)/2} \bmod f_n(x)$ ab und können es dann später bei der Potenzierung von Polynomen dieser zweiten Art immer wieder benutzen. Dieser Trick ist hier immer wieder anwendbar, da wir fast immer genau die p-te Potenz eines Polynoms berechnen müssen. Nur falls das Eingabepolynom $f(x,y)$ wirklich die dritte Form besitzt bzw. wenn die Polynome $x^p \bmod f_n(x)$ oder $y^p \bmod f_n(x)$ noch nicht berechnet sind, müssen wir wirklich schnelle Exponentiation benutzen. Dabei müssen wir darauf achten, daß nach jeweils einigen Durchläufen eventuell neu entstandene y^2-Terme reduziert werden.

Ansonsten wird Algorithmus 5.1 genauso implementiert, wie wir in auf Seite 70 beschrieben haben. Wir haben jedoch darauf geachtet, daß wir alle Terme, die wir später wieder benötigen, abspeichern, um sie nicht neu berechnen zu müssen. Dies ist insbesondere in der Schleife des Schritts 2.(e) von Bedeutung. Dort benötigen wir in jedem Schleifendurchlauf die Polynome

$$\psi_{t_n-1}^p \bmod f_n(x), \quad \psi_{t_n}^p \bmod f_n(x), \quad \psi_{t_n+1}^p \bmod f_n(x),$$

wobei t_n ein Laufindex ist. Mit diesem Trick können wir uns in jedem Schleifendurchlauf auf die Berechnung eines neuen Wertes $\psi_i^p \bmod f_n(x)$ beschränken.

7.3.2 Die Veränderungen am Shanks Algorithmus

Gegenüber der Beschreibung der Implementierung des Shanks Algorithmus 2.1 in Kapitel 4 bestehen bei der Implementierung des Kombinations-Algorithmus 7.1 nur leichte Unterschiede. Wir beginnen im Hauptprogramm hier damit, daß wir eine obere Grenze für die Primzahlen, die wir im Subalgorithmus 5.1 verwenden wollen, einlesen. Damit können wir dann t als das Produkt aller Primzahlen unterhalb dieser Grenze berechnen und damit den PARI-Stack initialisieren. Dabei legen wir die Größe fest als $\left\lceil\frac{2\sqrt[4]{p}}{\sqrt{t}}\right\rceil \cdot ZL \cdot 4$ Bytes plus einen Sicherheitsbereich (bei uns 400000 Byte). In dieser Formel sei ZL die Anzahl der 32-Bit-Wörter, die wir zur Abspeicherung eines Tabellenelements (also Anzahl der 32-Bit-Wörter zur Abspeicherung von p plus ein Wort zur Abspeicherung des Index) benötigen.

Anschließend berechnen wir mit dem Subalgorithmus 5.1 den Wert $\#E(\mathbb{F}_p) \bmod n$ für alle ungeraden Primzahlen n unterhalb unserer oberen Grenze. Zusätzlich berechnen wir mit dem Trick aus Kapitel 5 auch den Wert $\#E(\mathbb{F}_p) \bmod 2$. Diese Informationen setzen wir dann mit dem Chinesischen Restsatz, implementiert in der PARI-Funktion "chinois()", zusammen.

Bei den anschließenden Babysteps gehen wir genauso vor, wie wir es im ersten Abschnitt dieses Kapitels beschrieben haben. Zuerst berechnen wir die optimale Anzahl der Babysteps und dann bestimmen wir den kleinsten positiven Rest von $c \equiv \lfloor p+1-2\sqrt{p}\rfloor - \#E(\mathbb{F}_p) \bmod t$. Dann führen wir die Babysteps wie in Algorithmus 7.1 beschrieben durch. Dabei speichern wir wiederum nur die x-Koordinaten der Punkte $i \cdot P$ und den Index i ab. Leider speichern wir damit in diesem Algorithmus nicht mehr zwei Punkte implizit ab, denn im allgemeinen ist $i \not\equiv -i \bmod t$. Nach den Babysteps sortieren wir wie im "normalen" Shanks Algorithmus die Tabelle mit dem Sortieralgorithmus Quicksort.

Die Giantsteps laufen ebenfalls genauso ab wie im Shanks Algorithmus, abgesehen daß die Schrittweite um den Faktor t wächst. Auch in dieser Implementierung beenden wir die Giantsteps sofort, wenn wir ein Vielfaches der Punktordnung gefunden haben. Dann bestimmen wir die Punktordnung durch Faktorisierung und Ausprobieren der einzelnen Faktoren, wie schon in Kapitel 4 geschildert. Anschließend müssen wir nur noch überprüfen, ob nur ein Vielfaches dieser Punktordnung im Hasse-Intervall liegt, das kongruent zu $\#E(\mathbb{F}_p) \bmod t$ ist. Dies ist dann die Gruppenordnung $\#E(\mathbb{F}_p)$.

Auch in dieser Implementierung müssen wir aus praktischen Gründen die Größe der Tabelle nach oben beschränken. Überschreitet die optimale Tabellengröße diesen Wert, so müssen wir die Tabelle künstlich einschränken und die Anzahl der Giantsteps entsprechend erhöhen. Aus den gleichen Gründen wie in Kapitel 4 müssen wir weiterhin dafür sorgen, daß die Größe des PARI-Stacks nicht zu klein wird.

7.4 Erwartete Laufzeit des Kombinations-Algorithmus

In diesem Abschnitt wollen wir untersuchen, ob die erwartete Laufzeit des Kombinations-Algorithmus 7.1 besser ist als die erwartete Laufzeit des Shanks Algorithmus 2.1. Dabei spielt natürlich die Anzahl der Primzahlen, die wir in Schritt 1 verwenden, eine große Rolle. Nehmen wir zuerst einmal an, diese Anzahl wäre konstant. Dann erhalten wir den folgenden Satz über die zu erwartende Laufzeit des Kombinations-Algorithmus 7.1.

Satz 7.1 *Der Kombinationsalgorithmus 7.1 hat bei Benutzung einer konstanten Anzahl von Primzahlen in Schritt 1 diesselbe erwartete Laufzeit wie der erste Teil des Shanks Algorithmus 2.1, d.h. er besitzt ebenfalls die erwartete Laufzeit*

$$O\left(\sqrt[4]{p} \cdot \log(p)^2 \cdot (\log\log(p))^2\right).$$

Beweis: Sei die Anzahl der in Schritt 1 benutzten Primzahlen konstant, etwa k'. Dann sind sicherlich alle Primzahlen nach oben durch eine Zahl p' beschränkt. Nach Satz 6.2 kann dann der Wert $\#E(\mathbb{F}_p) \bmod p_i$ für ein $i \leq k'$ in Zeit $O\left(p'^4 \cdot \log(p)^3\right)$ berechnet werden. Damit besitzt der erste Schritt von Algorithmus 7.1 die (deterministische) Laufzeit $O\left(k' \cdot p'^4 \cdot \log(p)^3\right) = O\left(\log(p)^3\right)$, denn sowohl k' als auch p' sind Konstanten.

Nach Satz 3.5 beträgt die erwartete Laufzeit des ersten Teils des Shanks Algorithmus 2.1 $O\left(\sqrt[4]{p} \cdot \log(p)^2 \cdot (\log\log(p))^2\right)$. An der Analyse, die wir in Kapitel 3 unternommen haben, ändert sich jetzt nur die Tabellengröße. Diese schrumpft aber nur um einen konstanten Faktor (da k' konstant), so daß dies keine Auswirkungen auf die Komplexität besitzt. Damit besitzt auch Algorithmus 7.1 die Komplexität

$$O\left(\sqrt[4]{p} \cdot \log(p)^2 \cdot (\log\log(p))^2 + \log(p)^3\right) = O\left(\sqrt[4]{p} \cdot \log(p)^2 \cdot (\log\log(p))^2\right).$$

■

Bemerkung 7.1 *In der Praxis benutzen wir diesen Kombinations-Algorithmus immer nur mit einer konstanten Anzahl von Primzahlen im ersten Schritt. Praktisch stellte sich dabei heraus, daß dies dennoch eine deutliche Verbesserung gegenüber dem normalen Shanks Algorithmus bedeutet. Wird allerdings die Anzahl der benutzten Primzahlen zu groß, so wächst die praktische Laufzeit rapide an. Dies liegt an der extrem wachsenden (praktischen) Laufzeit des Algorithmus 5.1. Wir benutzen daher diesen inneren Schoof-Teil nur für alle Primzahlen bis einschließlich 13.*

Nun kann man sich die Frage stellen, wie sich die Komplexität verhält, wenn wir die Anzahl der Primzahlen in Schritt 1 nicht konstant wählen, sondern auch anwachsen lassen. Es wäre denkbar, daß wir dann weniger Primzahlen als im normalen Schoof-Algorithmus benötigen und dennoch eine logarithmische Laufzeit für den Kombinations-Algorithmus erhalten. Dies wird aber im folgenden Satz widerlegt.

Satz 7.2 *Benutzt man in Schritt 1 des Kombinations-Algorithmus 7.1 soviele Primzahlen, daß die Laufzeit des Kombinations-Algorithmus polynomiell in der Länge der Eingabe wird, so benutzt man größenordnungsmäßig genausoviele Primzahlen wie im Schoof-Algorithmus 5.2 auch.*

Beweis: Wir haben in Satz 6.3 schon gesehen, daß wir im Schoof-Algorithmus $O\left(\log(p)\right)$ viele Primzahlen verwenden müssen, um die Ordnung der Punktgruppe einer elliptischen Kurve zu berechnen. Die erwartete Laufzeit der Schritte 2 bis 8 des Kombinations-Algorithmus unterscheidet sich sicherlich von der erwarteten Laufzeit des Shanks Algorithmus nur in dem Anteil, der von der Anzahl der Babysteps bzw. Giantsteps herrührt. Damit erhalten wir als erwartete Laufzeit des Algorithmus 7.1

$$O\left(\frac{\sqrt[4]{p}}{\sqrt{t}} \cdot T(\log(p))\right),$$

wobei $T(p)$ ein Polynom ist. Wir wollen nun $t = \prod_{i=1}^{k'} p_i$ so wählen, daß auch der Anteil $\frac{\sqrt[4]{p}}{\sqrt{t}}$ polynomiell wird, d.h. daß $\frac{\sqrt[4]{p}}{\sqrt{t}} \leq c \cdot \log(p)^j$ für eine natürliche Zahl j und eine Konstante c. Damit erhalten wir

$$\prod_{i=1}^{k'} p_i = t \geq \frac{c' \sqrt{p}}{\log(p)^{2j}} \qquad \text{für eine natürliche Zahl } j.$$

Wir wissen aus [RoSc62], daß $\log(t) \leq c \cdot p_{k'}$ ist. Daher gilt

$$p_{k'} \geq c' \cdot \log(t) \approx \frac{c'' \cdot \log(p)}{\log\log(p)} \quad = \quad O(\log(p))$$

und nach dem Primzahlsatz ist damit auch k' von der Größenordnung $O(\log(p))$. Damit müssen wir auch in diesem Fall $O(\log(p))$ viele Primzahlen, also größenordnungsmäßig diesselbe Anzahl wie im Schoof-Algorithmus, in Schritt 1 benutzen. Deshalb bedeutet der Kombinations-Algorithmus dann keine Verbesserung gegenüber dem Schoof-Algorithmus. ■

7.5 Verifikation der Gruppenordnung

Sei im folgenden $E : y^2 = x^3 + ax + b$ eine feste elliptische Kurve über dem Primkörper $\mathbb{F}_p$. Wir kennen einen "möglichen" Wert $p + 1 - c$ für die Gruppenordnung und wollen beweisen, daß dieser Wert wirklich die Gruppenordnung ist. Als Grundlage eines Algorithmus zur Lösung dieses Problems ([At91]) benötigen wir zuerst folgende Proposition.

Proposition 7.1 *Sei $E : y^2 = x^3 + ax + b$ eine elliptische Kurve über dem Primkörper $\mathbb{F}_p$ mit Ordnung $\#E(\mathbb{F}_p) = p + 1 - c$. Sei weiterhin $d \in \mathbb{F}_p$ ein quadratischer Nichtrest modulo p. Dann besitzt die elliptische Kurve*

$$E' : y^2 = x^3 + (ad^2)x + (bd^3)$$

die Ordnung $\#E'(\mathbb{F}_p) = p + 1 + c$.

Beweis: Zuerst beweisen wir, daß E' wirklich eine elliptische Kurve ist, d.h. daß E' nicht-singulär ist. Dies folgt jedoch direkt aus

$$4a'^3 + 27b'^2 = 4a^3d^6 + 27b^2d^6 = (4a^3 + 27b^2)d^6 \neq 0,$$

da E eine elliptische Kurve und somit $4a^3 + 27b^2 \neq 0$ ist.

Nun zum Beweis des zweiten Teils der Proposition, d.h. der Behauptung über die Ordnung von $E'(\mathbb{F}_p)$. Wir wissen nach Lemma 1.2, daß

$$\#E(\mathbb{F}_p) = p + 1 + \sum_{x \in \mathbb{F}_p} \left(\frac{x^3 + ax + b}{p} \right)$$

ist. Also ist

$$-c = \sum_{x \in \mathbb{F}_p} \left(\frac{x^3 + ax + b}{p} \right).$$

Benutzen wir, daß d ein quadratischer Nichtrest modulo p ist, daß also $\left(\frac{d}{p}\right) = -1$ ist, so erhalten wir

$$\begin{aligned}
c &= \sum_{x \in \mathbb{F}_p} - \left(\frac{x^3 + ax + b}{p} \right) \\
&= \sum_{x \in \mathbb{F}_p} \left(\frac{d}{p} \right) \cdot \left(\frac{x^3 + ax + b}{p} \right) \\
&= \sum_{x \in \mathbb{F}_p} \left(\frac{d^3}{p} \right) \cdot \left(\frac{x^3 + ax + b}{p} \right) \\
&= \sum_{x \in \mathbb{F}_p} \left(\frac{(xd)^3 + ad^2(xd) + bd^3}{p} \right).
\end{aligned}$$

Da $d \neq 0 \bmod p$ und $\mathbb{F}_p$ ein Körper ist, ist die Menge $\{xd \ ; \ x \in \mathbb{F}_p\}$ gleich $\mathbb{F}_p$ und damit erhalten wir für diese Summe

$$c = \sum_{x' \in \mathbb{F}_p} \left(\frac{x'^3 + (ad^2)x' + (bd^3)}{p} \right).$$

Also besitzt die Kurve E' genau die Ordnung $p+1+c$, denn nach der Definition der Kurve E' und Lemma 1.2 gilt

$$\begin{aligned} \#E'(\mathbb{F}_p) &= p+1+\sum_{x'\in\mathbb{F}_p}\left(\frac{x'^3+(ad^2)x'+(bd^3)}{p}\right) \\ &= p+1+c. \end{aligned}$$

■

Sei im folgenden $E(\mathbb{F}_p)$ eine elliptische Kurve über $\mathbb{F}_p$ und sei $e = p+1-c$ die vermeintliche Ordnung von $E(\mathbb{F}_p)$. Weiterhin sei $e' = p+1+c$. Dann bestimmen wir zuerst jeweils eine Zerlegung von e und e' in

$$e = f_1 \cdot u_1 \qquad \text{und} \qquad e' = f_2 \cdot u_2,$$

wobei die f_i $(i = 1,2)$ Potenzprodukte von Primzahlen kleiner als eine Schranke B und die u_i $(i = 1,2)$ der verbleibende Rest (d.h. Potenzprodukte von Primzahlen größer als B) sind. Diese Werte kann man beispielsweise durch Probedivisionen oder durch Anwendung der ECM-Faktorisierungsmethode (vgl. [Le87]) berechnen.

Dann bestimmen wir mit Hilfe von Proposition 7.1 eine elliptische Kurve E', die die vermeintliche Ordnung $e' = p+1+c$ besitzt. Danach bestimmen wir zu beiden elliptischen Kurven E und E' jeweils eine Untergruppe der Ordnung $f_i' \cdot x_i$ $(i = 1,2)$ mit $f_i'|f_i$, $x_i|u_i$, wobei die x_i unbekannt sind. Dabei sei jedoch mindestens ein $x_i > 1$. Dazu wählen wir zufällig Punkte $P_1 \in E(\mathbb{F}_p)$ bzw. $P_2 \in E'(\mathbb{F}_p)$ aus. Folgendes Lemma liefert die Grundlage der Bestimmung der Werte f_i'.

Lemma 7.1 *Sei $P \in E(\mathbb{F}_p)$ ein beliebiger Punkt der elliptischen Kurve E. Sei $\#E(\mathbb{F}_p) = f \cdot u$, wobei f ein Potenzprodukt von Primzahlen kleiner als eine Schranke B und u der verbleibende Rest ist. Sei die Ordnung von P gegeben als $ord(P) = f' \cdot u'$, wobei $f'|f$ und $u'|u$ gelte. Dann gilt für den Punkt $Q = u \cdot P$*

$$ord(Q) = f'.$$

Beweis: Sei $u = u' \cdot u''$ und $f = f' \cdot f''$. Offensichtlich gilt dann

$$f' \cdot Q = f' \cdot u \cdot P = f' \cdot u' \cdot u'' \cdot P = u'' \cdot (f' \cdot u' \cdot P) = \mathcal{O}.$$

Nehmen wir an, daß ord $(Q) = \tilde{f} < f'$ wäre. Dann muß nach der letzten Gleichung $\tilde{f}$ ein Teiler von f' sein. Betrachten wir daher den Punkt $H = \tilde{f} \cdot u' \cdot P$. Der Punkt H ist sicherlich nicht der Nullpunkt $\mathcal{O}$, da $\tilde{f} < f'$ ist. Deshalb besitzt er eine Ordnung größer Eins. Weiterhin muß die Ordnung von H ein Teiler von u'' sein, denn nach unserer Annahme gilt

$$\mathcal{O} = \tilde{f} \cdot Q = \tilde{f} \cdot u \cdot P = u'' \cdot (\tilde{f} \cdot u' \cdot P) = u'' \cdot H.$$

Falls $u'' = 1$ ist, so liefert dies schon einen Widerspruch. Sei daher $u'' > 1$. Dann gilt aber auch

$$\frac{f'}{\tilde{f}} \cdot H = \frac{f'}{\tilde{f}} \cdot \tilde{f} \cdot u' \cdot P = f' \cdot u' \cdot P = \mathcal{O}.$$

Da aber $\frac{f'}{f} > 1$ nur aus Primzahlen kleiner als B und $u'' > 1$ nur aus Primzahlen größer als B besteht, beides jedoch Vielfache der Ordnung von H sein müssen, führt dies zu einem Widerspruch. Also ist die Ordnung von Q genau f'. ■

Damit können wir – da wir die Faktorisierung der Zahlen f_i kennen– leicht die Anteile f_i' berechnen. Der Test $x_i > 1$ läßt sich ebenfalls leicht bewerkstelligen. Zusätzlich sollten wir für einen gefundenen Punkt P_i noch überprüfen, ob $(p+1-c) \cdot P_1 = \mathcal{O}$ und $(p+1+c) \cdot P_2 = \mathcal{O}$ (auf den jeweiligen Kurven) erfüllt ist. Ist dies nicht erfüllt, dann haben wir sicherlich bewiesen, daß unsere Eingabe nicht die korrekte Gruppenordnung war.

Sei nun die korrekte Gruppenordnung gegeben als $\#E(\mathbb{F}_p) = p + 1 - C$. Dann ist $\mathrm{ord}\,(P_i) = f_i' \cdot x_i$ $(i = 1, 2)$ sicherlich ein Teiler von $|C - c|$. Weiterhin gilt sicherlich $|C - c| \leq 4\sqrt{p}$. Damit wissen wir, daß $\mathrm{kgV}(f_1', f_2') \cdot \mathrm{kgV}(x_1, x_2) \geq \mathrm{kgV}(f_1', f_2') \cdot B$ ein Teiler von $|C-c|$ ist. Mit hoher Wahrscheinlichkeit sind außerdem x_1 und x_2 teilerfremd, so daß dann sogar

$$\mathrm{kgV}(f_1', f_2') \cdot x_1 \cdot x_2$$

eine Teiler von $|C - c|$ ist. Schätzen wir diesen Wert ab, so erhalten wir

$$\mathrm{kgV}(f_1', f_2') \cdot x_1 \cdot x_2 \geq \begin{cases} \mathrm{kgV}(f_1', f_2') \cdot B^2 & \text{wenn } x_1, x_2 > 1. \\ \mathrm{kgV}(f_1', f_2') \cdot B & \text{wenn ein } x_i = 1. \end{cases}$$

Ist also $\mathrm{kgV}(f_1', f_2') \cdot B$ bzw. $\mathrm{kgV}(f_1', f_2') \cdot B^2$ größer als $4\sqrt{p}$, so muß $|C - c|$ gleich Null sein, d.h. unsere Eingabe war die korrekte Gruppenordnung. Ansonsten sollte man neue Punkte P_i wählen oder den Parameter B vergrößern und damit e und e' weiter faktorisieren.

Eine entscheidende Stelle dieses Algorithmus ist die Wahl der Konstanten B. Atkin schlägt in [At91] vor, den Parameter B in der Größenordnung 10^7 zu wählen. Weiterhin gibt er an der gleichen Stelle noch einen Hinweis, wie man unter Verwendung einer zweiten Stufe in dem ECM-Algorithmus den Parameter B bis zur Größenordnung 10^{12} vergrößern kann.

Kapitel 8

Bestimmung des Typs einer elliptischen Kurve

Wir haben bei der Analyse des Shanks Algorithmus häufig Informationen über den Typ der Punktgruppe einer elliptischen Kurve E über einem Primkörper $\mathbb{F}_p$ benutzt. Nach dem Satz von Rück 2.2 wissen wir, daß

$$E(\mathbb{F}_p) \quad \cong \quad \mathbb{Z}/n\mathbb{Z} \oplus \mathbb{Z}/m\mathbb{Z}$$

mit $n, m \in \mathbb{N},\ n \mid m,\ n \mid p-1$ ist. In diesem Kapitel werden wir einen Algorithmus vorstellen, um den Typ der Punktgruppe einer elliptischen Kurve wirklich zu bestimmen. Dieser Algorithmus setzt voraus, daß die Gruppenordnung $\#E(\mathbb{F}_p)$ bekannt ist. Für eine beliebige elliptische Kurve müssen wir daher zuerst mit Hilfe des Shanks Algorithmus 2.1 die Ordnung der Punktgruppe bestimmen und können dann mit dem nun vorgestellten Algorithmus den Typ der Punktgruppe berechnen.

8.1 Die Idee

Sei im folgenden eine feste, über dem Primkörper $\mathbb{F}_p$ definierte elliptische Kurve $E : y^2 = x^3 + ax + b$ vorgegeben. Wir geben zuerst die theoretische Grundlage der Berechnung des Typs einer Gruppe an.

Satz 8.1 (Elementarteilersatz)
Sei $\mathcal{N}$ Untermodul des Moduls $\mathcal{M}$. Dann gibt es eine Basis $(u_1, \ldots, u_n)$ von $\mathcal{M}$ und eine Basis $(v_1, \ldots, v_m)$ von $\mathcal{N}$, so daß

$$v_i = u_i \cdot \epsilon_i \qquad \textit{und} \qquad \epsilon_i \mid \epsilon_{i+1} \quad (i = 1, \ldots, m).$$

Weiterhin gilt dann für die Faktorgruppe

$$\mathcal{M}/\mathcal{N} \quad \cong \quad \mathbb{Z}/\epsilon_1\mathbb{Z} \oplus \ldots \oplus \mathbb{Z}/\epsilon_m\mathbb{Z}.$$

Beweis: [Hun74, Seite 343] ■

Bemerkung 8.1 *Der Basiswechsel zwischen den Moduln $\mathcal{M}$ und $\mathcal{N}$ wird häufig auch in Form einer Matrix beschrieben. Dann liefert dieser Satz, daß es eine Matrix des Basiswechsels der Form*

$$\begin{pmatrix} \epsilon_1 & & 0 \\ & \ddots & \\ 0 & & \epsilon_m \end{pmatrix}$$

mit $\epsilon_i \mid \epsilon_{i+1}$ gibt. Man nennt diese Form der Basistransformationsmatrix auch **Smith-Normalform***. Die Smith-Normalform einer Matrix kann durch elementare Spalten- und Zeilenoperationen aus dieser Matrix bestimmt werden. Die Vorgehensweise zur Bestimmung der Smith-Normalform leitet sich dabei direkt aus dem Beweis des Elementarteilersatzes ab, denn dieser Beweis ist konstruktiv. Eine genaue Beschreibung des Algorithmus findet man in [Co91, Seite 126].*

Wie wenden wir nun diesen Satz zur Bestimmung des Typs einer elliptischen Kurve E an?

Seien dazu zwei Punkte $P, Q \in E(\mathbb{F}_p)$ gegeben. Dann definieren wir mit Hilfe dieser zwei Punkte die Menge

$$\Gamma \quad := \quad \left\{(i,j) \in \mathbb{Z}^2 \,;\, i \cdot P + j \cdot Q = \mathcal{O}\right\}. \tag{8.1}$$

Dann gilt

Lemma 8.1 Γ *ist ein Gitter vom Rang 2.*

Beweis: Die Behauptung, daß Γ ein Gitter ist, folgt direkt aus der Definition eines Gitters und der Tatsache, daß $E(\mathbb{F}_p)$ ein $\mathbb{Z}$-Modul ist. Zum Beweis der zweiten Behauptung beachte man, daß die zwei $\mathbb{Z}$-linear unabhängigen Vektoren $(\text{ord}\,(P), 0)$ und $(0, \text{ord}\,(Q))$ Elemente von Γ sind. Daher muß der Rang von Γ mindestens zwei sein. Andererseits ist aber $\Gamma \subseteq \mathbb{Z}^2$ und somit ist der Rang von Γ auch höchstens zwei. ■

Wir versuchen nun, den Satz 8.1 auf diese Situation anzuwenden. Dazu wählen wir $\mathcal{N} = \Gamma$ und $\mathcal{M} = \mathbb{Z}^2$. Um die Verbindung zu den elliptischen Kurven herzustellen, benutzen wir folgende Proposition.

Proposition 8.1

$$\langle P, Q\rangle \quad \cong \quad \mathbb{Z}^2\big/\Gamma.$$

Beweis: Zum Beweis definieren wir folgende Abbildung

$$\begin{array}{rrcl} \varphi: & \langle P, Q\rangle & \longrightarrow & \mathbb{Z}^2\big/\Gamma \\ & i_P \cdot P + i_Q \cdot Q & \longmapsto & (i_P, i_Q). \end{array}$$

Wir beweisen nun, daß φ ein Isomorphismus ist. Daraus folgt dann direkt die Behauptung der Proposition.

- Wohldefiniertheit:
  Seien $P' = i'_P \cdot P + i'_Q \cdot Q$ und $P'' = i''_P \cdot P + i''_Q \cdot Q$ Punkte aus $\langle P, Q\rangle$. Dann folgt aus $P' = P''$ die Gleichung $P' - P'' = \mathcal{O}$ und damit

  $$(i'_P - i''_P) \cdot P + (i'_Q - i''_Q) \cdot Q = \mathcal{O} \quad \text{und so} \quad (i'_P - i''_P, i'_Q - i''_Q) \in \Gamma.$$

  Also gilt

  $$\varphi(P'') = (i''_P, i''_Q) = (i''_P, i''_Q) + (i'_P - i''_P, i'_Q - i''_Q) = (i'_P, i'_Q) = \varphi(P')$$

  und die Abbildung φ ist wohldefiniert.

- Linearität:
  Die Linearität von φ läßt sich leicht auf ähnliche Weise nachrechnen.

- Injektivität:
  Gelte die Gleichheit $(i', j') = (i'', j'')$ in $\mathbb{Z}^2\big/\Gamma$. Dann ist $(i'', j'') = (i', j') + (k, l)$ mit $(k, l) \in \Gamma$. Damit gilt für die Urbilder

  $$P' = \varphi^{-1}((i', j')) \qquad \text{und} \qquad P'' = \varphi^{-1}((i'', j'')) = \varphi^{-1}((i' + k, j' + l)),$$

  daß sie gleich sind, denn

  $$P'' = (i' + k) \cdot P + (j' + l) \cdot Q = i' \cdot P + j' \cdot Q + (k \cdot P + l \cdot Q) = P'.$$

  Dabei beachte man, daß $(k, l) \in \Gamma$ und damit $k \cdot P + l \cdot Q = \mathcal{O}$ ist.

- Surjektivität:
  Die Surjektivität der Abbildung φ ist nach Konstruktion von φ trivial, denn jedes Element $(i, j) \in \mathbb{Z}^2\big/\Gamma$ besitzt das Urbild $i \cdot P + j \cdot Q \in \langle P, Q\rangle$. ■

Damit können wir den Typ der elliptischen Kurve E bestimmen, wenn wir zwei Punkte P und Q kennen, die die ganze Punktgruppe erzeugen. Dann ist nämlich $\langle P, Q\rangle = E(\mathbb{F}_p)$ und demnach $E(\mathbb{F}_p) \cong \mathbb{Z}^2 \big/ \Gamma$. Im folgenden nehmen wir nun an, daß wir so zwei Punkte P und Q kennen. Dann stehen wir vor dem Problem, eine Matrix des Basiswechsels zwischen zwei Basen von $\mathbb{Z}^2$ und Γ zu bestimmen. Kennen wir eine solche Matrix, so können wir leicht mit Hilfe von Satz 8.1 und Proposition 8.1 den Typ der Punktgruppe $E(\mathbb{F}_p)$ bestimmen.

Zur Lösung dieses Problems setzen wir eine Matrix des Basiswechsels allgemein an als

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}.$$

Dann kann jedes Element $(i,j) \in \Gamma$ mit Hilfe dieser Matrix A und einem Element $x = (x_1, x_2) \in \mathbb{Z}^2$ dargestellt werden als

$$(i,j) = (x_1, x_2) \cdot A. \tag{8.2}$$

Außerdem wird jedes Basiselement von $\mathcal{M}$ in ein Basiselement von $\mathcal{N}$ abgebildet. Als Basis von $\mathbb{Z}^2$ wählen wir naürlich die Standardbasis $(1,0)$ und $(0,1)$. Ein Element der Basis von Γ ist sicherlich gegeben durch $(\mathrm{ord}\,(P), 0)$.

Wir wollen nun A so bestimmen, daß das Basiselement $(1,0) \in \mathbb{Z}^2$ auf das Basiselement $(\mathrm{ord}\,(P), 0) \in \Gamma$ abgebildet wird. Dann erhalten wir durch Einsetzen in (8.2)

$$a_{11} = \mathrm{ord}\,(P) \qquad \text{und} \qquad a_{12} = 0.$$

Ein weiteres Element des Gitters Γ ist der Vektor $(0, \mathrm{ord}(Q))$. Somit gibt es einen Vektor $(x_1, x_2) \in \mathbb{Z}^2$ mit $(0, \mathrm{ord}(Q)) = (x_1, x_2) \cdot A$, woraus wir durch Einsetzen die Bedingungen

$$x_1 \cdot \mathrm{ord}\,(P) + x_2 \cdot a_{21} = 0 \qquad \text{und} \qquad x_2 \cdot a_{22} = \mathrm{ord}\,(Q). \tag{8.3}$$

erhalten. Also muß der Matrixeintrag a_{22} ein Teiler der Ordnung von Q sein. Weiterhin gilt, daß auch der Vektor $(0,1) \cdot A = (a_{21}, a_{22})$ ein Element des Gitters Γ sein muß. Nun setzen wir alle diese Bedingungen zusammen.

Wir wissen, daß a_{22} ein Teiler der Ordnung von Q sein muß. Demnach müssen wir a_{22} als den minimalen Teiler der Ordnung von Q wählen, bei dem alle anderen Bedingungen ebenfalls erfüllt sind. Die Minimalität wird dadurch verursacht, daß der Vektor $(a_{21}, a_{22}) \in \Gamma$ ein Basiselement von Γ sein muß.

Sei nun $\mathrm{ord}\,(Q) = d \cdot d'$ und $a_{22} = d$. Dann folgt aus dem zweiten Teil von Bedingung (8.3) $x_2 = d'$ und damit muß für ein $x_1 \in \mathbb{Z}$

$$x_1 \cdot \mathrm{ord}\,(P) + d' \cdot a_{21} = 0$$

gelten. Durch Umformen erhalten wir hieraus

$$a_{21} = -x_1 \cdot \frac{\mathrm{ord}\,(P)}{d'}.$$

Da der Vektor (a_{21}, a_{22}) ein Element von Γ sein muß, muß nach der Definition von Γ gelten

$$\left(-x_1 \cdot \frac{\mathrm{ord}\,(P)}{d'}\right) \cdot P + d \cdot Q = \mathcal{O}.$$

Sei jetzt $g_1 = \mathrm{ggt}(\mathrm{ord}\,(P), d')$ und $d = g_1 \cdot g_2$. Dann müssen wir also überprüfen, ob es mit $P' := \frac{\mathrm{ord}\,(P)}{g_1} \cdot P$ eine Zahl $x_1 \in \mathbb{Z}$ gibt, so daß

$$\frac{x_1}{g_2} \cdot P' = d \cdot Q. \tag{8.4}$$

Dabei können wir o.B.d.A. annehmen, daß $0 \leq x_1 \cdot \frac{\mathrm{ord}\,(P)}{d'} < \mathrm{ord}(P)$ und damit $0 \leq x_1 < d'$ gilt. Da $\frac{x_1}{g_2} \in \mathbb{Z}$ sein muß, gilt $x_1 = i \cdot g_2$ mit $0 \leq i < g_1$.

Dieser Test ist wiederum ein typisches Anwendungsgebiet für einen Babystep-Giantstep Algorithmus. Dazu wählen wir die Anzahl der Baby- und Giantsteps als $\lceil \sqrt{g_1 - 1}\, \rceil$ und überprüfen damit, ob es ein $i \in \{0, \ldots, g_1 - 1\}$ gibt, für das $i \cdot P' = d \cdot Q$ ist. Verläuft dieser Test negativ, so war die Wahl von a_{22} falsch und wir müssen den Test mit einem anderen Wert für a_{22} wiederholen. Kennen wir andernfalls den Wert von i (und damit von $x_1 = i \cdot g_2$) und ist d minimal, so haben wir die Basistransformationsmatrix bestimmt als

$$A = \begin{pmatrix} \mathrm{ord}\,(P) & 0 \\ -x_1 \cdot \frac{\mathrm{ord}\,(P)}{d'} & d \end{pmatrix}.$$

Der Algorithmus besitzt noch eine problematische Stelle. Wir haben angenommen, daß wir zwei Punkte P und Q kennen, die die ganze Punktgruppe $E(\mathbb{F}_p)$ erzeugen. In der Realität ist dies allerdings selten der Fall. Darum wählen wir die zwei Punkte P und Q zufällig und hoffen, daß wir tatsächlich zwei Punkte gefunden haben, die die ganze Punktgruppe erzeugen. Wenn die zwei gewählten Punkte P und Q diese Eigenschaft jedoch nicht besitzen, so kann man dies nach Bestimmung der Smith-Normalform der Basistransformationsmatrix feststellen. Wir wissen nämlich aus Proposition 8.1 und Satz 8.1, daß $\#\langle P, Q\rangle = \epsilon_1 \cdot \epsilon_2$, wenn ϵ_i die Diagonalelemente der Basistransformationsmatrix in Smith-Normalform sind. Ist diese Ordnung echt kleiner als die Ordnung der Punktgruppe, so können P und Q nicht die ganze Punktgruppe erzeugen. Gilt jedoch die Gleichheit der beiden Ordnungen, so wissen wir, daß wir "gute" Punkte gewählt hatten.

Im Algorithmus benötigen wir häufig die Ordnung der beiden Punkte P und Q. Zur effizienten Bestimmung dieser Punktordnungen benötigen wir die Ordnung der Punktgruppe. Dazu berechnen wir einfach den kleinsten Teiler k der Gruppenordnung, für den $k \cdot P = \mathcal{O}$ ist. Nach den Bemerkungen aus dem zweiten Kapitel ist dieser Wert dann genau die Ordnung des Punktes P.

Falls wir weiterhin einen Punkt gewählt haben, dessen Ordnung schon gleich der Gruppenordnung ist, so sind wir fertig. Denn dann ist die Punktgruppe zyklisch und wir haben gerade einen Erzeuger der Punktgruppe gewählt. In diesem Fall können wir nach

Berechnung der Punktordnung das Ergebnis "Kurve zyklisch" ausgeben und den Algorithmus beenden, ohne eine Basistransformationsmatrix bestimmen zu müssen. Deshalb werden wir nach jeder Wahl eines zufälligen Punktes überprüfen, ob dieser Punkt schon die ganze Punktgruppe erzeugt.

8.2 Der Algorithmus

In diesem Abschnitt werden wir den oben beschriebenen Algorithmus genau formulieren. Die Korrektheit des Algorithmus folgt dann direkt aus den Bemerkungen des vorherigen Abschnitts.

Algorithmus 8.1 (Bestimmung des Typs)

Eingabe: *elliptische Kurve E über Primkörper $\mathbb{F}_p$.*
Ausgabe: *Typ der Punktgruppe $E(\mathbb{F}_p)$.*

1. *Bestimme mit Hilfe des Shanks Algorithmus 2.1 die Ordnung $\#E(\mathbb{F}_p)$ der Punktgruppe $E(\mathbb{F}_p)$.*

2. *Wähle zufällig Punkte P und Q aus $E(\mathbb{F}_p)$ und bestimme ihre Ordnungen gemäß*

$$ord(P) = \min\{d\ ;\ d|\#E(\mathbb{F}_p)\ ,\ d \cdot P = \mathcal{O}\},$$

$$ord(Q) = \min\{d\ ;\ d|\#E(\mathbb{F}_p)\ ,\ d \cdot Q = \mathcal{O}\}.$$

   Sei o.B.d.A. $ord(P) \geq ord(Q)$.

3. *Falls $ord(P) = \#E(\mathbb{F}_p)$, so terminiere mit Ergebnis "elliptische Kurve zyklisch".*

4. *Für alle Teiler d von $ord(Q) = d \cdot d'$ in aufsteigender Reihenfolge tue*

   (a) *Berechne $g_1 = ggt(ord(P), d')$ und $g_2 = \frac{d'}{g_1}$.*

   (b) *Setze $P' = \frac{\mathrm{ord}(P)}{g_1} \cdot P$.*

   (c) *Teste, ob $d \cdot Q \in \langle P' \rangle$ durch*

      i. *Setze $k' = \lceil \sqrt{g_1} \rceil$.*

      ii. *Für i=0 bis $k' - 1$:*
      Berechne $i \cdot P'$.
      Falls $i \cdot P' = d \cdot Q$, so setze $x_1 = i \cdot g_2$ und gehe zu Schritt 4(d), ansonsten speichere $(i \cdot P', i)$ in einer Tabelle T.

      iii. *Setze $Q' = k' \cdot P'$.*

iv. *Für $j = 1$ bis k':*
Überprüfe, ob $j \cdot Q' - d \cdot Q = i \cdot P'$ für einen Punkt $i \cdot P'$ in der Tabelle T.
Falls ja, so setze $x_1 = (j \cdot k' - i) \cdot g_2$ und gehe zu Schritt 4(d).

(d) *Falls in Schritt 4(c) kein x_1 berechnet wurde, fahre mit Schleife fort, ansonsten gehe zu Schritt 5.*

5. *Berechne die Smith-Normalform der Matrix*

$$A = \begin{pmatrix} ord(P) & 0 \\ -x_1 \cdot \frac{\mathrm{ord}(P)}{d'} & d \end{pmatrix}.$$

6. *Sei $e = (\epsilon_1, \epsilon_2)$ der Vektor der Diagonalelemente der Smith-Normalform von A. Falls $\#E(\mathbb{F}_p) = \epsilon_1 \cdot \epsilon_2$, so terminiere mit Ausgabe*

$$E(\mathbb{F}_p) \quad \cong \quad \mathbb{Z}/\epsilon_1 \oplus \mathbb{Z}/\epsilon_2,$$

ansonsten gehe zu Schritt 2.

Wie wir schon im vorherigen Abschnitt bemerkt haben, haben wir in Schritt 4(c) wiederum eine Babystep-Giantstep Strategie angewandt, um zu testen, ob $d \cdot Q \in \langle P' \rangle$ ist. Dabei gehen wir analog zum zweiten Teil des Shanks Algorithmus vor, wo wir denselben Test machen müssen. Die Korrektheit von Schritt 4 beweist man analog zum entsprechenden Teil des Shanks Algorithmus.

8.3 Erwartete Laufzeit des Typ-Algorithmus

In diesem Abschnitt wollen wir die erwartete Laufzeit des gerade vorgestellten Algorithmus 8.1 zur Bestimmung des Typs der Punktgruppe einer elliptischen Kurve berechnen. Dabei gehen wir ähnlich vor wie bei den Laufzeitbestimmungen in Kapitel 3. Als erstes müssen wir demnach die Wahrscheinlichkeit, daß zwei zufällig gewählte Punkte die ganze Punktgruppe erzeugen, abschätzen. Mit dieser Information können wir dann die erwartete Laufzeit berechnen. Dazu formulieren wir folgenden Satz:

Satz 8.2 *Sei E eine elliptische Kurve über $\mathbb{F}_p$. Dann gibt es eine Konstante c, so daß die Wahrscheinlichkeit, daß zwei zufällige Punkte $P', Q' \in E(\mathbb{F}_p)$ die ganze Punktgruppe $E(\mathbb{F}_p)$ erzeugen, mindestens*

$$\frac{c}{\log\log(p)^2}$$

beträgt.

Beweis: Nach dem Satz von Rück 2.2 gibt es zwei Punkte P und Q, so daß

$$E(\mathbb{F}_p) \;=\; \langle P\rangle \oplus \langle Q\rangle$$

ist. Dabei ist o.B.d.A. $n = \#\langle Q\rangle$ ein Teiler von $m = \#\langle P\rangle$.

Seien P' und $Q' \in E(\mathbb{F}_p)$ zwei zufällige Punkte der Punktgruppe $E(\mathbb{F}_p)$. Dann werden diese beiden Punkte sicherlich von P und Q erzeugt, d.h. es gibt eine Matrix

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad \text{mit} \qquad A \cdot \begin{pmatrix} P \\ Q \end{pmatrix} = \begin{pmatrix} P' \\ Q' \end{pmatrix}. \tag{8.5}$$

Dabei können wir ohne Beschränkung der Allgemeinheit annehmen, daß Matrixeinträge die Bedingungen $0 \leq a_{11}, a_{21} < m$ und $0 \leq a_{12}, a_{22} < n$ erfüllen. Erfüllen die Matrixeinträge aber diese Bedingung, dann ist die Menge der Matrizen dieser Gestalt zu der Menge der Paare von Punkten (P', Q') bijektiv.

Seien nun P' und Q' zwei zufällige Punkte, die die ganze Punktgruppe $E(\mathbb{F}_p)$ erzeugen. Da wir jeden Punkt $H \in E(\mathbb{F}_p)$ als Linearkombination von P und Q darstellen können, muß es also für alle $0 \leq a_P < m$ und alle $0 \leq a_Q < n$ Zahlen $x, y \in \mathbb{Z}$ geben, so daß

$$H \quad = \quad x \cdot P' + y \cdot Q' \quad = \quad a_P \cdot P + a_Q \cdot Q$$

ist. Dabei sollte man beachten, daß o.B.d.A. $0 \leq x, y < m$ angenommen werden kann, da die Ordnung aller Punkte wegen der Aussage über den Typ der Punktgruppe maximal m sein kann. Unter Verwendung von Gleichung (8.5) können wir die linke Seite dieser Gleichung auch formulieren als

$$(x, y) \cdot \begin{pmatrix} P' \\ Q' \end{pmatrix} = (x, y) \cdot A \cdot \begin{pmatrix} P \\ Q \end{pmatrix}.$$

Beachten wir die Ordnungen der Punkte P und Q, so erhalten wir hieraus

$$x \cdot a_{11} + y \cdot a_{21} \equiv a_P \bmod m. \tag{8.6}$$

$$x \cdot a_{12} + y \cdot a_{22} \equiv a_Q \bmod n. \tag{8.7}$$

Wenn wir nun annehmen, daß ein Wert für $y \bmod n$ schon bestimmt ist, dann erhalten wir durch Umformung von Gleichung (8.6)

$$a_{11} \cdot x \equiv a_P - (y + k \cdot n) \cdot a_{21} \bmod m \qquad \text{für ein } k \in \mathbb{Z}.$$

Damit dies für alle Werte a_P sicherlich lösbar ist, muß a_{11} in dem Ring $\mathbb{Z}\big/m\mathbb{Z}$ invertierbar sein, d.h. es muß $\mathrm{ggt}(a_{11}, m) = 1$ gelten. Dies liefert eine erste Bedingung an einen der Matrixeinträge.

Da n ein Teiler von m ist, gilt Gleichung (8.6) auch modulo n. Dann können wir die Gleichungen (8.6) und (8.7) kombinieren zu

$$(a_{11} \cdot a_{22} - a_{12} \cdot a_{21}) \cdot y \equiv a_Q \cdot a_{11} - a_P \cdot a_{12} \bmod n.$$

Da $det(A) = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}$ ist, können wir diese Gleichung sicherlich nach y auflösen, wenn $det(A)$ modulo n invertierbar ist. Ist diese Bedingung sowie die obige Bedingung an den Eintrag a_{11} erfüllt, so lassen sich die Gleichungen (8.6) und (8.7) für alle möglichen Werte von a_P und a_Q lösen.

Seien daher nun $0 \leq a_{12} < n$ und $0 \leq a_{11}, a_{21} < m$ mit $\text{ggt}(a_{11}, m) = 1$ fest, aber beliebig vorgegeben. Dann muß

$$det(A) = a_{11} \cdot a_{22} - a_{12} \cdot a_{21} \equiv k \bmod n$$

sein, wobei k eine positive Zahl kleiner als n mit $\text{ggt}(k, n) = 1$ ist. Da wegen $\text{ggt}(a_{11}, m) = 1$ der Matrixeintrag a_{11} auch modulo n invertierbar ist, erhalten wir damit

$$a_{22} \equiv (k - a_{12} \cdot a_{21}) \cdot a_{11}^{-1} \bmod n.$$

Betrachten wir die rechte Seite dieser Gleichung für zwei verschiedene geeignete Werte $k \not\equiv k' \bmod n$, so erhalten wir zwei modulo n verschiedene Ergebnisse. Daher gibt es $\varphi(n)$ viele Möglichkeiten, einen "guten" Matrixeintrag a_{22} zu wählen.

Zählen wir nun die Anzahl der möglichen Matrizen dieser besonderen Gestalt, so erhalten wir das Ergebnis $m \cdot n \cdot \varphi(m) \cdot \varphi(n)$. Damit gibt es also auch mindestens $m \cdot n \cdot \varphi(m) \cdot \varphi(n)$ viele Paare von Punkten, die die ganze Punktgruppe erzeugen. Insgesamt gibt es aber offensichtlich $\#E(\mathbb{F}_p)^2 = m^2 \cdot n^2$ viele Möglichkeiten, Paare aus zwei zufälligen Punkten zu bilden. Somit erhalten wir für die Wahrscheinlichkeit h, daß zwei zufällige Punkte die ganze Punktgruppe erzeugen, mindestens

$$h \quad \geq \quad \frac{m \cdot n \cdot \varphi(m) \cdot \varphi(n)}{m^2 \cdot n^2} = \frac{\varphi(m)}{m} \cdot \frac{\varphi(n)}{n}.$$

In Satz 3.3 haben wir schon gesehen, daß

$$\frac{\varphi(x)}{x} \geq \frac{c'}{\log\log(x)}$$

für eine Konstante c' ist. Damit erhalten wir

$$h \quad \geq \quad \frac{c'}{\log\log(m)} \cdot \frac{c'}{\log\log(n)}.$$

Da wegen der Typaussage die Abschätzungen $n \leq c_1 \cdot \sqrt{p}$ und $m \leq c_2 \cdot p$ mit Konstanten c_1, c_2 gelten, erhalten wir hieraus das gewünschte Resultat

$$h \quad \geq \quad \frac{c}{(\log\log(p))^2}.$$

■

Damit können wir nun auf die gleiche Art und Weise wie in Satz 3.5 eine erwartete Laufzeit unseres Algorithmus zur Bestimmung des Typs der Punktgruppe einer elliptischen Kurve über einem Primkörper bestimmen. Dies erfolgt im nächsten Satz.

Satz 8.3 *Die erwartete Laufzeit von Algorithmus 8.1 beträgt*

$$O\left(\sqrt{p} \cdot \log(p)^3 \cdot (\log\log(p))^4\right).$$

Beweis: Wir wollen zuerst nur die erwartete Laufzeit der Schritte 2 bis 6 in Algorithmus 8.1 bestimmen. Dazu setzen wir die deterministische Laufzeit eines Durchlaufs der Schritte 3 bis 6 sowie die Bestimmung der Punktordnungen in Schritt 2 an als T_T. Wir werden später den genauen Wert von T_T bestimmen. Nun bestimmen wir analog zum Beweis von Satz 3.5 zuerst die erwartete Laufzeit, wenn wir genau n Schleifendurchläufe benötigen. Schleifen entstehen hier in Schritt 1-2 des Punktsuche-Algorithmus 3.3, in Schritt 2 des RESSOL-Algorithmus 3.5 und in Schritt 2-7 des Typbestimmungsalgorithmus 8.1. Sei nun i die Anzahl der Schleifendurchläufe im Punktsuche-Algorithmus, j die Anzahl der Schleifendurchläufe im RESSOL-Algorithmus und k die Anzahl der Durchläufe im Typbestimmungsalgorithmus. Da wir zwei Punkte zufällig wählen, gilt dann offensichtlich $i, j \geq 2$. Außerdem gilt natürlich weiterhin, daß die Anzahl der Durchläufe im Punktsuche- bzw. RESSOL-Algorithmus mindestens so groß ist wie die Anzahl der Durchläufe im Typbestimmungsalgorithmus, also $i, j \geq k$. Damit erhalten wir dann nach Lemma 3.4

$$E_n \leq \sum_{\substack{i+j+k=n;\\ i,j\geq \max\{2,k\}}} \left(\frac{2}{3}\right)^{i+j-2} \cdot \left(1 - \frac{c}{\log\log(p)^2}\right)^{k-1} \cdot \left((i+j)\cdot\log(p)^2 + k\cdot(T_T + \log(p)^4)\right).$$

Durch analoge Rechnungen zum Beweis von Satz 3.5, den Grenzübergang $n \to \infty$ und Benutzung von Lemma 3.5 erhalten wir mit Konstanten c' und c'' hieraus

$$E_\infty \leq c' \cdot \log(p)^2 + c'' \cdot (\log\log(p))^4 \cdot (T_T + \log(p)^4).$$

Um die erwartete Laufzeit zu bestimmen, bleibt uns jetzt noch die Bestimmung von T_T:

- Die Bestimmung der Punktordnungen in Schritt 2 kann in Zeit $O(\log(p)^4)$ erfolgen, denn die Gruppenordnung $\#E(\mathbb{F}_p)$ besitzt maximal $O(\log(p))$ viele Teiler.
- Schritt 3,4 und 7 benötigen offensichtlich Zeit $O(1)$.
- In Schritt 4(a)-(c) führen wir wieder einen Babystep-Giantstep Test durch. Wir haben in Satz 3.5 schon gesehen, daß die Laufzeit dieses Teils von der Anzahl der Baby- bzw. Giantsteps abhängt. In diesem Fall ist diese Anzahl maximal die Quadratwurzel aus der Punktordnung von Q, denn $g_1 \leq d' \leq \mathrm{ord}\,(Q)$. Da die Ordnung von Q maximal von der Größenordnung $O(p)$ ist, ist die Anzahl der Babysteps von der Größenordnung $O(\sqrt{p})$ und wir erhalten als Laufzeit eines Durchlaufs der Schritte 4(a)-(c) die Laufzeit $O(\sqrt{p} \cdot \log(p)^2)$. Dies folgt aus der Tatsache, daß wir innerhalb der Baby- bzw. Giantsteps im großen und ganzen nur Additionen von Punkten der elliptischen Kurve durchführen müssen. Dieser Babystep-Giantstep

Test in Schritt 4 wird jedoch maximal für alle Teiler der Ordnung von Q durchlaufen, also maximal $O(\log(p))$-mal. Damit besitzt der komplette Schritt 4 die Laufzeit $O(\sqrt{p} \cdot \log(p)^3)$.

- Die Berechnung der Smith-Normalform einer 2×2-Matrix kann in polynomieller Zeit erfolgen. Dies folgt direkt aus der Beschreibung des Algorithmus in [Co91, Seite 126].

Damit erhalten wir als Laufzeit eines Durchlaufs der Schritte 2 bis 6 die Zeit $T_T = O(\sqrt{p} \cdot \log(p)^3)$. Somit erhalten wir als erwartete Laufzeit der Schritte 2 bis 6 des Typbestimmungsalgorithmus 8.1 die Laufzeit

$$O(\sqrt{p} \cdot \log(p)^3 \cdot (\log\log(p))^4).$$

Zusätzlich bestimmen wir vorher noch mit Hilfe des Shanks Algorithmus 2.1 die Gruppenordnung $\#E(\mathbb{F}_p)$. Nach Satz 3.8 besitzt der Shanks Algorithmus die erwartete Laufzeit $O(\sqrt[4]{p} \cdot \log(p)^3)$. Damit erhalten wir als erwartete Laufzeit des kompletten Typ-Algorithmus 8.1

$$O\left(\sqrt{p} \cdot \log(p)^3 \cdot (\log\log(p))^4 + \sqrt[4]{p} \cdot \log(p)^3\right) = O\left(\sqrt{p} \cdot \log(p)^3 \cdot (\log\log(p))^4\right).$$

■

8.4 Platzbedarf des Typ-Algorithmus

Analog zum Vorgehen in Kapitel 3 wollen wir jetzt noch die Platzkomplexität unseres Algorithmus zur Bestimmung des Typs einer elliptischen Kurve bestimmen. Diese Komplexität wird hauptsächlich durch die abzuspeichernden Tabellen bestimmt. Näheres ist aus folgendem Satz ersichtlich.

Satz 8.4 *Der Typ-Algorithmus 8.1 benötigt zur Berechnung des Typs einer elliptischen Kurve über dem Primkörper $\mathbb{F}_p$ $O\left(\sqrt{p} \cdot \log(p)\right)$ viele Bits.*

Beweis: Nach Satz 3.10 wissen wir, daß der Shanks Algorithmus in Schritt 1 maximal $O\left(\sqrt[4]{p} \cdot \log(p)\right)$ viele Bits benötigt. Offensichtlich wird der Platzbedarf im restlichen Teil von Algorithmus 8.1 bestimmt durch den Platz, den wir zum Abspeichern der Tabellen in Schritt 4 benötigen. Wir haben in Satz 8.3 schon gesehen, daß diese Tabelle $O(\sqrt{p})$ viele Punkte enthalten. Jeder Punkt benötigt aber nur Platz $O(\log(p))$. Da wir nach jedem Durchlauf in Schritt 4 die alte Tabelle löschen können, erhalten wir als Platzkomplexität der Schritte 2-7 des Typbestimmungsalgorithmus 8.1 $O(\sqrt{p} \cdot \log(p))$. Dies dominiert eindeutig die Platzkomplexität des Shanks Algorithmus, so daß der Satz damit bewiesen ist. ■

Kapitel 9

Praktische Ergebnisse

9.1 Ergebnisse des Shanks Algorithmus

In diesem Anhang beschreiben wir einige praktische Beispiele, die wir mit unserer Implementierung des Shanks Algorithmus 2.1 berechnet haben. Dabei wurden alle Berechnungen auf einer SUN SPARCSTATION 2 mit 32 MByte Hauptspeicher durchgeführt. Die maximale Tabellengröße war in allen Beispielen auf 500000 Tabelleneinträge festgelegt.

Zuerst nun einige Laufzeitresultate, die wir mit unserer Implementierung des einfachen Shanks Algorithmus erhielten:

Größe der Charakteristik	benötigte Zeit
$\approx 10^{11}$	2 sec
$\approx 5 \cdot 10^{11}$	3 sec
$\approx 10^{12}$	3 sec
$\approx 5 \cdot 10^{12}$	6 sec
$\approx 10^{13}$	10 sec
$\approx 5 \cdot 10^{13}$	10 sec
$\approx 10^{14}$	19 sec
$\approx 5 \cdot 10^{14}$	19 sec
$\approx 10^{15}$	32 sec
$\approx 5 \cdot 10^{15}$	45 sec
$\approx 10^{16}$	55 sec
$\approx 5 \cdot 10^{16}$	1 min 26 sec
$\approx 10^{17}$	1 min 41 sec
$\approx 5 \cdot 10^{17}$	2 min 43 sec

Größe der Charakteristik	benötigte Zeit
$\approx 10^{18}$ $\approx 5 \cdot 10^{18}$	3 min 38 sec 3 min 41 sec
$\approx 10^{19}$ $\approx 5 \cdot 10^{19}$	6 min 12 sec 8 min 41 sec
$\approx 10^{20}$ $\approx 5 \cdot 10^{20}$	9 min 22 sec 14 min 38 sec
$\approx 10^{21}$ $\approx 5 \cdot 10^{21}$	18 min 46 sec 28 min 21 sec
$\approx 10^{22}$ $\approx 5 \cdot 10^{22}$	45 min 17 sec 48 min 26 sec
$\approx 10^{23}$ $\approx 5 \cdot 10^{23}$	1 h 23 min 6 sec 1 h 17 min 3 sec
$\approx 10^{24}$ $\approx 5 \cdot 10^{24}$	2 h 44 min 34 sec 5 h 59 min 59 sec
$\approx 10^{25}$ $\approx 5 \cdot 10^{25}$	8 h 45 min 12 sec 19 h 26 min 54 sec

Es fällt in dieser Tabelle auf, daß der einfache Shanks Algorithmus für kleine Charakteristiken bis etwa 10^{18} extrem schnell ist. Bis zu dieser Charakteristik ist daher der Shanks Algorithmus eine gute Wahl zur Bestimmung der Gruppenordnung. Interessant ist weiterhin die Tatsache, wie sich diese Zeiten auf die einzelnen Prozeduren aufteilen. Dazu geben wir die folgende Tabelle an, in der der Zeitanteil der einzelnen Prozeduren an der Gesamtlaufzeit angegeben wird.

Charakteristik	Babysteps	Quicksort	Giantsteps
$\approx 10^{12}$	2 sec $\approx 66\%$	0 sec $\approx 2\%$	1 sec $\approx 31\%$
$\approx 10^{15}$	20 sec $\approx 63\%$	0 sec $\approx 2\%$	11 sec $\approx 35\%$
$\approx 10^{18}$	1 min 10 sec $\approx 70\%$	2 sec $\approx 3\%$	27 sec $\approx 27\%$
$\approx 10^{21}$	13 min 15 sec $\approx 70\%$	30 sec $\approx 3\%$	5 min $\approx 27\%$
$\approx 10^{22}$	27 min 12 sec $\approx 60\%$	1 min 2 sec $\approx 2\%$	17 min 1 sec $\approx 38\%$
$\approx 5 \cdot 10^{22}$	27 min 50 sec $\approx 57\%$	59 sec $\approx 2\%$	19 min 37 sec $\approx 41\%$

Charakteristik	Babysteps	Quicksort	Giantsteps
$\approx 10^{23}$	28 min 18 sec $\approx 34\%$	59 sec $\approx 1\%$	53 min 48 sec $\approx 65\%$
$\approx 10^{24}$	29 min 6 sec $\approx 18\%$	57 sec $\approx 1\%$	2 h 14 min 30 sec $\approx 81\%$
$\approx 10^{25}$	30 min 40 sec $\approx 6\%$	57 sec $\approx 0\%$	8 h 13 min 35 sec $\approx 94\%$

Es fällt in dieser Tabelle auf, daß ab Charakteristik $\approx 10^{23}$ die Laufzeit der Babysteps etwa gleich bleibt, dafür aber die Laufzeit der Giantsteps schnell anwächst. Dies wird dadurch verursacht, daß wir aus praktischen Gründen die Tabellengröße nach oben beschränkt haben. Etwa ab $p \approx 10^{23}$ wird diese Tabellengröße voll ausgenutzt. Dies erkennt man auch, wenn man die Tabellengröße für 10^{23} nach der in Kapitel 4 angegebenen Formel berechnet. Wie wir schon im Kapitel über unsere Implementierung erläutert haben, können wir für Primkörper größerer Charakteristik die Anzahl der Giantsteps dann nicht mehr optimal wählen. Dadurch steigt natürlich die Anzahl der Giantsteps und damit auch die dazu benötigte Zeit stark an. Weiterhin wird aus der Tabelle deutlich, daß ab etwa 10^{25} der normale Shanks Algorithmus immer schlechter praktisch anwendbar wird, da die Giantsteps immer mehr Zeit verschlingen, und daß damit eine Grenze für diesen Algorithmus erreicht ist.

9.2 Der Kombinations-Algorithmus

In diesem Abschnitt beschreiben wir Laufzeitresultate für unseren Kombinationsalgorithmus 7.1. Zur Ermittlung der Laufzeiten haben wir dabei dieselben elliptischen Kurven wie beim normalen Shanks Algorithmus benutzt. Auch alle anderen Bedingungen wie die Tabellengröße oder der Rechnertyp wurden nicht verändert.

In der folgenden Tabelle geben wir einen ersten Überblick über die Laufzeitresultate, die wir mit unserem Kombinationsalgorithmus 7.1 erreichten. Dabei unterscheiden wir die Fälle, welche Primzahlen wir im Subalgorithmus 5.1 benutzt haben. Im folgenden bedeutet K_i dabei immer, daß wir bei dieser Version alle Primzahlen von 2 bis einschließlich i im Subalgorithmus 5.1 verwendet haben und somit nach der Vorberechnung den Wert $\#E(\mathbb{F}_p) \bmod \left(\prod_{2\leq p\leq i} p\right)$ kennen.

Charakteristik	Laufzeit K_3	Laufzeit K_5	Laufzeit K_7
$\approx 10^{11}$	2 sec	10 sec	43 sec
$\approx 5 \cdot 10^{11}$	2 sec	11 sec	40 sec
$\approx 10^{12}$	3 sec	7 sec	36 sec
$\approx 5 \cdot 10^{12}$	4 sec	10 sec	44 sec
$\approx 10^{13}$	6 sec	6 sec	56 sec
$\approx 5 \cdot 10^{13}$	7 sec	7 sec	47 sec
$\approx 10^{14}$	11 sec	18 sec	40 sec
$\approx 5 \cdot 10^{14}$	13 sec	17 sec	1 min 3 sec
$\approx 10^{15}$	18 sec	22 sec	1 min 13 sec
$\approx 5 \cdot 10^{15}$	24 sec	19 sec	41 sec
$\approx 10^{16}$	29 sec	28 sec	37 sec
$\approx 5 \cdot 10^{16}$	45 sec	24 sec	1 min 8 sec
$\approx 10^{17}$	56 sec	36 sec	1 min 20 sec
$\approx 5 \cdot 10^{17}$	1 min 35 sec	50 sec	1 min 14 sec
$\approx 10^{18}$	1 min 59 sec	1 min 2 sec	1 min 12 sec
$\approx 5 \cdot 10^{18}$	2 min 19 sec	1 min	1 min 3 sec
$\approx 10^{19}$	3 min 21 sec	1 min 39 sec	1 min 15 sec
$\approx 5 \cdot 10^{19}$	4 min 43 sec	2 min 17 sec	1 min 41 sec
$\approx 10^{20}$	5 min 2 sec	2 min 22 sec	1 min 23 sec
$\approx 5 \cdot 10^{20}$	7 min 44 sec	3 min 43 sec	2 min 39 sec
$\approx 10^{21}$	10 min 14 sec	4 min	3 min 4 sec
$\approx 5 \cdot 10^{21}$	15 min 22 sec	7 min 9 sec	4 min 22 sec
$\approx 10^{22}$	24 min 24 sec	11 min 18 sec	5 min 56 sec
$\approx 5 \cdot 10^{22}$	28 min 51 sec	13 min 3 sec	5 min 21 sec
$\approx 10^{23}$	44 min 30 sec	20 min 18 sec	9 min 23 sec
$\approx 5 \cdot 10^{23}$	42 min 39 sec	22 min 32 sec	9 min 32 sec
$\approx 10^{24}$	1 h 15 min 9 sec	33 min 24 sec	15 min 41 sec
$\approx 5 \cdot 10^{24}$	1 h 40 min 44 sec	46 min 20 sec	18 min 16 sec
$\approx 10^{25}$	3 h 5 min 5 sec	1 h 0 min 16 sec	25 min 18 sec
$\approx 5 \cdot 10^{25}$	6 h 51 min 12 sec	1 h 45 min 58 sec	43 min 56 sec

Vergleichen wir diese Laufzeiten mit den Laufzeiten des normalen Shanks Algorithmus, so bemerken wir, daß sich die Verwendung des Subalgorithmus 5.1 bis zur Primzahl 3 erst ab Charakteristik $p \approx 10^{18}$ positiv auswirkt. Für kleinere Charakteristik dauert der Subalgorithmus 5.1 so lange, daß die durch diese Information bewirkte Laufzeitverbesserung des Babystep-Giantstep Teils dies nicht mehr ausgleicht. Also ist bis etwa zur Charakteristik $p \approx 10^{18}$ der einfache Shanks Algorithmus die bessere Wahl.

Ab dann jedoch macht sich die Information, die wir aus dem Subalgorithmus 5.1 erhalten, positiv an der Gesamtlaufzeit bemerkbar. Als weiteres fällt an diesen Zeiten auf, daß sich bei Vergrößerung der Charakteristik die Laufzeit des Kombinationsalgorithmus nicht unbedingt auch immer vergrößert. Dies wird dadurch verursacht, daß die Gesamtlaufzeit in diesen Fällen stark von der Laufzeit des Subalgorithmus 5.1 beeinflußt wird. Diese hängt aber davon ab, wieviele Schleifendurchläufe wir im zweiten Teil von Algorithmus 5.1 machen müssen, d.h. wie das Verhalten der Gruppenordnung modulo der Primzahl, die wir im Subalgorithmus verwenden, gerade ist. Daher geben wir nun in der nächsten Tabelle einen Überblick über den prozentualen Anteil des Subalgorithmus 5.1 an der Gesamtlaufzeit.

Charakteristik	K_3	K_5	K_7
$\approx 10^{12}$	41 %	85 %	98 %
$\approx 10^{15}$	9 %	64 %	96 %
$\approx 10^{18}$	1 %	16 %	73 %
$\approx 10^{21}$	0.3 %	2.5 %	44 %
$\approx 10^{22}$	0.1 %	2 %	30 %
$\approx 10^{23}$	0 %	1.5 %	20 %
$\approx 10^{24}$	0 %	1 %	11 %
$\approx 10^{25}$	0 %	0 %	7 %

Im folgenden bringen wir nun Laufzeitresultate für Primkörper größerer Charakteristik. Für diese Primkörper haben wir die Primzahlen bis 11 im Subalgorithmus 5.1 verwendet.

Charakteristik	K_7	K_{11}
$\approx 10^{25}$	25 min 18 sec	17 min 44 sec
$\approx 5 \cdot 10^{25}$	43 min 56 sec	27 min 38 sec
$\approx 10^{26}$	49 min 26 sec	32 min 24 sec
$\approx 5 \cdot 10^{26}$	53 min 47 sec	22 min 28 sec
$\approx 10^{27}$	1 h 23 min 37 sec	39 min 14 sec
$\approx 5 \cdot 10^{27}$	1 h 14 min 12 sec	38 min 41 sec
$\approx 10^{28}$	2 h 39 min 34 sec	54 min 38 sec
$\approx 5 \cdot 10^{28}$	6 h 35 min 38 sec	1 h 27 min 34 sec

Bevor wir noch einige praktische Resultate vorstellen, wollen wir auch noch die Laufzeit unseres bisherigen “Rekords” vorstellen. Dieser wurde erreicht, als wir im Subalgorithmus zusätzlich noch die Primzahl 13 zugelassen haben. Ein Versuch, zusätzlich noch die Primzahl 17 im Subalgorithmus 5.1 zu verwenden, scheiterte, da die Laufzeit der Vorberechnung dann überdimensional wuchs. Daher ist es in der Praxis nur möglich, alle Primzahlen bis 13 zur Vorberechnung zu verwenden.

Charakteristik	K_{11}	K_{13}
$\approx 10^{29}$	1 h 52 min 21 sec	1 h 14 min 24 sec
$\approx 5 \cdot 10^{29}$	3 h 35 min 57 sec	1 h 29 min 18 sec
$\approx 10^{30}$	/	1 h 43 min 58 sec
$\approx 5 \cdot 10^{30}$	/	2 h 0 min 10 sec
$\approx 10^{31}$	/	1 h 44 min 11 sec
$\approx 5 \cdot 10^{31}$	/	3 h 7 min 30 sec
$\approx 10^{32}$	/	3 h 3 min 4 sec
$\approx 5 \cdot 10^{32}$	/	3 h 23 min 5 sec

Zum Schluß dieses Abschnitts wollen wir auch noch einige wirklich berechnete Ergebnisse angeben. Dazu wählten wir die Kurve

$$E: \qquad y^2 \quad = \quad x^3 + x + 1$$

über den Primkörpern der Charakteristik p_i $(i = 1, \ldots, 10)$, wobei p_i die kleinste Primzahl größer als 10^{3i} ist. Dabei erhielten wir die folgenden Ergebnisse:

i	Charakteristik p_i	$\#E(\mathbb{F}_{p_i})$
1	1009	1034
2	1000003	1000727
3	1000000007	999953006
4	1000000000039	1000001795702
5	1000000000000037	999999998152255
6	1000000000000000003	999999999342281658
7	1000000000000000000117	999999999978613585736
8	1000000000000000000000007	999999999999288272844428
9	1000000000000000000000000103	1000000000000044679400742805
10	1000000000000000000000000000057	999999999999998088794205084600

9.3 Typberechnungen

Nun noch zum Abschluß einige Ergebnisse, die wir bei der Berechnungen von Typen von elliptischen Kurven erhalten haben. Ein von uns durchgeführter Test war der, daß wir zu fester Charakteristik p und damit festem Primkörper $\mathbb{F}_p$ für alle Paare $(a, b) \in \mathbb{F}_p^2$ getestet haben, ob dieses Paar zu einer elliptischen Kurve führt und wenn das der Fall war, welchen Typ die Punktgruppe dieser elliptischen Kurve besitzt. Diese Tests konnten natürlich nur für kleine Primzahlen durchgeführt werden. Dabei erhielten wir für die Primzahl 41 beispielsweise folgende Ergebnisse:

Primzahl:	41
Anzahl der möglichen Paare $(a, b) \in \mathbb{F}_{41}$:	1681
davon führen zu keiner elliptischen Kurve:	41
ellipt. Kurve mit zyklischer Punktgruppe:	1370 ($\approx$ 83.5%)
ellipt. Kurve vom Typ [x,2]:	230 ($\approx$ 14%)
ellipt. Kurve vom Typ [x,4]:	30 ($\approx$ 1.8%)
ellipt. Kurve vom Typ [x,5]:	10 ($\approx$ 0.6%)

Es wird an diesem Beispiel ganz offensichtlich, daß für ungefähr 97.5 % aller elliptischen Kurven über dem Primkörper $\mathbb{F}_{41}$ Punkte mit großer Ordnung existieren, so daß der Shanks Algorithmus schon im ersten Teil terminiert. Damit wurde also experimentell bestätigt, daß fast immer nur der erste Teil des Shanks Algorithmus benötigt wird. Derselbe Test wurde auch noch für weitere Primzahlen durchgeführt. Die Ergebnisse liefert die folgende Tabelle:

	Primzahl 71	Primzahl 89	Primzahl 131
Paare insgesamt	5041	7921	17161
keine elliptische Kurve	71	89	131
zyklische elliptische Kurve	4130 $\approx$ 83.1%	6555 $\approx$ 83.7%	14105 $\approx$ 82.8%
Typ [x,2]	805 $\approx$ 16.2%	1122 $\approx$ 14.3%	2795 $\approx$ 16.4%
Typ [x,4]	35 $\approx$ 0.7%	155 $\approx$ 1.9%	0
Typ [x,5]	0	0	130 $\approx$ 0.8%

	Primzahl 151	Primzahl 251
Paare insgesamt	22801	63001
keine elliptische Kurve	151	251
zyklische elliptische Kurve	17974 $\approx$ 79.4%	51875 $\approx$ 82.7%
Typ [x,2]	3576 $\approx$ 15.8%	10375 $\approx$ 16.5%
Typ [x,3]	775 $\approx$ 3.4%	0
Typ [x,5]	175 $\approx$ 0.8%	500 $\approx$ 0.8%
Typ [x,6]	150 $\approx$ 0.6%	0

Nun wollen wir noch zwei Extreme vorstellen. Bei Wahl der Charakteristik als $p = 167$ oder $p = 179$ waren alle elliptischen Kurven über dem Primkörper $\mathbb{F}_p$ entweder zyklisch oder vom Typ $[x, 2]$. Andere Typen kamen bei diesen Primkörpern nicht vor. Im einzelnen erhielten wir:

	Primzahl 167	Primzahl 179
Paare insgesamt	27889	32041
keine elliptische Kurve	167	179
zyklische elliptische Kurve	$23157 \approx 83.5\%$	$26611 \approx 83.5\%$
Typ [x,2]	$4565 \approx 16.5\%$	$5249 \approx 16.5\%$

Ein anderes Extrem erhielten wir bei Charakteristik $p = 43$. Dabei erhielten wir insgesamt 14 elliptische Kurven des Typs $[x, x]$. Wir geben von diesen elliptischen Kurven die Paare (a, b) an:

$$\begin{array}{ll} \text{Typ } [6,6]: & (0,1),\ (0,4),\ (0,11),\ (0,16),\ (0,21),\ (0,35),\ (0,41) \\ \text{Typ } [7,7]: & (0,3),\ (0,5),\ (0,12),\ (0,19),\ (0,20),\ (0,33),\ (0,37) \end{array}$$

Auch bei der Charakteristik $p = 211$ erhielten wir relativ viele elliptische Kurven des Typs $[x, x]$. Es gibt über dem Körper $\mathbb{F}_{211}$ jeweils 35 elliptische Kurven des Typs $[14, 14]$ bzw. $[15, 15]$. Dabei verteilten sich die anderen Typen wie folgt:

	Primzahl 43	Primzahl 211
Paare insgesamt	1849	44521
keine elliptische Kurve	43	211
zyklische elliptische Kurve	$1449 \approx 80.2\%$	$35175 \approx 79.4\%$
Typ [x,2]	$282 \approx 15.6\%$	$6860 \approx 15.5\%$
Typ [x,3]	$61 \approx 3.4\%$	$1469 \approx 3.3\%$
Typ [x,5]	0	$210 \approx 0.5\%$
Typ [x,6]	$7 \approx 0.1\%$	$316 \approx 0.7\%$
Typ [x,7]	$7 \approx 0.1\%$	$105 \approx 0.2\%$
Typ [x,10]	0	$105 \approx 0.2\%$
Typ [x,14]	0	$35 \approx 0.1\%$
Typ [x,15]	0	$35 \approx 0.1\%$

Diese Beispiele machen schon deutlich, daß es durchaus elliptische Kurven gibt, bei denen wir zur Bestimmung der Ordnung der Punktgruppe mit Hilfe des Shanks Algorithmus 2.1 den zweiten Teil dieses Algorithmus benötigen. Allerdings besitzt die überwiegende Anzahl von elliptischen Kurven (zumindest über den von uns betrachteten Körpern mit kleiner Charakteristik) einen solchen Typ, so daß der normale Shanks Algorithmus schon im ersten Teil terminiert. Damit ist experimentell unsere Aussage verifiziert, daß in der Praxis fast immer nur der erste Teil des Shanks Algorithmus benötigt wird.

Literaturverzeichnis

[At91] A.O.L. Atkin: *Private Kommunikation*

[BBCO91] C. Batut, D. Bernardi, H. Cohen, M. Olivier: *User's Guide to PARI-GP,* Version 1.35, 1991

[BuMü91] J. Buchmann, V. Müller: *Computing the Number of Points of Elliptic Curves over Finite Fields,* Proceedings of ISSAC 91, ACM Press, 1991, Seite 179 - 182

[ChRo88] L. Charlap, D. Robbins: *An Elementary Introduction to Elliptic Curves,* CRD Expository Report No. 31, Institute for Defense Analysis Princeton, 1988

[Co91] H. Cohen: *A Course in Algorithmic Algebraic Number Theory,* Manuskript, 1991

[Hun74] Th.W. Hungerford: *Algebra,* Graduate Texts in Mathematics 73, Springer-Verlag, 1974

[Hus87] D. Husemöller: *Elliptic Curves,* Graduate Texts in Mathematics 111, Springer-Verlag, 1987

[Ke82] H.L. Keng: *Introduction to Number Theory,* Springer-Verlag, 1982

[Kn73] D. Knuth: *The Art of Computer Programming,* vol. 2 Seminumerical Algorithms, Addison-Wesley, 1973

[Ko87] N. Koblitz: *Elliptic Curve Cryptosystems*, Mathematics of Computation, vol. 48, 1987, Seite 203 - 209

[La78] S. Lang: *Elliptic Curves: Diophantine Analysis,* Springer-Verlag, 1978

[Le87] H.W. Lenstra: *Factoring Integers with Elliptic Curves,* Annuals of Mathematics, vol. 126, 1987, Seite 649 - 678

[Me84] K. Mehlhorn: *Data Structures and Algorithms1: Sorting and Searching,* Springer-Verlag, 1984

[Mi86] V. Miller: *Uses of Elliptic Curves in Cryptography*, Advances in Cryptology: Proceedings of Crypto '85, Lecture Notes in Computer Science, vol. 218, 1986, Springer-Verlag, Seite 417 - 426

[MoOl90] F. Morain, J. Olivos: *Speeding up the Computations on an Elliptic Curve using Addition-Subtraction Chains,* Thèse, L'Université Claude Bernard-Lyon I, 1990

[Mu83] M.R. Murty: *On Artin's Conjecture,* Journal Number Theory, vol. 16, 1983, Seite 147 - 168

[Od86] A. Odlyzko: *Discrete Logarithms and their Cryptographic Significance*, Advances in Cryptology: Proceedings of Eurocrypt '84, Lecture Notes in Computer Science, vol. 209, 1985, Springer-Verlag, Seite 224 - 314

[RoSc62] J. Rosser, L. Schoenfeld: *Approximate Formulas for some Functions of Prime Numbers,* Illinois Journal of Mathematics, vol. 6, 1962, Seite 64 - 94

[Rü87] H.G. Rück: *A Note on Elliptic Curves over Finite Fields,* Mathematics of Computation, vol. 49, 1987, Seite 301 - 304

[Scho85] R. Schoof: *Elliptic Curves over Finite Fields and the Computation of Square Roots mod p,* Mathematics of Computation, vol. 44, 1985, Seite 483 - 494

[Sh70] D. Shanks: *Class Number, A Theory of Factorization and Genera,* Proceedings Symposium Pure Mathematics, vol. 20, American Mathematical Society, 1970, Seite 415 - 440

[Sh72] D. Shanks: *Five Number Theoretic Algorithms,* Congressus Numerantium No. VII, Proceedings 2nd Manitoba Conference on Numerical Mathematics, 1972, Seite 51 - 70

[Si86] J. Silverman: *The Arithmetic of Elliptic Curves,* Graduate Texts in Mathematics 106, Springer-Verlag, 1986