Effiziente Implementierung von Elliptischen und Hyperelliptischen Kurven für Anwendungen in der Kryptographie

von
Peter Schwabe

Diplomarbeit

im Studiengang Informatik

vorgelegt der
Fakultät für Mathematik, Informatik und Naturwissenschaften
der Rheinisch-Westfälischen Technischen Hochschule Aachen
im August 2006

Angefertigt am
Lehrstuhl für Theoretische Informationstechnik
bei
Prof. Dr. R. Mathar

Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst und keine als die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtliche oder sinngemäße Wiedergaben aus diesen Quellen sind als solche kenntlich gemacht und durch Zitate belegt.

Aachen, den 15.08.2006

Danksagung

Ich danke Prof. Dr. Mathar für die Vergabe und Betreuung des Themas sowie Prof. Dr. Vöcking für die Übernahme des Zweitgutachtens. Weiterhin danke ich Michael Naehrig für die umfangreiche fachliche Unterstützung, Daniel Catrein und Michael Reyer für ihre Ratschläge zur Programmierung in C++ sowie Tanja Lange für ihre Hilfsbereitschaft wenn ich an einigen Stellen nicht mehr weiterkam. Weiterhin gebührt meinen Eltern Dank für ihre Unterstützung, insbesondere meinem Vater dafür, dass er die ganze Arbeit aufmerksam gelesen und mich auf den einen oder anderen Fehler aufmerksam gemacht hat.

Inhaltsverzeichnis

Algorithmen

1 Einleitung

Elliptische Kurven sind algebraische Strukturen, die schon seit dem 19. Jahrhundert in der Mathematik eine wichtige Rolle spielen. Überlegungen von Mathematikern wie Dirichlet, Jacobi, Gauss, Abel und Eisenstein bildeten die Grundlage für die Theorie der elliptischen Kurven in der Form, wie sie heute bekannt ist.
Lange blieb die mathematische Theorie ohne praktische Anwendung, bis 1985 Hendrik Lenstra einen Algorithmus zur Faktorisierung von großen Ganzzahlen vorstellte, der auf elliptischen Kurven basiert [22]. Noch im gleichen Jahr präsentierten Neal Koblitz [17] und Victor Miller [26] unabhängig voneinander die Idee, elliptische Kurven in Kryptosystemen zu verwenden. In den folgenden Jahren wurde diese Idee weiterentwickelt, so dass 1999 der "Elliptic Curve Digital Signature Algorithm" ECDSA in ANSI X9.62 standardisiert werden konnte.
Mit der zunehmenden Bedeutung dieser Kryptosysteme rückte auch die Frage, wie diese und damit auch die zugrunde liegenden mathematischen Strukturen effizient implementiert werden können, immer mehr in den Mittelpunkt der Forschung.
1989 war es wiederum Neal Koblitz [18], der Kryptosysteme vorstellte, die auf hyperelliptischen Kurven basieren, womit auch Überlegungen zur Implementierung dieser Strukturen zum Thema der Forschung in Mathematik, Informatik und ingenieurwissenschaftlichen Bereichen wurde.
Diese Diplomarbeit zeigt zunächst, wie elliptische und hyperelliptische Kurven in der Kryptographie verwendet werden können, was bei der Wahl der Parameter zu beachten ist, und welche Verfahren es gibt, um die auf ihnen basierenden Kryptosysteme zu brechen. Anschließend werden der mathematische Hintergrund sowie Algorithmen zur effizienten Implementierung vorgestellt.
Im Rahmen der Diplomarbeit habe ich eine C++ Bibliothek programmiert, die es ermöglicht, auf beliebigen elliptischen und hyperelliptischen Kurven über endlichen Körpern zu rechnen. Für eine effiziente Ganzzahlarithmetik habe ich hierbei die GnuMP Bibliothek [2] verwendet, als einzige weitere Bibliothek kommt boost [1] zum Einsatz. Durch die Entscheidung, auch die endlichen Körper selbst zu implementieren, bleibt die Bibliothek weitestgehend unabhängig von externem Code, auch die GnuMP Bibliothek kommt in einer Weise zum Einsatz, die es ermöglicht, diese einfach durch eine andere oder durch eine eigene Implementierung zu ersetzen. Sollte man andererseits etwa für die Darstellung von Polynomen auf schnellere Implementierungen wie die in der NTL [4] zurückgreifen wollen, so ist es ohne großen Aufwand möglich, diese anstatt des hier verwendeten Codes einzubinden.
Für kryptographische Anwendungen sind nicht alle implementierten Funktionalitäten interessant, so werden etwa Kurven über Körpern der Charakteristik 3 kaum verwendet, auch gibt es hierzu bisher nur sehr wenige Veröffentlichungen. Dennoch sind diese Kur-

ven repräsentiert, auch wenn der Schwerpunkt bei der Optimierung eher bei den praktisch relevanten Fällen liegt. Bei der Implementierung von interessanten Spezialfällen habe ich eine Auswahl getroffen, für viele, wie etwa Koblitz-Kurven, optimale Erweiterungskörper oder auch hyperelliptische Kurven vom Geschlecht 3 findet sich in der Bibliothek allerdings keinerlei spezielle Optimierung, dennoch kann unter Verwendung von generischen Klassen in diesen Strukturen gerechnet werden.
Bei der Darstellung in dieser Arbeit werde ich auch insbesondere auf die Fälle und Algorithmen eingehen, die ich selbst implementiert habe, andere werde ich nur kurz darstellen oder an den entsprechenden Stellen auf die Literatur verweisen.

1.1 Anwendungen von elliptischen und hyperelliptischen Kurven in der Kryptographie

In den folgenden Kapiteln werden wir sehen, dass sowohl elliptische, als auch hyperelliptische Kurven geeignet sind um endliche abelsche Gruppen zu konstruieren, deren Verknüpfung üblicherweise additiv geschrieben wird. Für diese Gruppen lässt sich der diskrete Logarithmus definieren, die folgende Definition erfasst diesen ganz allgemein für additiv geschriebene endliche abelsche Gruppen:

Definition 1.1. Sei G eine endliche ablesche Gruppe der Ordnung n und sei $H \subseteq G$ eine zyklische Untergruppe der Ordnung m von G mit Erzeuger P. Für ein $Q \in H$ ist der *diskrete Logarithmus* zur Basis P von Q, $\log_P Q$ die eindeutige Zahl $a \in \mathbb{N}_0,\ a < m$, so dass $a \cdot P = Q$ ist.

1.1.1 Das Problem des diskreten Logarithmus und hierauf basierende Kryptosysteme

In der Regel ist die Berechnung des diskreten Logarithmus für Gruppen hinreichend großer Ordnung schwierig in dem Sinne, dass sie mit heutzutage vorhandenen technischen Hilfsmitteln nicht in angemessener Zeit durchzuführen ist. Daher wird das Problem, den diskreten Logarithmus zu berechnen auch als das *Problem des diskreten Logarithmus* (DLP) bezeichnet. Auf dem DLP basieren verschiedene Kryptosysteme, von denen einige im folgenden kurz vorgestellt werden sollen. Hierbei werden an einigen Stellen kryptographische Hashfunktionen (SHA1, MD5) vorausgesetzt.
Ein sehr bekanntes Verfahren zum Austausch eines gemeinsamen Schlüssels zwischen zwei Parteien A und B über einen unsicheren Kanal ist der Diffie-Hellman Schlüsselaustausch, der in Algorithmus 1 beschrieben ist.
Die nun folgenden Algorithmen gehen alle davon aus, dass die Gruppe G, sowie die zyklische Untergruppe H, deren Ordnung m und deren Erzeuger P bekannt sind. Da es sich bei diesen Verfahren um asymmetrische Verschlüsselungsverfahren handelt, muss weiterhin festgelegt sein, was der öffentliche - und was der private Schlüssel eines Benutzers ist: Der private Schlüssel ist ein $d \in [2, m-1]$, der öffentliche Schlüssel ist $Q = d \cdot P \in H$. Man sieht, dass man aus dem öffentlichen Schlüssel den privaten gerade durch Berechnung des diskreten Logarithmus zur Basis P erhält. Weiterhin geht man davon aus, dass man die

Algorithmus 1 Diffie-Hellman-Schlüsselaustausch

1. *Festlegung der Parameter:* A und B einigen sich auf eine endliche Gruppe G mit zyklischer Untergruppe H der Ordnung m mit Erzeuger P.
2. *Protokollaktionen:*
   - A wählt zufällig $a \in [2, m-1]$, berechnet und sendet zu $B:\ a \cdot P$
   - B wählt zufällig $b \in [2, m-1]$, berechnet und sendet zu $A:\ b \cdot P$
3. *Berechnung des gemeinsamen Schlüssels:* Jeder der beiden berechnet den gemeinsamen Schlüssel $a \cdot (b \cdot P) = b \cdot (a \cdot P)$

zu verschlüsselnde oder zu signierende Nachricht M als Element der Gruppe H darstellen kann.
Eines der meistverwendeten Verfahren zur asymmetrischen Verschlüsselung ist das ElGamal-Verfahren, auch dieses lässt sich auf beliebigen Gruppen durchführen, in denen das Problem des diskreten Logarithmus definiert ist, in einer allgemeinen Form ist es in Algorithmus 2 und Algorithmus 3 beschrieben.

Algorithmus 2 ElGamal-Verschlüsselung mit öffentlichem Schlüssel Q

1. Wähle ein zufälliges $k \in [2, m-1]$ und berechne $c_1 = k \cdot P$
2. Berechne $c_2 = M + k \cdot Q$
3. Der verschlüsselte Text ist (c_1, c_2)

Algorithmus 3 ElGamal-Entschlüsselung mit privatem Schlüssel d

1. Berechne $\gamma = d \cdot c_1 = k \cdot Q$
2. Berechne $M = c_2 - \gamma$

Asymmetrische Kryptosysteme lassen sich neben der Verschlüsselung auch zum Signieren von Nachrichten verwenden. Hierbei kommt meist das DSA-Verfahren zum Einsatz, das in der multiplikativen Gruppe $\mathbb{Z}_n^*$ operiert. Tatsächlich lassen sich aber auch hier beliebige endliche Gruppen einsetzen; die Algorithmen 4 und 5 setzen im Hinblick auf Punktemengen von elliptischen Kurven in ihrer Formulierung schon voraus, dass die Elemente der Gruppe Punkte mit x und y-Koordinate aus einem endlichen Körper $\mathbb{F}_q$ sind. Weiterhin verwendet das Verfahren eine kryptographische Hash-Funktion h.
Über diese häufig verwendeten Verfahren hinaus gibt es zahlreiche weitere, teilweise auch speziell für elliptische Kurven angepasste Verfahren. Erklärungen zu Systemen wie ECIES, PSEC oder EC-KCDSA finden sich zum Beispiel in [14].

Algorithmus 4 Erzeugung einer ECDSA-Signatur mit privatem Schlüssel d

1. Wähle zufälliges $k \in [2, m-1]$
2. Berechne $k \cdot P = (x_1, y_1)$, forme x_1 um in eine Ganzzahl x_1'
3. Berechne $r = x_1' \mod m$, wenn $r = 0$ gehe zu Schritt 1
4. Berechne $e = h(M)$
5. Berechne $s = k^{-1}(e + dr) \mod n$, wenn $s = 0$ gehe zu Schritt 1
6. Die Signatur ist das Tupel (r, s)

Algorithmus 5 Verifikation einer ECDSA-Signatur mit öffentlichem Schlüssel Q

1. Überprüfe, ob r und s im Intervall $[1, m-1]$ liegen, wenn nicht, lehne die Signatur ab
2. Berechne $e = h(M)$
3. Berechne $w = s^{-1} (\mathrm{mod}\ m)$
4. Berechne $u_1 = ew \mod m$ und $u_2 = rw \mod m$
5. Berechne $X = u_1 P + u_2 Q$
6. Wenn X das neutrale Element von H ist, lehne die Signatur ab
7. Forme die x-Koordinate von X in eine Ganzzahl x_1' um und berechne $v = x_1' \mod m$
8. Wenn $v = r$ akzeptiere die Signatur, sonst lehne die Signatur ab

Eine weitere interessante Anwendung von elliptischen und hyperelliptischen Kurven in der Kryptographie ist die sog. paarungsbasierte Kryptographie. Um die Ideen der hierzu gehörenden Verfahren skizzieren zu können, benötigt man zunächst die Definition einer Paarung (siehe [7, IX.1]):

Definition 1.2. Seien G_1 und G_2 additive abelsche Gruppen mit Exponent n, das heißt, dass $nP = 0$ für alle Elemente P aus G_1 bzw. G_2. Sei G_3 eine multiplikative zyklische Gruppe der Ordnung n mit neutralem Element 1. Eine Abbildung $e : G_1 \times G_2 \to G_3$ heißt *Paarung*, wenn die folgenden zwei Bedingungen erfüllt sind:

- e ist bilinear, das heißt, dass für alle $P, P' \in G_1$ und $Q, Q' \in G_2$ gilt, dass $e(P + P', Q) = e(P, Q) \cdot e(P', Q)$ und $e(P, Q + Q') = e(P, Q) \cdot e(P, Q')$.
- e ist nicht degeneriert:
  - Für alle $P \in G_1,\ P \neq 0$ gibt es ein $Q \in G_2$, so dass $e(P, Q) \neq 1$.
  - Für alle $Q \in G_2,\ Q \neq 0$ gibt es ein $P \in G_1$, so dass $e(P, Q) \neq 1$.

Häufig wird bei der Formulierung kryptographischer Protokolle vorausgesetzt, dass G_1 und G_2 identisch sind, dass also eine Abbildung der Form $\hat{e} : G_1 \times G_1 \to G_3$ mit den oben angegebenen Eigenschaften existiert, eine solche Paarung nennen wir im Folgenden *geeignete Paarung* und bezeichnen sie mit $\hat{e}$. Man sollte dabei allerdings beachten, dass sich eine geeignete Paarung nicht immer konstruieren lässt, Details zu diesem Problem finden sich in [7, IX.7.1 und X.1.2].
Die Gruppen G_1 und G_2 sind in den meisten Anwendungen Gruppen von Punktemengen elliptischer Kurven oder Gruppen von Divisorklassen hyperelliptischer Kurven, die Gruppe G_3 ist dann eine Untergruppe der multiplikativen Gruppe $\mathbb{F}_q{}^*$. Zunächst wurden Paarungen eingesetzt, um das DLP auf elliptischen und hyperelliptischen Kurven zu lösen (siehe Abschnitt 1.1.2), mittlerweile gibt es allerdings auch zahlreiche konstruktive Anwendungen. Hierzu zählen vor allem identitätsbasierte Verschlüsselungs- und Signaturverfahren — asymmetrische Verfahren, bei denen die Identitäten der Benutzer (zum Beispiel die e-Mail Adressen) gleichzeitig deren öffentliche Schlüssel darstellen, Algorithmen zur Konstruktion besonders kurzer Signaturen und spezielle Schlüsselaustauschprotokolle. Einen Überblick über diese Verfahren findet man in [7], einen Eindruck davon, wie Paarungen in der Kryptographie verwendet werden können, vermittelt Algorithmus 6, ein tripartites Schlüsselaustauschprotokoll, das nur einer Runde Informationsaustausch bedarf, um einen gemeinsamen geheimen Schlüssel zwischen drei Parteien festzulegen. Es wurde im Jahre 2000 von Joux in [16] vorgestellt.

1.1.2 Verfahren zum Berechnen des diskreten Logarithmus

Das offensichtlich einfachste Verfahren zum Lösen des DLP ist ein Brute-Force-Angriff. Bei geeigneten Gruppen (siehe Abschnitt 1.1.3) hat diese Attacke allerdings keinerlei Aussicht auf Erfolg. Es gibt allerdings einige ausgeklügeltere Verfahren, die im Folgenden vorgestellt werden sollen. Ein erstes Verfahren, das nicht mehr tut, als den hohen Zeitaufwand der vollständigen Suche zu senken, indem es einige Zwischenergebnisse speichert und so den

Algorithmus 6 Tripartites Schlüsselaustauschprotokoll nach Joux

1. *Festlegung der Parameter:* A, B und C einigen sich auf endliche abelsche Gruppen G_1 und G_3 wie in Definition 1.2 und eine geeignete Paarung $\hat{e} : G_1 \times G_1 \to G_3$
2. *Protokollaktionen:*
   - A wählt zufällig $a \in [2, n-1]$, berechnet und sendet zu B und C : $a \cdot P$
   - B wählt zufällig $b \in [2, n-1]$, berechnet und sendet zu A und C : $b \cdot P$
   - C wählt zufällig $c \in [2, n-1]$, berechnet und sendet zu A und B : $c \cdot P$
3. *Berechnung des gemeinsamen Schlüssels* Jede der drei Parteien berechnet den gemeinsamen Schlüssel $\hat{e}(bP, cP)^a = \hat{e}(aP, cP)^b = \hat{e}(aP, bP)^c$

Speicherbedarf erhöht ist das Babystep-Giantstep-Verfahren. Hier soll nicht näher auf dieses Verfahren eingegangen werden, Details finden sich in [24]. Interessant für das DLP auf elliptischen oder hyperelliptischen Kurven sind vor allem der Algorithmus von Pohlig und Hellmann, sowie Pollards ρ-Algorithmus:

Die Idee des Verfahrens von Pohlig und Hellmann ist es, das Problem des Berechnens des diskreten Logarithmus in einer zyklischen Gruppe H der Ordnung m zu zerlegen in die Berechnung des diskreten Logarithmus in Untergruppen primer Ordnung von H und diese dann mit Hilfe des Chinesischen Restsatzes (CRT, siehe [24, Algorithmus 2.121]) zusammenzusetzen. Hierzu ist es zunächst notwendig, die Primfaktorzerlegung der Ordnung m von H zu kennen, die im Allgemeinen nicht leicht zu berechnen ist. Bei vielen Gruppen ist diese allerdings bereits bekannt, oder kann – je nach Größe des kleinsten Primfaktors von m – recht effizient berechnet werden. Das Pohlig-Hellmann-Verfahren im Pseudocode ist in Algorithmus 7 beschrieben ([24, Algorithmus 3.63]).

Sind die Primfaktoren von m hinreichend klein, so ist der Schritt in Zeile 10 von Algorithmus 7 schnell zu berechnen und das DLP ist in der zyklischen Gruppe H nicht schwer genug für kryptographische Zwecke. Daher wählt man in der Regel eine Gruppe G, die eine *große* zyklische Untergruppe H *primer Ordnung* mit Erzeuger P besitzt, und betrachtet den diskreten Logarithmus zur Basis P. Dann hilft das Pohlig-Hellmann-Verfahren beim Berechnen des diskreten Logarithmus nicht weiter.
Ein Verfahren, dass auch sinnvoll verwendet werden kann, wenn die zyklische Gruppe H von primer Ordnung ist, ist Pollards ρ-Verfahren. Die Darstellung dieses Verfahrens folgt weitestgehend [14, Abschnitt 4.1.2]. Beim Pollard ρ-Verfahren handelt es sich um ein randomisiertes Verfahren, das auf der folgenden Überlegung basiert: Findet man zwei unterschiedliche Paare (c, d) und (c', d') jeweils aus $\mathbb{Z}_m \times \mathbb{Z}_m$ mit $cP + dQ = c'P + d'Q$, dann lässt sich der diskrete Logarithmus a von Q zur Basis P einfach berechnen, denn es

Algorithmus 7 Das Pohlig-Hellmann-Verfahren

Eingabe: Ein Erzeuger P einer additiven zyklischen Gruppe H der Ordnung m und ein Element $Q \in H$.
Ausgabe: Der diskrete Logarithmus $a = \log_P Q$.

```
Berechne die Primfaktorzerlegung n = ∏_{i=1}^{r} p_i^{e_i} von m
for (i ← 1; i ≤ r; i ← i + 1) do
    Berechne x_i = l_0 + l_1 p_i + ··· + l_{e_i−1} p_i^{e_i−1} mit x_i = x mod p_i^{e_i}
    q ← p_i, e ← e_i
    R ← 0, l_{−1} ← 0
    P̄ ← (m/q) P
    for (j ← 0; j ≤ e − 1; j ← j + 1) do
        R ← R + (l_{j−1} q^{j−1}) P
        Q̄ ← (m/q^{j+1}) (Q − R)
        l_j ← log_{P̄} Q̄
    end for
    x_i ← l_0 + l_1 q + ··· + l_{e−1} q^{e−1}
end for
Berechne x, 0 ≤ x ≤ m − 1 so, dass x ≡ x_i (mod p_i^{e_i}) ∀i = 1, ..., r (CRT)
return x
```

gilt:

$$
\begin{aligned}
(c-c')P &= (d-d')Q = (d-d')aP \\
\Rightarrow (c-c') &= (d-d')a \mod m \\
\Rightarrow a &= (c-c')(d-d')^{-1} \mod m
\end{aligned}
$$

Der naive Ansatz, zufällige c und d zu wählen und zusammen mit dem Ergebnis der Berechnung $cP+dQ$ zu speichern, löst das DLP zwar nach dem Geburtstagsparadoxon (siehe [24, Kapitel 2, 2.27]) in einer erwarteten Laufzeit von $O(\sqrt{m})$, hat aber den Nachteil, dass auch $O(\sqrt{m})$ Speicher benötigt wird. Pollards ρ-Verfahren hat eine vergleichbare asymptotische Laufzeit, benötigt aber wesentlich weniger Speicher. Hierzu definiert man eine sogenannte *iterierende Funktion* $F : H \to H$, so, dass sich für gegebenes $X \in H$ sowie $c, d \in [0, n-1]$ mit $X = cP + dQ$ der Wert der Funktion $\overline{X} = f(X)$ sowie $\overline{c}, \overline{d}$ mit $\overline{X} = \overline{c}P + \overline{d}Q$ leicht berechnen lassen. Außerdem sollte f die Eigenschaften einer Zufallsfunktion haben.
Eine Idee zur Konstruktion einer solchen Funktion f ist es, eine zufällige Partition $S_1, \ldots, S_L$ von H in L Mengen ungefähr gleicher Größe zu betrachten. Sei etwa $L = 32$, dann kann die Partition so gebildet werden, dass ein Element $X \in H$ ein Element von S_j ist, falls die letzten 5 Bit seiner Binärdarstellung (hierbei ist eine entsprechende Implementierung der Elemente der Gruppe H vorausgesetzt) gerade die Ganzzahl j darstellen.
Die Funktion $h : H \to \{1, 2, \ldots, L\},\ X \mapsto j,$ falls $X \in S_j$ nennt man Partitionsfunktion. Nun wählt man noch zufällige $a_j, b_j \in [0, n-1]$ für $1 \leq j \leq L$ und definiert die Funktion f durch:

$$f(X) = X + a_j P + b_j Q \text{ mit } j = h(X)$$

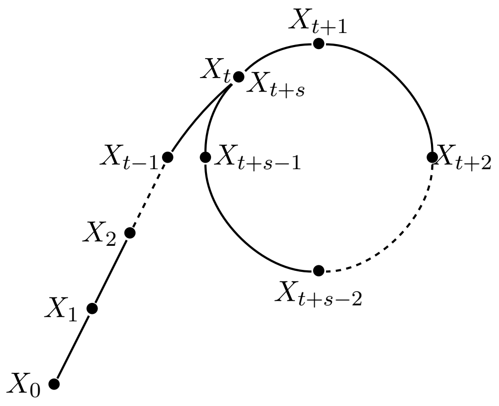

Abbildung 1.1: ρ-artige Form der Sequenz in Pollard's ρ-Algorithmus mit Schwanz (X_0 bis X_{t-1}) und Kopf (X_t bis X_{t+s-1})

Ist nun $X = cP + dQ$, so ergibt sich für $\overline{X} = f(X) = \overline{c}P + \overline{d}Q$ mit $\overline{c} = c + a_j \mod m$ und $\overline{c} = d + b_j \mod m$
Man nutzt nun aus, dass jedes Element $X_0 \in H$ eine Sequenz $\{X_i\}_{i \geq 0}$ definiert durch $X_i = f(X_{i-1})$. Da H eine endliche Gruppe ist, wird die Sequenz ab einem gewissen X_t zyklisch (siehe auch Abbildung 1.1), das heißt es existiert ein $s \in \mathbb{N}$, so dass gilt $X_t = X_{t+is}\ \forall i \in \mathbb{N}$. Man bezeichnet t als Schwanzlänge und s als Zykluslänge. Findet man nun eine Kollision, also $X_t = X_{t+s}$ mit $X_t = c_tP + d_tQ$ und $X_{t+s} = c_{t+s}P + d_{t+s}Q$ und sind weiter $d_t \neq d_{t+s}$, so kann man die obige Idee verwenden, um das DLP zu lösen. Algorithmus 8 stellt dieses Verfahren in Pseudocode dar, Aussagen über die erwartete Laufzeit und den zu erwartenden Speicherbedarf des Algorithmus finden sich in [14], auch Methoden, wie er parallelisiert werden kann, werden dort vorgestellt.

Das schnellste bekannte Verfahren zur Lösung des DLP ist der sog. *Index Calculus Algorithmus*. Dieses Verfahren hat allerdings die Eigenschaft, dass es sich nicht wie die vorigen Algorithmen ohne weiteres in beliebigen Gruppen verwenden lässt. Die Idee dieses Verfahrens ist die folgende: Zunächst wählt man eine Teilmenge $S = \{S_1, \ldots, S_t\}$ von H mit der Eigenschaft, dass sich viele Elemente von H als Linearkombination von Elementen aus S darstellen lassen, diese Teilmenge von H nennt man Faktorbasis.
Im nächsten Schritt sammelt man Relationen der Form

$$P = \sum_{i=1}^{t} c_i S_i, \quad c_i \in \mathbb{N}_0. \tag{1.1}$$

Diese lassen sich umformen zu

$$k \equiv \sum_{i=1}^{t} c_i \log_P S_i \mod m.$$

Algorithmus 8 Pollard's ρ-Algorithmus zur Berechnung des DLP

Eingabe: Ein Erzeuger P der additiven zyklischen Gruppe H der Ordnung m sowie ein $Q \in H$

Ausgabe: Der diskrete Logarithmus $a = \log_P Q$

Wähle den Betrag L der Partition von H (zum Beispiel 16 oder 32)
Wähle eine Partitionsfunktion $h : H \to \{1, 2, \dots L\}$
for $(j \leftarrow i; j \leq L; j \leftarrow j + 1)$ **do**
    Wähle zufällige $a_j, b_j \in [0, \dots, m - 1]$
    Berechne $R_j = a_j P + b_j Q$
end for
Wähle zufällige $c', d' \in [0, m - 1]$ und berechne $X' = c'P + d'Q$
$X'' \leftarrow X'$
$c'' \leftarrow c'$
$d'' \leftarrow d'$
repeat
    $j \leftarrow h(X')$
    $X' \leftarrow X' + R_j$
    $c' \leftarrow c' + a_j \mod m$
    $d' \leftarrow d' + b_j \mod m$
    for $(i \leftarrow 1; i \leq 2; i \leftarrow i + 1)$ **do**
        $j \leftarrow h(X'')$
        $X'' \leftarrow X'' + R_j$
        $c'' \leftarrow c'' + a_j \mod m$
        $d'' \leftarrow d'' + b_j \mod m$
    end for
until $X' = X''$
if $(d' = d'')$ **then**
    return "failure"
else
    Berechne $a = (c' - c'')(d' - d'') \mod m$
    return a
end if

Hat man mehr als $t + c$ dieser Relationen gesammelt, löst man das lineare Gleichungssystem der $t + c$ Gleichungen mit t Unbekannten. Hierbei ist c so zu wählen, dass das Gleichungssystem eine eindeutige Lösung hat. Nun kennt man die diskreten Logarithmen $\log_P S_1, \ldots, \log_P S_t$ und versucht für ein $k \in \mathbb{Z}$ eine Gleichung der Form

$$Q + kP = \sum_{i=1}^{t} d_i S_i, \quad d_i \in \mathbb{N}_0$$

zu finden. Diese Gleichung lässt sich umstellen zu

$$\log_P Q = (\sum_{i=1}^{t} d_i log_P S_i - k) \mod m$$

und somit lässt sich $\log_P Q$ leicht berechnen. Dass Index-Calculus nicht für beliebige Gruppen verwendbar ist, liegt an zwei Schritten des Algorithmus, die in der obigen Darstellung nicht detailliert dargestellt wurden: Zum einen ist nicht genauer erklärt, wie die Faktorbasis zu wählen ist. Im Allgemeinen ist dies nicht einfach, bisher existierende Verfahren setzen zusätzliche Eigenschaften der Gruppe H voraus. Zum zweiten bleibt die Frage, wie Relationen der Form 1.1 effizient zu generieren sind. Für einige Gruppen lässt sich dieses so effizient bewerkstelligen, dass die Gesamtlaufzeit subexponentiell ist ([24, Kapitel 3, 3.96, 3.70, 3.71, 3.72],[10]), gerade allerdings für Gruppen von Punktemengen auf elliptischen Kurven (siehe Kapitel 3) sowie die Divisorklassengruppe von hyperelliptischen Kurven vom Geschlecht 2 (siehe Kapitel 4) ist dies bisher nicht gelungen, daher eignen sich diese Gruppen besonders gut für kryptographische Zwecke.

Gerade die Tatsache, dass das DLP in diesen Gruppen nicht mit dem schnellsten bekannten Algorithmus lösbar ist, hat zu einer neuen Art von Angriffen geführt: Da die Gruppenordnung hier relativ klein gewählt werden kann, lohnt es sich, darüber nachzudenken, ob es nicht eine Möglichkeit gibt, das DLP in einer Gruppe G durch eine geeignete Abbildung in eine Gruppe G' zu übertragen, in der sich der schnelle Index-Calculus-Algorithmus wieder anwenden lässt. Diese Klasse von Angriffen gliedert sich in drei konkrete Verfahren, die hier nur kurz dargestellt werden sollen:
Den einfachsten Weg hierzu stellen offensichtlich Isomorphismen dar: Ist etwa das DLP in einer zyklischen abelschen Gruppe H primer Ordnung gegeben, so ist diese Gruppe isomorph zu jeder anderen zyklischen abelschen Gruppe gleicher Ordnung. Lässt sich nun ein Isomorphismus zwischen H und einer Gruppe H', in der das DLP leicht zu lösen ist, einfach berechnen, so lässt sich über diesen Isomorphismus auch das DLP in H effizient lösen. Ein Beispiel für eine solche Gruppe H' könnte etwa die additive Gruppe $\mathbb{Z}_p$ sein, in der sich das DLP mit Hilfe des euklidischen Algorithmus (siehe Algorithmus 13) in Polynomialzeit lösen lässt. Semaev stellt in [27] eine Klasse von elliptischen Kurven vor, für die diese Attacke anwendbar ist.
Eine weitere Idee zur Transformation des DLP in H in das DLP in H' ist die sog. MOV/Frey-Rück-Reduktion, sie basiert auf den in Definition 1.2 schon vorgestellten Paarungen. Die Idee der Anwendung dieser bilinearen Abbildung zur Lösung des DLP ist

die folgende: Handelt es sich etwa bei H um eine Untergruppe der Gruppe $E(\mathbb{F}_q)$ der Punkte auf einer elliptischen Kurve E über einem endlichen Körper $\mathbb{F}_q$ (siehe Kapitel 3) und ist weiter die Ordnung m von H prim und teilt nicht q, dann gibt es eine Paarung $e : H \times E(\mathbb{F}_{q^k}) \to {\mathbb{F}_{q^k}}^*$, wobei k die kleinste natürliche Zahl ist, für die gilt, dass $q^k \equiv 1 \pmod{|H|}$, sie wird als Einbettungsgrad bezeichnet. Diese Paarung wird nun verwendet, um das DLP in H in das DLP in ${\mathbb{F}_{q^k}}^*$ zu überführen, wie dies geschieht ist in Algorithmus 9 beschrieben, er stammt aus [7, Algorithmus IX.2].
In den meisten Fällen ist der Einbettungsgrad k so groß, dass der Schritt in Zeile 5 von

Algorithmus 9 MOV/Frey-Rück-Reduktion

Eingabe: $P, Q \in E(\mathbb{F}_q)$ von primer Ordnung, so, dass $Q = aP$
Ausgabe: Der diskrete Logarithmus $a = \log_P Q$
1: Berechne den Einbettungsgrad k von E hinsichtlich $|\langle P \rangle|$
2: Finde ein $S \in E(\mathbb{F}_{q^k})$, so dass $e(P, S) \neq 1$
3: Berechne $\zeta_1 = e(P, S)$
4: Berechne $\zeta_2 = e(Q, S)$
5: Verwende den Index-Calculus-Algorithmus um ein a zu bestimmen mit $\zeta_1^a = \zeta_2$
6: **return** a

Algorithmus 9 nicht effizient zu berechnen ist. Insbesondere für supersinguläre elliptische Kurven (siehe Definition 3.6) ist die Attacke allerdings durchführbar, hier ist k immer kleiner oder gleich sechs.
Für Gruppen von Divisorklassen hyperelliptischer Kurven sieht die Paarungs-Attacke ähnlich aus, auch hier sollte bei der Wahl geeigneter Parameter für kryptographische Zwecke darauf geachtet werden, dass der Einbettungsgrad groß ist.
Die dritte Art dieser Klasse von Attacken ist insbesondere interessant für Kurven, die über Körpern der Charakteristik 2 definiert sind, sie nennt sich Weil-Descent. Der Gedanke ist es, eine Transformation durchzuführen, die zu einer in gewisser Weise komplizierteren Kurve führt, die dann aber über einem kleineren Körper definiert ist. In vielen Fällen ist hier das DLP sogar schwieriger zu lösen, es gibt allerdings wenige elliptische Kurven, für die der Weil-Descent die Grundlage für einen erfolgreichen Angriff darstellen kann. Details hierzu finden sich in [9, Abschnitt 22.3].

1.1.3 Geeignete Gruppen für kryptographische Zwecke

Aus den Verfahren des vorigen Abschnittes ersieht man, welche Eigenschaften die Gruppe G haben muss, um für kryptographische Zwecke geeignet zu sein: Zum einen muss sie eine zyklische Untergruppe H primer Ordnung p besitzen, wobei p mindestens in der Größenordnung von 2^{160} liegen sollte. Weiter sollte diese Untergruppe H die Eigenschaft besitzen, dass eine Index-Calculus-Attacke nicht mit subexponentiellem Zeitaufwand durchzuführen ist, anderenfalls muss ihre Ordnung deutlich größer sein (ca. 2^{1024}).
Darüberhinaus sollte keine zu H isomorphe Gruppe H' existieren, so dass sich sowohl der Isomorphismus zwischen H und H' als auch das DLP in H' effizient lösen lassen.
Bei Gruppen, die auf elliptischen oder hyperelliptischen Kurven basieren, also solchen, die

wir im folgenden detailliert betrachten wollen, muss – um die Paarungs-Attacke zu vermeiden – der Einbettungsgrad k so groß sein, dass das DLP in $\mathbb{F}_{q^k}$ nicht effizient lösbar ist. Weiter sollte hier darauf geachtet werden, dass das Lösen des DLP über den Weil-Descent unmöglich ist.

2 Endliche Körper

Da für kryptographische Anwendungen vor allem elliptische bzw. hyperelliptische Kurven über endlichen Körpern relevant sind, stellt dieses Kapitel die notwendigen Informationen über endliche Körper zusammen. Ein besonderer Fokus wird hierbei auf Algorithmen zur Berechnung der Körperoperationen gerichtet, da diese die Grundlage für eine effiziente Implementierung von elliptischen und hyperelliptischen Kurven darstellen.

2.1 Mathematischer Hintergrund

Satz 2.1. *Sei p eine Primzahl. Dann ist $\mathbb{Z}/p\mathbb{Z}$ ein Körper mit p Elementen.*

Beweis. Da $\mathbb{Z}$ ein Hauptidealring ist, ist jedes Primideal von $\mathbb{Z}$ maximales Ideal. Also ist auch das Ideal $p\mathbb{Z}$ ein maximales Ideal und $\mathbb{Z}/p\mathbb{Z}$ ein Körper. □

Satz 2.2. *Sei $f \in \mathbb{F}_p[X]$ ein irreduzibles Polynom vom Grad k. Dann ist $\mathbb{F}_p[X]/f\mathbb{F}_p[X]$ ein Körper mit p^k Elementen.*

Beweis. Da $\mathbb{F}_p[X]$ ein faktorieller Ring ist, ist jedes irreduzible Element $f \in \mathbb{F}_p[X]$ auch ein Primelement. Weiterhin gilt, dass $\mathbb{F}_p[X]$ ein Hauptidealring ist, also ist das von f erzeugte Ideal $f\mathbb{F}_p[X]$ auch ein maximales Ideal, also ist $\mathbb{F}_p[X]/f\mathbb{F}_p[X]$ ein Körper. □

In [23, Theorem 2.5] und [23, Theorem 2.2] finden sich die folgenden wichtigen Aussagen:

Satz 2.3. *Sei K ein endlicher Körper. Dann hat K p^k Elemente, wobei die Primzahl p die* Charakteristik *von K und k der Erweiterungsgrad von K über seinem Teilkörper primer Ordnung ist.*

Satz 2.4. *Seien K_1 und K_2 zwei endliche Körper mit p^k Elementen, p prim, $k \in \mathbb{N}$. Dann sind K_1 und K_2 isomorph.*

Diese beiden Sätze rechtfertigen die Aussage, dass jeder endliche Körper die in 2.1 oder in 2.2 vorgestellte Form besitzt. Dies führt zu der Bezeichnung $\mathbb{F}_q$ für *den* endlichen Körper mit $q = p^k$ Elementen.

2.2 Implementierung endlicher Körper

In diesem Abschnitt soll zunächst vorgestellt werden, wie endliche Körper sowie deren Elemente repräsentiert werden können, um dann auf einige Algorithmen einzugehen, die beim Rechnen in endlichen Körpern verwendet werden. Hierbei ist es sinnvoll, Körper primer Ordnung getrennt von deren Erweiterungskörpern zu betrachten.

2.2.1 Körper primer Ordnung

Die Darstellung eines Elementes von $\mathbb{F}_p$ kann einfach geschehen, indem der Wert dieses Elementes, sowie die Primzahl p als Ganzzahlen gespeichert werden. Bei der Implementierung in C++, wie auch in zahlreichen anderen Programmiersprachen (z.B. C und Java) ist hierbei zu beachten, dass hierfür keine Standarddatentypen wie etwa `int` verwendet werden können, da die Werte zu groß sind. In meiner Implementierung kommt hier der Datentyp `mpz_t` aus der GnuMP-Bibliothek [2] zum Einsatz.
Die meistverwendeten Operationen auf den Elementen von $\mathbb{F}_p$ sind die Addition, die Multiplikation, die Quadrierung, die Division mit Rest sowie die Invertierung. Häufig verwendet man dabei die Operationen auf Ganzzahlen und reduziert das Ergebnis anschließend modulo p.
Da ich die Algorithmen nicht selbst implementiert habe, sondern auf die schnellen Algorithmen der GnuMP-Bibliothek zurückgegriffen habe, gebe ich an dieser Stelle nur einen kurzen Einblick in grundlegende Verfahren zur Multiplikation, Quadrierung und Invertierung modulo p, deutlich detailliertere Darstellungen finden sich in [9, Kapitel 10], [14, Kapitel 2, 2.2] und [11]. Im Folgenden verwende ich die Notation $(a_{n-1} \dots a_0)_2$ für die Bitfolge, die die Ganzzahl a repräsentiert.

Multiplikation großer Ganzzahlen

Die einfachste Form der Multiplikation von Ganzzahlen ist die Schulbuchmultiplikation: Gegeben zwei Ganzzahlen a und b mit Bitlängen n und m berechnet Algorithmus 10 das Produkt $a \cdot b$ in $O(n \cdot m)$.

Algorithmus 10 Schulbuchmultiplikation für Ganzzahlen

Eingabe: $a = (a_{n-1} \dots a_0)_2, b = (b_{m-1} \dots b_0)_2 \in \mathbb{Z}$
Ausgabe: Das Produkt $a \cdot b$
  $result \leftarrow 0$
  for $(i \leftarrow m-1; i \geq 0; i--)$ **do**
    $result \leftarrow 2 \cdot result$
    $result \leftarrow result + a \cdot b_i$
  end for
  return $result$

Gerade für große Zahlen a und b lohnt es sich, Algorithmen mit einer schnelleren asymptotischen Laufzeit zu betrachten, ein Beispiel für einen solchen Algorithmus ist die Karatsuba-Ofman-Multiplikation, die auf der Idee basiert, die Zahlen a und b zu zerlegen und auf rekursivem Weg das Produkt zu berechnen. In Algorithmus 11 ist dargestellt, wie dies für Zahlen gleicher Größe funktioniert, für Zahlen unterschiedlicher Größe sieht der Algorithmus sehr ähnlich aus (siehe etwa [9, Algorithmus 10.11]).
Der Algorithmus reduziert also das Problem der Multiplikation zweier Zahlen der Größe n auf drei Multiplikationen von Zahlen der halben Größe. Mit dem Master-Theorem (siehe etwa [15, Theorem 2.2.2.24]) lässt sich nun die Zeitkomplexität des Algorithmus analysieren, man kommt zu dem Ergebnis, dass die Karatsuba-Ofman-Multiplikation das Produkt

Algorithmus 11 Karatsuba-Ofman-Multiplikation für Ganzzahlen

Eingabe: $a = (a_{n-1} \dots a_0)_2, b = (b_{n-1} \dots b_0)_2 \in \mathbb{Z}$, Größenbegrenzung d_0
Ausgabe: Das Produkt $a \cdot b$
  if $(n \leq d_0)$ **then**
    return $a \cdot b$ (Berechnung mit Algorithmus 10)
  end if
  $s \leftarrow \lfloor n/2 \rfloor$
  $A_0 \leftarrow (a_s \dots a_0)$
  $B_0 \leftarrow (b_s \dots b_0)$
  $A_1 \leftarrow (a_{n-1} \dots a_{s+1})$
  $B_1 \leftarrow (b_{n-1} \dots b_{s+1})$
  Berechne durch rekursive Anwendung A_0B_0, A_1B_1 und $(A_0 + B_1)(A_1 + B_0)$
  return $A_1B_1 \cdot 2^{2s+2} + ((A_0 + A_1)(B_0 + B_1) - A_1B_1 - A_0B_0)2^{s+1} + A_0B_0$

von a und b in $O(n^{\log_2 3}) \approx O(n^{1,58})$ berechnet.

Quadrierung von Ganzzahlen

Intuitiv könnte man vermuten, dass eine Quadrierung effizienter zu bewerkstelligen ist als eine Multiplikation, schließlich kann man mit einem Multiplikationsalgorithmus immer auch quadrieren, was andersherum nicht möglich ist. Algorithmus 12 (siehe [9]) bestätigt diese Intuition, er benötigt für die Quadrierung einer Ganzzahl a mit n Bit $(n^2 + n)/2$ elementare Operationen wohingegen Algorithmus 10 n^2 elementare Operationen benötigt.

Algorithmus 12 Quadrierung einer Ganzzahl

Eingabe: $a = (a_{n-1} \dots a_0)_2 \in \mathbb{Z}$
Ausgabe: $r = (r_{2n-1} \dots r_0)_2$ mit $a^2 = r$
  Initialisiere alle $r_i,\ i = 0, \dots, 2n-1$ mit 0
  for $(i \leftarrow 0; i \leq n-1; i \leftarrow i+1)$ **do**
    $t \leftarrow a_i^2 + r_{2i}$
    $r_{2i} \leftarrow t \mod 2$
    $k \leftarrow \lfloor \frac{t}{2} \rfloor$
    for $(j = i+1; j \leq n-1; j \leftarrow j+1)$ **do**
      $t \leftarrow 2a_ia_j + r_{i+j} + k$
      $r_{i+j} \leftarrow t \mod 2$
      $k \leftarrow \lfloor \frac{t}{2} \rfloor$
    end for
    $r_{i+n} \leftarrow k$
  end for
  return $r = (r_{2n-1} \dots r_0)$

Invertierung modulo p

Grundlage der meisten Algorithmen zur Invertierung modulo einer Primzahl p ist der Erweiterte Euklidische Algorithmus. Es existieren einige Algorithmen mit geringerer Laufzeit (siehe etwa [9, Abschnitt 11.1.3]), in Algorithmus 13 ist die klassische Form des erweiterten Euklidischen Algorithmus beschrieben, die Darstellung basiert auf [8, Abschnitt 1.3.2].

Algorithmus 13 Der erweiterte euklidische Algorithmus zur Invertierung modulo p

Eingabe: Ganzzahlen a und b.
Ausgabe: Eine Tupel von Ganzzahlen (u, d, v) mit $a \cdot u + b \cdot v = d = \gcd(a, b)$

```
  u ← 1
  v ← 0
  d ← a
  v1 ← 0
  v3 ← b
  while (v3 ≠ 0) do
      q ← ⌊d/v3⌋
      t3 ← d mod v3
      t1 ← u − q·v1
      u ← v1
      d ← v3
      v1 ← t1
      v3 ← t3
  end while
  v ← (d − au)/b
  return (u, d, v)
```

Ist $\gcd(a, p) = 1$, so ist u in der Ausgabe von Algorithmus 13 das Inverse von a modulo p.

2.2.2 Erweiterungskörper von $\mathbb{F}_p$

Betrachtet man Körpererweiterungen eines Körpers primer Ordnung $\mathbb{F}_p$, so handelt es sich bei den Elementen dieses Körpers um Polynome, deren Grad durch den Erweiterungsgrad nach oben beschränkt ist. Für die Darstellung dieser Polynome gibt es mehrere Möglichkeiten: Die offensichtliche ist, die Koeffizienten in einem Vektor zu speichern. Diese Darstellung heißt Polynombasisdarstellung und ist die, die ich in meiner Implementierung verwende. Eine andere Möglichkeit der Darstellung verwendet die sogenannte Normalbasis. Der Gedanke ist hier der folgende: $\mathbb{F}_{p^k}$ ist ein k-dimensionaler Vektorraum über $\mathbb{F}_p$. Bei der Polynombasisdarstellung verwendet man zur Darstellung der Elemente dieses Vektorraumes die Basis $\{1, X, X^2, \ldots, X^{k-1}\}$, die Normalbasis besteht aus Potenzen $\alpha, \alpha^p, \alpha^{p^2}, \ldots, \alpha^{p^{k-1}}$ eines *normalen* Elementes α. Normal heißt ein Element $\alpha \in \mathbb{F}_{p^k}$ gerade dann, wenn $\alpha, \alpha^p, \alpha^{p^2}, \ldots, \alpha^{p^{k-1}}$ linear unabhängig sind. Gerade für $p = 2$ ist letztere Darstellung interessant, da in diesem Fall Quadrierungen besonders leicht zu berechnen sind, Multiplikationen werden aber ein wenig aufwändiger (siehe hierzu auch [9, Abschnitt

11.2.2.b]).
Die folgenden Algorithmen gehen von einer Darstellung über die Polynombasis aus, auch hier ist — ähnlich wie bei Körpern primer Ordnung — die Idee, zunächst Algorithmen wie Multiplikation, Quadrierung, Invertierung und Division in dem Polynomring $\mathbb{F}_p[X]$ durchzuführen und das Ergebnis anschließend modulo des irreduziblen Polynoms, das zur Definition des endlichen Körpers $\mathbb{F}_{p^k}$ verwendet wurde (siehe Satz 2.2) zu reduzieren, hierzu wird aber zunächst ein irreduzibles Polynom vom Grad k über $\mathbb{F}_p$ benötigt. Die folgenden Lemmata zeigen einen einfachen Weg auf, dieses zu finden:

Lemma 2.5. *[23, Theorem 3.20] Das Produkt aller irreduzibler Polynome über einem endlichen Körper $\mathbb{F}_p$, deren Grad n teilt, ist $X^{p^n} - X$*

Lemma 2.6. *Sei $f(X) \in \mathbb{F}_p[X]$ ein Polynom vom Grad k. Dann ist f irreduzibel genau dann, wenn $\gcd(f(X), X^{p^i} - X) \in \mathbb{F}_p\ \forall 1 \leq i \leq \lfloor \frac{k}{2} \rfloor$ ist.*

Beweis. “$\Rightarrow$”: Ist $\gcd(f(X), X^{p^i} - X) = g \notin \mathbb{F}_p$ für ein $i \in [1, \ldots, \lfloor \frac{k}{2} \rfloor]$, so gilt $\deg(g) < k$ und $g \mid f$, also ist f nicht irreduzibel.

“$\Leftarrow$”: Ist f nicht irreduzibel, so existiert ein $0 < i < k, i \mid k$ (also $i \leq \lfloor \frac{k}{2} \rfloor$) und ein normiertes irreduzibles Polynom $g \in \mathbb{F}_p[X]$ mit $\deg(g) = i$, so dass $g \mid f$. Aus Lemma 2.5 folgt, dass auch $g \mid X^{p^i} - X$, also ist $\gcd(f(X), X^{p^i} - X) = g \notin \mathbb{F}_p$.

□

Lemma 2.7. *[24, Fact 4.67] Sei $\mathbb{F}_p$ ein endlicher Körper primer Ordnung und sei $k \in \mathbb{N}$. Für die Anzahl $N_p(k)$ der irreduziblen Polynome vom Grad k über $\mathbb{F}_p$ gilt, dass:*

$$\frac{1}{2k} \leq \frac{N_p(k)}{p^k} \approx \frac{1}{k}.$$

Das heißt, dass die Wahrscheinlichkeit, dass ein zufällig gewähltes Polynom vom Grad k über $\mathbb{F}_p$ irreduzibel ist, ungefähr $\frac{1}{k}$ beträgt.

Also ist ein einfacher Ansatz zum Finden irreduzibler Polynome vom Grad k, so lange zufällige, normierte Polynome auf Irreduzibilität zu testen, bis man eines gefunden hat. Die erwartete Anzahl von solchen Tests ist gerade k. Dies führt zu dem Algorithmus 14. Der in diesem Algorithmus verwendete größte gemeinsame Teiler lässt sich mit dem erweiterten euklidischen Algorithmus berechen (siehe Algorithmus 13 und Abschnitt 2.2.2), die Potenz u^p in Zeile 6 lässt sich mit dem Square-and-Multiply-Algorithmus (siehe etwa Algorithmus 30 bzw. 31 oder [24, Algorithmus 2.143]) berechnen.

Multiplikation von Polynomen aus $\mathbb{F}_p[X]$

Die Konzepte, die schon zur Multiplikation von Ganzzahlen verwendet wurden, finden hier eine Entsprechung: Algorithmus 15 beschreibt die Schulbuchmultiplikation von Polynomen, Algorithmus 16 zeigt, dass sich auch eine Karatsuba-Ofman-Multiplikation ohne weiteres auf Polynome anwenden lässt:

Quadrierung von Polynomen aus $\mathbb{F}_p[X]$

Auch bei Polynomen lässt sich eine Quadrierung effizienter berechnen, als eine Multiplikation, in Algorithmus 17 ist gezeigt, wie dies geht.

Invertierung modulo f

In Algorithmus 13 ist gezeigt, wie zu $a, b \in \mathbb{Z}$ Werte $u, v, d \in \mathbb{Z}$ berechnet werden können, so dass gilt $au + bv = d = \gcd(a, b)$. Tatsächlich ist dieser Algorithmus allerdings nicht auf den Ring $\mathbb{Z}$ festgelegt, sondern kann in jedem euklidischen Ring, also insbesondere auch in $\mathbb{F}_p[X]$ verwendet werden.
Bei der Verwendung dieses Algorithmus zur Reduktion eines Polynoms $a \in \mathbb{F}_p[X]$ modulo des irreduziblen Polynoms $f \in \mathbb{F}_p[X]$ ist zu beachten, dass der größte gemeinsame Teiler zweier Elemente eines Ringes nur modulo der Einheiten definiert ist, das bedeutet, man erhält als Ausgabe ein $u \in \mathbb{F}_p[X]$, so dass $a \cdot u \mod f = d$, mit $d \in \mathbb{F}_p$. Um wirklich das Inverse von a modulo f zu erhalten, muss u also noch mit d^{-1} multipliziert werden.

Modulare Reduktion

Natürlich lässt sich sowohl bei Ganzzahlen, als auch bei Polynomen die modulare Reduktion durch eine Division mit Rest berechnen. Dies ist die Variante, die ich für Polynome implementiert habe, sie ist in Algorithmus 18 beschrieben, er verwendet die Notation $l(a)$ für den führenden Koeffizienten eines Polynomes a. Man kann allerdings auch ausnutzen, dass das Ergebnis der Division gar nicht benötigt wird und hierdurch auf effizientere Algorithmen kommen. Einige hiervon sind (für Ganzzahlen) in [9, Abschnitt 10.4] beschrieben.

2.2.3 Binäre Körper

Erweiterungskörper von $\mathbb{F}_2$, sogenannte binäre Körper haben einige Eigenschaften, die sie für praktische Anwendungen sehr interessant machen: Zunächst stellt man fest, dass eine Darstellung in Polynombasis nichts anderes als eine Bitfolge ist, wobei die einzelnen Bits gerade die Koeffizienten des Polynoms sind. Addition solcher Polynome kann durch eine einfache XOR-Operation auf diesen Bitfolgen realisiert werden, die Multiplikation mit Potenzen von X, wie sie in einigen Algorithmen verwendet wird ist gerade ein Shift und auch für aufwändigere Operationen existieren häufig effizientere Algorithmen, als für Polynome über endlichen Körpern ungerader Charakteristik. Gerade die modulare Reduktion und die Quadrierung sind hier hervorzuheben, sie sollen im folgenden kurz dargestellt werden:

Quadrierung von Polynomen aus $\mathbb{F}_2[X]$

Schon Algorithmus 17 basiert auf der Formel

$$\left(\sum_{i=0}^{n-1} a_i X^i\right)^2 = \sum_{i=0}^{n-1} a_i^2 X^{2i} + 2 \sum_{i<j} a_i a_j X^{i+j}.$$

Bei Polynomen über $\mathbb{F}_2$ ist der zweite Summand auf der rechten Seite Null, was dazu führt, dass sich das Quadrat eines Polynoms über $\mathbb{F}_2$ wie in Algorithmus 19 beschrieben in Linearzeit berechnen lässt. Der geschwindigkeitsbestimmende Schritt bei einer Quadrierung in F_{2^k} ist also die modulare Reduktion.

Modulare Reduktion von Polynomen aus $\mathbb{F}_2[X]$

Die modulare Reduktion von Polynomen über $\mathbb{F}_2$ ist mit Algorithmus 20 mit quadratischem Zeitaufwand zu berechnen. Für einige Reduktionspolynome ist es möglich, spezielle, effizientere Algorithmen zu entwickeln, siehe hierzu etwa [14, Abschnitt 2.3.5].

Algorithmus 14 Erzeugen eines irreduziblen Polynoms vom Grad k über $\mathbb{F}_p$

Eingabe: Primzahl p und Grad k
Ausgabe: Ein normiertes, irreduzibles Polynom f vom Grad k über $\mathbb{F}_p$

```
repeat
    Generiere zufällig ein normiertes Polynom f vom Grad k über F_p
    u ← X
    i ← true
    for (j ← 1; j ≤ ⌊k/2⌋; j ← j + 1) do
        u ← u^p mod f
        d ← gcd(f, u − X)
        if (d ∉ F_p) then
            i ← false
        end if
    end for
until i = true
return f
```

Algorithmus 15 Schulbuchmultiplikation von Polynomen aus $\mathbb{F}_p[X]$

Eingabe: $a = a_{n-1}X^{n-1} + a_{n-2}X^{n-2} \dots a_0, b = b_{m-1}X^{m-1} + b_{m-2}X^{m-2} \dots b_0 \in \mathbb{F}_p[X]$
Ausgabe: Das Produkt $a \cdot b$

```
result ← 0
for (i ← m − 1; i ≥ 0; i − −) do
    result ← X · result
    result ← result + a · b_i
end for
return result
```

Algorithmus 16 Karatsuba-Ofman Multiplikation für Polynome aus $\mathbb{F}_p[X]$

Eingabe: $a = a_{n-1}X^{n-1} + a_{n-2}X^{n-2} \dots a_0, b = b_{n-1}X^{n-1} + b_{n-2}X^{n-2} \dots b_0 \in \mathbb{F}_p[X]$, Größenbegrenzung d_0
Ausgabe: Das Produkt $a \cdot b$

```
if (n ≤ d_0) then
    return a · b (Berechnung mit Algorithmus 15)
end if
s ← ⌊n/2⌋
A_0 ← (a_s X^s ... a_0)
B_0 ← (b_s X^s ... b_0)
A_1 ← (a_{n−1} X^{n−1} ... a_{s+1} X^{s+1})
B_1 ← (b_{n−1} X^{n−1} ... b_{s+1} X^{s+1})
Berechne durch rekursive Anwendung A_0 B_0, A_1 B_1 und (A_0 + B_1)(A_1 + B_0)
return A_1 B_1 · X^{2s+2} + ((A_0 + A_1)(B_0 + B_1) − A_1 B_1 − A_0 B_0) X^{s+1} + A_0 B_0
```

Algorithmus 17 Quadrierung von Polynomen aus $\mathbb{F}_p[X]$

Eingabe: $a = a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \cdots + a_0 \in \mathbb{F}_p[X]$
Ausgabe: $r = r_{2n-2}X^{2n-2} + r_{2n-3}X^{2n-3} + \cdots + r_0$ mit $a^2 = r$
  Initialisiere alle $r_i,\ i = 0, \ldots, 2n-2$ mit 0
  for $(i \leftarrow 0; i \leq n-1; i \leftarrow i+1)$ **do**
    $r_{2i} \leftarrow a_i^2 + r_{2i}$
    for $(j = i+1; j \leq n-1; j \leftarrow j+1)$ **do**
      $r_{i+j} \leftarrow 2a_ia_j + r_{i+j}$
    end for
  end for
  return $r = r_{2n-2}X^{2n-2} + r_{2n-3}X^{2n-3} + \cdots + r_0$

Algorithmus 18 Schulbuchdivision von Polynomen aus $\mathbb{F}_p[X]$

Eingabe: $a, b \in \mathbb{F}_p[X], b \neq 0$
Ausgabe: $(q, r) \in \mathbb{F}_p[X] \times \mathbb{F}_p[X]$ mit $a = q \cdot b + r$
  $r \leftarrow a$
  $q \leftarrow 0$
  while $(\deg(r) \geq \deg(b))$ **do**
    $s \leftarrow \frac{l(r)}{l(b)} \cdot X^{\deg(r)-\deg(b)}$
    $q \leftarrow q + s$
    $r \leftarrow r - s \cdot b$
  end while
  return (q, r)

Algorithmus 19 Quadrierung eines Polynoms aus $\mathbb{F}_2[X]$

Eingabe: $a = a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \cdots + a_0 \in \mathbb{F}_2[X]$
Ausgabe: $r = r_{2n-2}X^{2n-2} + r_{2n-3}X^{2n-3} + \cdots + r_0$ mit $a^2 = r$
  for $(i \leftarrow 0; i \leq n-1; i \leftarrow i+1)$ **do**
    $r_{2i} \leftarrow a_i$
    $r_{2i+1} \leftarrow 0$
  end for
  return $r = r_{2n-2}X^{2n-2} + r_{2n-3}X^{2n-3} + \cdots + r_0$

Algorithmus 20 Modulare Reduktion eines Polynoms aus $\mathbb{F}_2[X]$

Eingabe: $a = a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \cdots + a_0, f = f_k X^k + f_{k-1}X^{k-1} + \cdots + f_0 \in \mathbb{F}_2[X]$
Ausgabe: $r = r_{k-1}X^{k-1} + r_{k-2}X^{k-2} + \cdots + r_0 = a \mod f$

$r \leftarrow a$
$m = f \cdot X^{n-1-k}$
for $(i \leftarrow n-1; i \geq k; i--)$ **do**
  if $(r_i = 1)$ **then**
    $r + m$
  end if
  $m \leftarrow m/X$
end for

3 Elliptische Kurven

3.1 Mathematischer Hintergrund

Definition 3.1. Seien K ein Körper und $\overline{K}$ der algebraische Abschluss von K. Desweiteren seien $a_1, a_2, a_3, a_4, a_6 \in K$. Dann definiert die folgende Gleichung eine *elliptische Kurve*:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{3.1}$$

wenn es keinen Punkt $(u, v) \in \overline{K} \times \overline{K}$ gibt, der sowohl die Gleichung E als auch beide partiellen Ableitungen $2y + a_1x + a_3 = 0$ und $-2x^2 - 2a_2x - a_4 + a_1y = 0$ erfüllt.

Um letztere Bedingung zu überprüfen, verwendet man in der Regel die sogenannte Diskriminante:

Definition 3.2. Sei $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ die Gleichung einer Kurve E. Die *Diskriminante* von E wird mit Δ bezeichnet und ist wie folgt definiert:

$$\begin{aligned}
&\Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \quad \text{mit} \\
&d_2 = a_1^2 + 4a_2 \\
&d_4 = 2a_4 + a_1a_3 \\
&d_6 = a_3^2 + 4a_6 \\
&d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2
\end{aligned}$$

Satz 3.3. *[28, Kapitel III,Proposition 1.4(a)(i)] Eine Kurve, die von einer Gleichung der Form 3.1 definiert wird, ist genau dann eine elliptische Kurve, wenn $\Delta \neq 0$.*

Definition 3.4. Sei K ein Körper, $L \subseteq \overline{K}$ ein Erweiterungskörper von K und E eine elliptische Kurve über K. Dann heißt die Menge

$$E(L) = \{(u, v) \in L \times L \mid v^2 + a_1uv + a_3v = u^3 + a_2u^2 + a_4u + a_6\} \cup \{\mathcal{O}\}$$

die Menge *L-rationaler Punkte auf E*. Das mit $\mathcal{O}$ bezeichnete Element dieser Menge ist der *Punkt im Unendlichen*. Die Menge $\overline{K}$-rationaler Punkte ist gerade die elliptische Kurve E.

Handelt es sich bei L um einen endlichen Körper, so ist auch die Menge der L-rationalen Punkte auf einer elliptischen Kurve E endlich und offensichtlich gilt, dass $|E(L)| \leq 2|L|+1$. Engere Schranken für den Betrag der Menge $E(L)$ gibt der Satz von Hasse:

Satz 3.5 (Satz von Hasse). *Sei E eine elliptische Kurve über $\mathbb{F}_q$, dann ist*

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

Das Intervall $[q + 1 - 2\sqrt{q}, q + 1 + 2\sqrt{q}]$ heißt Hasse Intervall.
Häufig wird der Satz von Hasse auch wie folgt formuliert: Sei E eine elliptische Kurve über $\mathbb{F}_q$, dann gilt

$$|E(\mathbb{F}_q)| \leq q + 1 - t, \ \textit{wobei}\ |t| \leq 2\sqrt{q}$$

Hierbei ist t die Spur *der elliptischen Kurve E über $\mathbb{F}_q$.*

Beweis. Siehe [28, Kapitel V, Satz 1.1] □

Definition 3.6. Sei E eine elliptische Kurve über $\mathbb{F}_q$ mit $|E(\mathbb{F}_q)| = q + 1 - t$. Dann heißt E *supersingulär*, falls die Charakteristik p von $\mathbb{F}_q$ die Spur t teilt, sonst heißt die Kurve *nicht-supersingulär*.

Die Form der Gleichung 3.1 wird als Weierstraß-Form bezeichnet. In der Regel wird allerdings mit Vereinfachungen der Weierstraß-Form gearbeitet, um diese darstellen zu können, benötigt man eine zusätzliche Definition:

Definition 3.7. [14] Seien E_1 und E_2 elliptische Kurven über K definiert durch die Weierstraß-Gleichungen

$$E_1 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$
$$E_2 : y^2 + \overline{a}_1xy + \overline{a}_3y = x^3 + \overline{a}_2x^2 + \overline{a}_4x + \overline{a}_6.$$

Dann heißen E_1 und E_2 *isomorph über* K, falls $u, r, s, t \in K$ existieren, so dass die Variablentransformation

$$(x, y) \rightarrow (u^2x + r, u^3y + u^2sx + t)$$

die Gleichung E_1 in die Gleichung E_2 überführt. Solch eine Variablentransformation heißt *zulässige Variablentransformation*.

Solche Variablentransformationen werden nun verwendet, um Gleichung 3.1 zu vereinfachen, man unterscheidet hierbei fünf Fälle [14]:

1. Sei K ein Körper der Charakteristik $\neq 2, 3$. Dann wird die Gleichung 3.1 durch die zulässige Variablentransformation

$$(x, y) \rightarrow \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1x}{216} - \frac{a_1^3 + 4a_1a_2 - 12a_3}{24} \right)$$

   in die Form

$$y^2 = x^3 + ax + b \quad a, b \in K \tag{3.2}$$

   überführt. Die Diskriminate Δ dieser Kurve ist $\Delta = -16(4a^3 + 27b^2)$.

2. Sei K ein Körper der Charakteristik 2 und sei $a_1 \neq 0$. In diesem Fall lässt sich zeigen, dass die elliptische Kurve nicht-supersingulär ist und dass die Gleichung 3.1 durch die zulässige Variablentransformation

$$(x, y) \to \left(a_1^2 x + \frac{a_3}{a_1}, a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3} \right)$$

die die Form

$$y^2 + xy = x^3 + ax^2 + b \quad a, b \in K \tag{3.3}$$

überführt wird. Die Diskriminante Δ dieser Kurve ist $\Delta = b$.

3. Sei K ein Körper der Charakteristik 2 und sei $a_1 = 0$. Dann ist die elliptische Kurve supersingulär und die zulässige Variablentransformation

$$(x, y) \to (x + a_2, y)$$

überführt Gleichung 3.1 in die Form

$$y^2 + cy = x^3 + ax + b. \tag{3.4}$$

Solch eine Kurve hat die Diskriminante $\Delta = c^4$.

4. Sei K ein Körper der Charakteristik 3 und sei $a_1^2 \neq -a_2$. Dann ist die Kurve nicht-supersingulär und Gleichung 3.1 wird durch die zulässige Variablentransformation

$$(x, y) \to \left(x + \frac{a_4 - a_1 a_3}{a_1^2 + a_2}, y + a_1 x + a_1 \frac{a_4 - a_1 a_3}{a_1^2 + a_2} + a_3 \right)$$

in die Form

$$y^2 = x^3 + ax^2 + b \tag{3.5}$$

überführt. Die Diskriminante der Kurve ist hier $\Delta = -a^3 b$

5. Sei K ein Körper der Charakteristik 3 und sei $a_1^2 = -a_2$. Dann ist die Kurve supersingulär und Gleichung 3.1 wird durch die zulässige Variablentransformation

$$(x, y) \to (x, y + a_1 x + a_3)$$

in die Form

$$y^2 = x^3 + ax + b \tag{3.6}$$

überführt. Hier ist die Diskriminante der Kurve $\Delta = -a^3$.

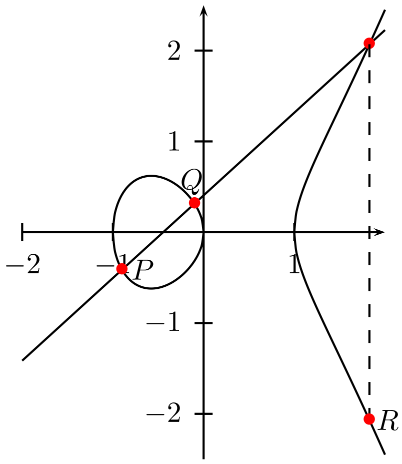

Abbildung 3.1: Punktaddition $P + Q = R$ auf $C : y^2 = x^3 - x$ über $\mathbb{R}$

3.1.1 Das Gruppengesetz von elliptischen Kurven

Auf der Menge L-rationaler Punkte lässt sich nun eine Verknüpfung $+$ so definieren, dass $(L, +)$ eine abelsche Gruppe bildet. Die Idee hierbei ist die folgende: Werden zwei *verschiedene* Punkte miteinander addiert, so betrachtet man die Gerade durch diese beiden Punkte, die einen dritten Schnittpunkt mit der elliptischen Kurve hat. Der gegenüberliegende Punkt dieses dritten Schnittpunktes ist das Ergebnis der Addition (siehe Abbildung 3.1).
Die Verdopplung eines Punktes verläuft ähnlich: Man bestimmt die Tangente an die elliptische Kurve in diesem Punkt und erhält einen weiteren Schnittpunkt mit der Kurve. Der gegenüberliegende Punkt dieses weiteren Schnittpunktes ist wieder das Ergebnis der Addition (siehe Abbildung 3.2).
Hierbei ist der gegenüberliegende Punkt eines Punktes P wie folgt definiert:

Definition 3.8. Sei E eine elliptische Kurve über K und $P = (x, y) \neq \mathcal{O}$ ein Punkt auf E. Dann ist $-P = (x, -y - (a_1x + a_3))$ der P *gegenüberliegende Punkt*.
Hat die Kurvengleichung von E die Form 3.2, 3.5 oder 3.6, so ist der gegenüberliegende Punkt gerade die Spiegelung von P an der x-Achse.

Existiert kein dritter Schnittpunkt der Geraden durch P und Q mit der Kurve, so ist Q gerade der P gegenüberliegende Punkt und das Ergebnis der Addition von P und Q ist der Punkt im Unendlichen $\mathcal{O}$.

Diese Idee der Addition über diese sogenannte Sehnen und Tangenten Methode lässt sich in explizite Formeln für die Addition überführen. Eine Herleitung dieser Formeln für den Fall einer Kurve mit einer Kurvengleichung der Form 3.2 sieht wie folgt aus (siehe [29]): Sei E eine elliptische Kurve über einem Körper K mit $\mathrm{char}(K) \notin \{2, 3\}$.

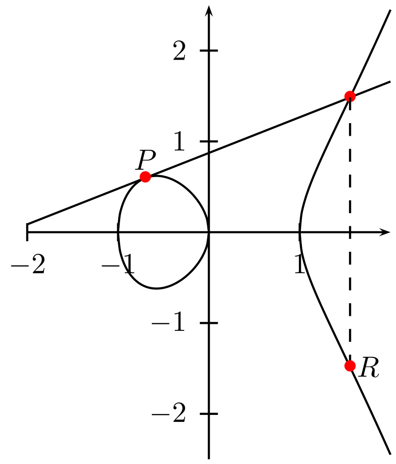

Abbildung 3.2: Punktverdopplung $2P = R$ auf $C : y^2 = x^3 - x$ über $\mathbb{R}$

Addition über Sekanten Seien $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ zwei Punkte auf E mit $x_1 \neq x_2$. Dann ist die Gleichung der Gerade durch P_1 und P_2 gegeben durch

$$y = m(x - x_1) + y_1 \text{ mit } m = \frac{y_1 - y_2}{x_1 - x_2},$$

oder abgekürzt $y = mx + n$. Setzt man dies in die Kurvengleichung ein so erhält man

$$(mx + n)^2 = x^3 + ax + b \Leftrightarrow 0 = x^3 - m^2x^2 + (a - 2mn)x + (b - n^2).$$

Über dem algebraischen Abschluss hat dieses Polynom drei Nullstellen, von denen zwei, nämlich x_1 uund x_2 bereits bekannt sind. Sei x_3 die dritte Nullstelle, dann erhält man:

$$x^3 - m^2x^2 + (a - 2mn)x + (b - n^2) = (x - x_1)(x - x_2)(x - x_3) = x^3 - (x_1 + x_2 + x_3)x^2 + \ldots$$

Vergleicht man die Koeffizienten von x^2 so erhält man

$$m^2 = x_1 + x_2 + x_3$$

und man erhält die Koordinaten des dritten Schnittpunktes der Geraden mit E durch

$$x_3 = m^2 - x_1 - x_2, \quad y_3 = m(x_3 - x_1) + y_1.$$

Addition über Tangenten Sei $P = (x_1, y_1)$ ein Punkt auf E mit $y_1 \neq 0$. Sei $y = mx + n$ die Gleichung der Tangente in P an E. Differenzierung der Kurvengleichung liefert $2yy' = 3x^2 + a$ und durch anschließendes Einsetzen von $P = (x_1, y_1)$ erhält man $2my_1 = 3x_1^2 + a$ und damit die Steigung m der Tangente:

$$m = \frac{3x_1^2 + a}{2y_1}$$

Durch Einsetzen der Tangentengleichung in die Kurvengleichung erhält man wiederum

$$(mx+n)^2 = x^3 + ax + b \Leftrightarrow 0 = x^3 - m^2x^2 + (a-2mn)x + (b-n^2)$$

Sei x_3 die wieder die dritte Nullstelle dieses Polynoms neben der Doppelnullstelle x_1, dann erhält man:

$$x^3 - m^2x^2 + (a-2mn)x + (b-n^2) = (x-x_1)^2(x-x_3) = (x^3) - (2x_1 + x_3)x^2 + \dots$$

Koeffizientenvergleich bei x^2 ergibt hier:

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

Die Herleitung der Formeln für Addition und Verdopplung für Kurven über Körpern der Charakteristik 2 oder 3 verläuft ähnlich, insgesamt erhält man für die fünf verschiedenen Fälle die folgenden Formeln:

1. Sei $E: y^2 = x^3 + ax + b$ eine elliptische Kurve über einem Körper K mit $\mathrm{char}(K) \notin \{2,3\}$.

   Punktaddition: Seien $P_1 = (x_1, y_1) \neq P_2 = (x_2, y_2)$ Punkte auf E. Ist $x_1 = x_2$, so ist $P_1 + P_2 = \mathcal{O}$, sonst ist $P_1 + P_2 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - x_1 - x_2 \quad \text{und}$$
$$y_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)(x_1 - x_3) - y_1$$

   Punktverdopplung: Sei $P_1 = (x_1, y_1)$ ein Punkt auf E. Ist $y_1 = 0$, so ist $2P_1 = \mathcal{O}$, sonst ist $2P_1 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \quad \text{und}$$
$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1$$

2. Sei $E: y^2 + xy = x^3 + ax^2 + b$ eine nicht-supersinguläre elliptische Kurve über einem Körper K der Charakteristik 2.

   Punktaddition: Seien $P_1 = (x_1, y_1) \neq P_2 = (x_2, y_2)$ Punkte auf E. Ist $x_1 = x_2$, so ist $P_1 + P_2 = \mathcal{O}$, sonst ist $P_1 + P_2 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + \left(\frac{y_1 + y_2}{x_1 + x_2}\right) + x_1 + x_2 + a \quad \text{und}$$
$$y_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + x_3 + y_1$$

Punktverdopplung: Sei $P_1 = (x_1, y_1)$ ein Punkt auf E. Ist $x_1 = 0$, so ist $2P_1 = \mathcal{O}$, sonst ist $2P_1 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{x_1 + y_1}{x_1}\right)^2 + \left(\frac{x_1 + y_1}{x_1}\right) + a \quad \text{und}$$
$$y_3 = x_1^2 + \left(\frac{x_1 + y_1}{x_1}\right) x_3 + x_3$$

3. Sei $E : y^2 + cy = x^3 + ax + b$ eine supersinguläre elliptische Kurve über einem Körper K der Charakteristik 2.

Punktaddition: Seien $P_1 = (x_1, y_1) \neq P_2 = (x_2, y_2)$ Punkte auf E. Ist $x_1 = x_2$, so ist $P_1 + P_2 = \mathcal{O}$, sonst ist $P_1 + P_2 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + x_1 + x_2 \qquad y_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + y_1 + c$$

Punktverdopplung: Sei $P_1 = (x_1, y_1)$ ein Punkt auf E. Dann ist $2P_1 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{x_1^2 + a}{c}\right)^2 \quad \text{und}$$
$$y_3 = \left(\frac{x_1^2 + a}{c}\right)(x_1 + x_3) + y_1 + c$$

4. Sei $E : y^2 = x^3 + ax^2 + b$ eine nicht-supersinguläre elliptische Kurve über einem Körper K der Charakteristik 3.

Punktaddition: Seien $P_1 = (x_1, y_1) \neq P_2 = (x_2, y_2)$ Punkte auf E. Ist $x_1 = x_2$, so ist $P_1 + P_2 = \mathcal{O}$, sonst ist $P_1 + P_2 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - a - x_1 - x_2 \quad \text{und}$$
$$y_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)(x_1 - x_3) - y_1$$

Punktverdopplung: Sei $P_1 = (x_1, y_1)$ ein Punkt auf E. Ist $y_1 = 0$, so ist $2P_1 = \mathcal{O}$, sonst ist $2P_1 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{ax_1}{y_1}\right)^2 - a - 2x_1 \quad \text{und}$$
$$y_3 = \left(\frac{ax_1}{y_1}\right)(x_1 - x_3) - y_1$$

5. Sei $E : y^2 = x^3 + ax + b$ eine supersinguläre elliptische Kurve über einem Körper K der Charakteristik 3.

Punktaddition: Seien $P_1 = (x_1, y_1) \neq P_2 = (x_2, y_2)$ Punkte auf E. Ist $x_1 = x_2$, so ist $P_1 + P_2 = \mathcal{O}$, sonst ist $P_1 + P_2 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - x_1 - x_2 \quad \text{und}$$
$$y_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)(x_1 - x_3) - y_1$$

Punktverdopplung: Sei $P_1 = (x_1, y_1)$ ein Punkt auf E. Ist $y_1 = 0$, so ist $2P_1 = \mathcal{O}$, sonst ist $2P_1 = P_3 = (x_3, y_3)$ mit

$$x_3 = \left(\frac{a}{2y_1}\right)^2 - 2x_1 \quad \text{und}$$
$$y_3 = \left(\frac{a}{2y_1}\right)(x_1 - x_3) - y_1$$

3.2 Implementierung elliptischer Kurven

3.2.1 Darstellung von Punkten

Die im vorigen Abschnitt dargestellten Formeln zur Berechnung der Gruppenoperation zu implementieren stellt keine wesentliche Schwierigkeit dar, allerdings ist zu beachten, dass jede dieser Berechnungen einer Invertierung in dem zugrundeliegenden Körper bedarf. Handelt es sich bei diesem Körper um einen endlichen Körper einer für kryptographische Zwecke geeigneten Größe, so ist die Invertierung in diesem Körper deutlich aufwändiger als etwa eine Multiplikation oder eine Quadrierung. Durch eine geeignete Darstellung der Punkte auf der elliptischen Kurve kann allerdings zum Preis von erhöhtem Speicherbedarf und mehr Multiplikationen diese Invertierung eingespart werden, was zu einer deutlich schnelleren Berechnung der Gruppenoperation führt.
Das grundsätzliche Konzept dieser Darstellung wird zum Beispiel auch in der Computergrafik eingesetzt, dort werden Punkte in sogenannten *homogenen Koordinaten* dargestellt. Diese sehen so aus, dass eine dritte Koordinate Z zu jedem Punkt (X, Y) hinzugefügt wird. Nun legt man fest, dass zwei solcher Darstellungen (X, Y, Z) und (X', Y', Z') dem gleichen Punkt entsprechen, wenn (X, Y, Z) gerade $(\gamma X', \gamma Y', \gamma Z')$ ist. Ist es nun im Rahmen einer Berechnung nötig, X durch ein a zu dividieren und Y durch ein b, so wird stattdessen X mit b multipliziert, Y mit a multipliziert und Z mit ab multipliziert, man erhält also statt $X/a, Y/b, 1$ die Darstellung Xb, Ya, ab, die aber dem gleichen Punkt in der Ebene entspricht.
Bei der Darstellung von Punkten aus $E(K)$ wird ein ähnliches Konzept verwendet, die folgende Definition führt die hierfür notwendigen theoretischen Grundlagen ein:

Definition 3.9. Sei K ein Körper, $\overline{K}$ der algebraische Abschluss von K und seien $c, d \in \mathbb{N}$. Zwei Elemente (X, Y, Z) und (X', Y', Z') aus $\overline{K}^3 \setminus \{(0,0,0)\}$ heißen äquivalent ($(X, Y, Z) \sim (X', Y', Z')$), wenn ein $\gamma \in \overline{K}$ existiert, so dass $(\gamma^c X, \gamma^d Y, \gamma Z) = (X', Y', Z')$.

Die Äquivalenzklasse, die (X, Y, Z) enthält, ist die Menge $\{\gamma X, \gamma Y, \gamma Z \mid \gamma \in \overline{K}^*\}$, sie wird mit $(X : Y : Z)$ bezeichnet. Eine solche Äquivalenzklasse $(X : Y : Z)$ heißt *projektiver Punkt*.
Die Menge aller projektiven Punkte bezeichnet man mit $\mathbb{P}_{c,d}(\overline{K})$. Die Menge aller projektiven Punkte, deren Z-Koordinate ungleich Null ist, wird mit $\mathbb{P}_{c,d}(\overline{K})^*$ bezeichnet.
Die Menge der Äquivalenzklassen aus $\mathbb{P}_{c,d}(\overline{K})$, die mindestens ein Element enthalten, dessen Koordinaten alle aus K sind, wird mit $\mathbb{P}_{c,d}(K)$ bezeichnet, auch hier bezeichnet $\mathbb{P}_{c,d}(K)^*$ die Menge der Elemente aus $\mathbb{P}_{c,d}(K)$, deren Z-Koordinate ungleich Null ist.

Man kann nun von der Menge $\mathbb{P}_{c,d}(K)^*$ in die Menge der affinen Punkte $K \times K$ eine bijektive Abbildung ϕ definieren durch:

$$\begin{aligned} \phi :& \mathbb{P}_{c,d}(K)^* \to \{(x, y) \mid x, y \in K\}, \\ & (X, Y, Z) \mapsto (X/Z^c, Y/Z^d). \end{aligned}$$

Die Umkehrabbildung ist offensichtlich

$$\begin{aligned} \phi^{-1} :& \{(x, y) \mid x, y \in K\} \to \mathbb{P}_{c,d}(K)^*, \\ & (x, y) \mapsto (x : y : 1). \end{aligned}$$

Definition 3.10. Die Menge der projektiven Punkte, deren Z-Koordinate Null ist, wird mit $\mathbb{P}_{c,d}(\overline{K})^0 = \{(X : Y : Z) \mid X, Y, Z \in K, Z = 0\}$ bezeichnet und heißt *Gerade im Unendlichen*.
Die Menge der projektiven Punkte, die auf der Gerade im Unendlichen liegen und mindestens ein Element enthalten, dessen Koordinaten alle in K sind, wird mit $\mathbb{P}_{c,d}(K)^0$ bezeichnet.

Die projektive Form der Kurvengleichung einer elliptischen Kurve E über einem Körper K erhält man, indem man in der Kurvengleichung x durch X/Z und y durch Y/Z ersetzt und anschließend mit dem Hauptnenner aller Terme multipliziert, für eine Kurve mit der allgemeinen Weierstraß-Gleichung $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ erhält man also zum Beispiel mit $c = d = 1$:

$$\begin{aligned} & Y^2/Z^2 + a_1XY/Z^2 + a_3Y/Z = X^3/Z^3a_2X^2/Z^2 + a_4X/Z + a_6 \\ \Rightarrow\ & E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3. \end{aligned}$$

Lemma 3.11. *Sei E eine elliptische Kurve über einem Körper K. Seien $c, d \in \mathbb{N}$. Dann gibt es genau einen projektiven Punkt $P \in \mathbb{P}_{c,d}(K)^0$, der die projektive Kurvengleichung von E erfüllt.*

Beweis. Ersetzt man in der allgemeinen Weierstrass-Gleichung x durch X/Z^c und y durch Y/Z^d, so erhält man:

$$Y^2/Z^{2d} + a_1XY/Z^{c+d} + a_3Y/Z^d = X^3/Z^{3c}a_2X^2/Z^{2c} + a_4X/Z^c + a_6$$

Der Hauptnenner aller dieser Summanden ist $\max\{Z^{2d}, Z^{3c}\}$. Multipliziert man mit diesem Hauptnenner und setzt dann $Z = 0$, so erhält man:

Für $2d > 3c$: $Y^2 = 0$, hier ist also der eine projektive Punkt aus $\mathbb{P}_{c,d}(K)^0$ gerade $P = (1 : 0 : 0)$

Für $2d < 3c$: $X^3 = 0$, der projektive Punkt aus $\mathbb{P}_{c,d}(K)^0$ ist hier $P = (0 : 1 : 0)$

Für $2d = 3c$: $Y^2 = X^3$, diese Gleichung wird erfüllt für jeden Punkt aus der Menge $\{(a^2 : a^3 : 0) \mid a \in K\} \subset \mathbb{P}_{c,d}(K)^0$.
Zu zeigen ist also, dass diese Menge genau ein Element, nämlich den projektiven Punkt $P = (1 : 1 : 0)$ enthält. Dafür zeigt man, dass es zu jedem $a \in K$ ein λ_a gibt, so dass $({\lambda_a}^c, {\lambda_a}^d, 0) = (a^2, a^3, 0)$ ist.
Da $2d = 3c$, existiert ein $g \in K$, so dass $c = 2g$ und $d = 3g$. Setze nun $\lambda_a = \sqrt[g]{a} \in \overline{K}$, dieses liefert das Gewünschte, also ist der einzige projektive Punkt $P \in \mathbb{P}_{c,d}(K)^0$, der die Kurvengleichung erfüllt der Punkt $P = (1 : 1 : 0)$.

□

Erfüllt nun ein Repräsentant (X, Y, Z) eines projektiven Punktes $P = (X : Y : Z) \in \mathbb{P}_{c,d}(K)$ die projektive Kurvengleichnung einer Kurve E, so tut dies auch jeder andere Repräsentant und für den Fall, dass $Z \neq 0$ erfüllt auch der affine Punkt $\phi^{-1}(X : Y : Z)$ die entsprechende affine Kurvengleichung. In diesem Fall sagt man, dass der projektive Punkt P auf der Kurve E liegt.
Lemma 3.11 besagt, dass genau ein Punkt aus $\mathbb{P}_{c,d}(K)^0$ auch auf der Kurve liegt, also haben die Menge der K-rationalen Punkte auf E und die der projektiven Punkte auf E die gleiche Mächtigkeit, man kann also jeden K-rationalen Punkt durch einen beliebigen Repräsentanten eines projektiven Punktes darstellen, man spricht dann von der Darstellung in *projektiven Koordinaten*.
Die Darstellung von Punkten in projektiven Koordinaten bietet bei der Implementierung zwei Vorteile gegenüber der affinen Darstellung: Zum einen kann man durch geeignete Wahl eines Repräsentanten eines projektiven Punktes Invertierungen im Laufe der Berechnung vermeiden, zum zweiten erhält man eine konsistente Darstellung des Punktes im Unendlichen $\mathcal{O}$.
Im folgenden wird kurz erläutert, welche projektiven Koordinaten für die verschiedenen elliptischen Kurven verwendet werden, also wie die Parameter c und d zu wählen sind um dann detailliert auf die resultierenden Algorithmen für Punktaddition und -verdopplung einzugehen. Bei der Betrachtung des Aufwandes eines solchen Punktadditionsalgorithmus ist es üblich, die Multiplikationen, Quadrierungen und Invertierungen im zugrundeliegenden Körper zu zählen; Additionen sind in der Regel so schnell, dass sie vernachlässigt werden können.

Elliptische Kurven über Körpern der Charakteristik $p \notin \{2, 3\}$

Bei der Implementierung elliptischer Kurven über Körpern, deren Charakteristik weder 2 noch 3 ist, kommen in der Regel sog. *Jacobische Koordinaten* zum Einsatz, hier ist $c = 2$ und $d = 3$. Die projektive Kurvengleichung ist dann $E : Y^2 = X^3 + aXZ^4 + bZ^6$. Der gegenüberliegende Punkt eines projektiven Punktes $(X : Y : Z)$ ist der Punkt $(X : -Y : Z)$,

der einzige projektive Punkt, der sowohl auf der Geraden im Unendlichen, als auch auf der Kurve liegt ist der Punkt $(1:1:0)$
Explizite Formeln für die Addition und Verdopplung erhält man, indem man in den entsprechenden Formeln für die affine Darstellung x durch X/Z^2 und y durch Y/Z^3 ersetzt und dann die Koordinaten X_R und Y_R des Ergebnisses auf die Form $X_R = X'_R/{Z'_R}^2$ und $Y_R = Y'_R/{Z'_R}^3$ bringt. Nun ist das Ergebnis der Addition der projektive Punkt (X'_R, Y'_R, Z'_R). Die Algorithmen 21 und 22 basieren auf [14], sie werden verwendet, um Punkte in projektiven Koordinaten auf einer elliptischen Kurve über einem Körper dessen Charakteristik weder 2 noch 3 ist zu addieren bzw. zu verdoppeln. Algorithmus 23 findet sich ebenfalls in [14] und ist eine Variante von Algorithmus 21, in der davon ausgegangen wird, dass die Z-Koordinate eines der beiden Punkte Eins ist, dass also die X und Y Koordinaten denen des entsprechenden affinen Punktes entsprechen. Wie wir in Abschnitt 3.2.2 sehen werden, ist dieser Spezialfall in der Praxis sehr relevant.

Algorithmus 21 Punktaddition auf $E: y^2 = x^3 + ax + b$ über K mit $\mathrm{char}(K) \notin \{2,3\}$, Jacobische Koordinaten

Eingabe: Projektive Punkte $P_1 = (X_1 : Y_1 : Z_1)$ und $P_2 = (X_2 : Y_2 : Z_2)$ auf $E : Y^2 = X^3 + aXZ^4 + bZ^6$ mit $P_1, P_2 \neq \mathcal{O}$
Ausgabe: Ein projektiver Punkt $R = (X_3 : Y_3 : Z_3)$ auf E mit $R = P + Q$

$r_1 \leftarrow {Z_2}^2$
$r_2 \leftarrow X_1 r_1$
$r_3 \leftarrow Y_1 Z_2 r_1$
$r_1 \leftarrow {Z_1}^2$
$r_5 \leftarrow X_2 r_1$
$r_6 \leftarrow Y_2 Z_1 r_1$
$r_5 \leftarrow r_5 - r_2$
$Z_3 \leftarrow Z_2 Z_1 r_5$
$r_6 \leftarrow r_6 - r_3$
$r_1 \leftarrow {r_5}^2$
$r_4 \leftarrow {r_6}^2$
$r_2 \leftarrow r_2 r_1$
$r_5 \leftarrow r_5 r_1$
$r_4 \leftarrow r_4 - r_5$
$r_1 \leftarrow 2r_2$
$X_3 \leftarrow r_4 - r_1$
$r_2 \leftarrow r_2 - X_3$
$r_6 \leftarrow r_6 r_2$
$r_1 \leftarrow r_3 r_5$
$Y_3 \leftarrow r_6 - r_1$
return $R = (X_3 : Y_3 : Z_3)$

Algorithmus 21 berechnet die Addition zweier Punkte mit vier Quadrierungen und zwölf Multiplikationen, liegt einer der Punkte in affinen Koordinaten vor, so kann mit Hilfe von Algorithmus 23 diese Punktaddition mit nur drei Quadrierungen und acht Multiplikatio-

Algorithmus 22 Punktverdopplung auf $E : y^2 = x^3+ax+b$ über K mit $\mathrm{char}(K) \notin \{2,3\}$, Jacobische Koordinaten

Eingabe: Ein projektiver Punkt $P = (X_1 : Y_1 : Z_1)$ auf $E : Y^2 = X^3 + aXZ^4 + bZ^6$
Ausgabe: Ein projektiver Punkt $R = (X_3 : Y_3 : Z_3)$ auf E mit $R = 2P$

if $(Y_1 = 0$ oder $Z_1 = 0)$ **then**
  return $\mathcal{O}$
else
  $r_1 = {Y_1}^2$
  $r_2 = 4X_1r_1$
  $r_3 = 8{r_1}^2$
  $r_4 = 3{X_1}^2 + a{Z_1}^4$
  $X_3 = {r_4}^2 - 2r_2$
  $Y_3 = r_4(r_2 - X_3) - r_3$
  $Z_3 = 2Y_1Z_1$
  return $R = (X_3 : Y_3 : Z_3)$
end if

Algorithmus 23 Punktaddition auf $E : y^2 = x^3 + ax + b$ über K mit $\mathrm{char}(K) \notin \{2,3\}$, Jacobische – affine Koordinaten

Eingabe: Projektive Punkte $P_1 = (X_1 : Y_1 : Z_1)$ und $P_2 = (X_2 : Y_2 : 1)$ auf $E : Y^2 = X^3 + aXZ^4 + bZ^6$ mit $P_1 \neq \pm P_2$ und $P_1 \neq \mathcal{O}$
Ausgabe: Ein projektiver Punkt $R = (X_3 : Y_3 : Z_3)$ auf E mit $R = P + Q$

$r_1 \leftarrow {Z_1}^2$
$r_2 \leftarrow Z_1r_1$
$r_3 \leftarrow X_2r_1$
$r_4 \leftarrow Y_2r_2$
$r_5 \leftarrow r_3 - X_1$
$r_6 \leftarrow r_4 - Y_1$
$r_7 \leftarrow {r_5}^2$
$r_8 \leftarrow r_7r_5$
$r_9 \leftarrow X_1r_7$
$X_3 \leftarrow {r_6}^2 - (r_8 + 2r_9)$
$Y_3 \leftarrow r_6(r_9 - X_3) - Y_1r_8$
$Z_3 \leftarrow Z_1r_5$
return $R = (X_3, Y_3, Z_3)$

nen berechnet werden. Algorithmus 22 berechnet das Doppelte eines Punktes mit sechs Quadrierungen und vier Multiplikationen im zugrundeliegenden Körper.

Nicht supersinguläre elliptische Kurven über Körpern der Charakteristik 2

Punkte auf nicht supersingulären Kurven über Körpern der Charakteristik 2 werden in der Regel in sog. López-Dahab-Koordinaten dargestellt, hier ist $c = 1$ und $d = 2$. Die projektive Form der Kurvengleichung wird zu $Y^2+XYZ = X^3Z+aX^2Z^2+bZ^4$, der einzige projektive Punkt aus $\mathbb{P}_{c,d}(K)$ auf der Kurve ist der Punkt $(1 : 0 : 0)$, Algorithmen zur Punktaddition und Verdopplung finden sich zum Beispiel in [21], sie sind in den Algorithmen 24, 25 und 26 beschrieben.

Algorithmus 24 Punktaddition auf $E : y^2 + xy = x^3 + ax^2 + b$ über K mit $\mathrm{char}(K) = 2$, López-Dahab Koordinaten

Eingabe: Projektive Punkte $P_1 = (X_1 : Y_1 : Z_1)$ und $P_2 = (X_2 : Y_2 : Z_2)$ auf $E : Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$ mit $P_1 \neq \pm P_2$ und $P_1, P_2 \neq \mathcal{O}$
Ausgabe: Ein projektiver Punkt $R = (X_3 : Y_3 : Z_3)$ auf E mit $R = P + Q$
$r_1 \leftarrow X_1Z_2$
$r_2 \leftarrow X_2Z_1$
$r_3 \leftarrow r_1 + r_2$
$r_4 \leftarrow {r_1}^2$
$r_5 \leftarrow {r_2}^2$
$r_6 \leftarrow r_4 + r_5$
$r_7 \leftarrow Y_1{Z_2}^2$
$r_8 \leftarrow Y_2{Z_1}^2$
$r_9 \leftarrow r_7 + r_8$
$r_{10} \leftarrow r_3r_9$
$Z_3 \leftarrow Z_1Z_2r_6$
$X_3 \leftarrow r_1(r_8 + r_5) + r_2(r_7 + r_4)$
$Y_3 \leftarrow (r_1r_{10} + r_7r_6) \cdot r_6 + (r_{10} + Z_3) \cdot X_3$
return $R = (X_3 : Y_3 : Z_3)$

Algorithmus 26 benötigt zur Berechnung der Punktaddition neun Multiplikationen (davon eine mit a) und fünf Quadrierungen in dem zugrundeliegenden Körper, Algorithmus 25 berechnet das Doppelte eines Punkte mit fünf Multiplikationen (davon eine mit a) und vier Quadrierungen.

Supersinguläre elliptische Kurven über Körpern der Charakteristik 2

Supersinguläre Kurven haben die Eigenschaft, dass ihr Einbettungsgrad (siehe Abschnitt 1.1.3) kleiner oder gleich sechs ist, sie eignen sich daher nur sehr bedingt für kryptographische Zwecke. Aus diesem Grund gibt es nur wenig Literatur über die effiziente Implementierung solcher Kurven, in meiner Implementierung verwende ich die affine Darstellung mit den ensprechenden Formeln für Punktaddition und -verdopplung.

Algorithmus 25 Punktverdopplung auf $E: y^2+xy = x^3+ax^2+b$ über K mit $\mathrm{char}(K) = 2$, López-Dahab Koordinaten

Eingabe: Ein projektiver Punkt $P = (X_1 : Y_1 : Z_1)$ auf $E: Y^2+XYZ = X^3Z+aX^2Z^2+bZ^4$

Ausgabe: Ein projektiver Punkt $R = (X_3 : Y_3 : Z_3)$ auf E mit $R = 2P$

$r_1 \leftarrow {X_1}^2$
$r_2 \leftarrow r_1 + Y_1$
$r_3 \leftarrow X_1 Z_1$
$Z_3 \leftarrow {r_3}^2$
$r_3 \leftarrow r_2 r_3$
$X_3 \leftarrow {r_2}^2 + r_3 + aZ_3$
$Y_3 \leftarrow (Z_3 + r_3)X_3 + {r_1}^2 Z_3$
return $R = (X_3 : Y_3 : Z_3)$

Algorithmus 26 Punktaddition auf $E: y^2 + xy = x^3 + ax^2 + b$ über K mit $\mathrm{char}(K) = 2$, López-Dahab – affine Koordinaten

Eingabe: Projektive Punkte $P_1 = (X_1 : Y_1 : Z_1)$ und $P_2 = (X_2 : Y_2 : 1)$ auf $E: Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$ mit $P_1 \neq \pm P_2$

Ausgabe: Ein projektiver Punkt $R = (X_3 : Y_3 : Z_3)$ auf E mit $R = P + Q$

$r_1 \leftarrow {Z_1}^2 Y_2 + Y_1$
$r_2 \leftarrow Z_1 X_2 + X_1$
$r_3 \leftarrow Z_1 r_2$
$Z_3 \leftarrow {r_3}^2$
$r_4 \leftarrow Z_3 X_2$
$r_5 \leftarrow X_2 + Y_2$
$X_3 \leftarrow {r_1}^2 + r_3(r_1 + {r_2}^2 + ar_3)$
$Y_3 \leftarrow (r_4 + X_3)(r_3 r_1 + Z_3) + {Z_3}^2 r_5$
return $R = (X_3 : Y_3 : Z_3)$

Nicht supersinguläre elliptische Kurven über Körpern der Charakteristik 3

Elliptische Kurven über Körpern der Charakteristik drei werden nur sehr wenig in der Literatur erwähnt, was wohl daran liegt, dass sich die zugrundeliegenden Körper nur deutlich weniger effizient implementieren lassen als etwa Körper der Charakteristik 2 oder Körper primer Ordnung. Die Algorithmen, die ich in meiner Implementierung zur Punktaddition und -verdopplung verwende, basieren auf Jacobischen projektiven Koordinaten, sind aber vermutlich nicht optimal im Sinne einer minimalen Anzahl von Multiplikationen und Quadrierungen im zugrundeliegenden Körper. Die projektive Kurvengleichung solcher Kurven hat bei Verwendung Jacobischer projektiver Koordinaten die Form $E : Y^2 = X^3 + aX^2Z^4 + bZ^6$, die Algorithmen 27, 28 und 29 sind die, die ich in meiner Implementierung verwendet habe.
Algorithmus 27 benötigt zur Addition zweier Punkte, die in Jacobischen projektiven Koordinaten vorliegen, vier Quadrierungen und 14 Multiplikationen (davon eine mit a), Algorithmus 28 berechnet das Doppelte eines Punktes mit neun Multiplikationen (davon zwei mit a) und sechs Quadrierungen, liegt einer der Punkte in affinen Koordinaten vor, so lässt sich eine Addition auch mit nur vier Quadrierungen und neun Multiplikationen (davon eine mit a) berechnen, wie in Algorithmus 29 gezeigt ist.

Supersinguläre elliptische Kurven über Körpern der Charakteristik 3

Auch supersinguläre Kurven über Körpern der Charakteristik drei sind aufgrund des kleinen Einbettungsgrades für kryptographische Zwecke eher uninteressant, da aber die Kurvengleichung genauso aussieht, wie die von Kurven über Körpern, deren Charakteristik weder zwei noch drei ist, lassen sie sich sehr einfach in Jacobischen projektiven Koordinaten implementieren.
Punktadditions- und verdopplungsalgorithmen für diese Klasse von Kurven ergeben sich aus den Algorithmen 21, 22 und 23 einfach dadurch, dass Terme, in denen 3 als Faktor vorkommt, gleich Null gesetzt werden.

3.2.2 Punktmultiplikation

Um das Problem des diskreten Logarithmus in einer Gruppe (siehe Definition 1.1) für kryptographische Anwendungen nutzen zu können, ist es notwendig, effizient das k-Fache des Erzeugers einer großen Untergruppe zu berechnen, wobei k eine große Ganzzahl ist. Im folgenden werden nun Algorithmen vorgestellt, die diese Berechnung in beliebigen zyklischen additiven Gruppen durchführen, sie eignen sich also sowohl zur Berechnung des Vielfachen eines Punktes auf einer elliptischen Kurve, als auch etwa zur Berechnung des Vielfachen einer Divisorklasse (siehe Kapitel 4)). Die hier vorgestellten Algorithmen basieren alle auf der Darstellung in [14], dort findet man auch noch detaillierte Beschreibungen einiger speziellerer Verfahren, auf die ich am Ende dieses Abschnittes nur kurz eingehen werde.
Der klassische Algorithmus zum Berechnen des Vielfachen eines Gruppenelementes, ist der sog. *Double and Add*-Algorithmus, er ist für multiplikative Gruppen auch unter dem Namen *Square and Multiply* bekannt. Die Idee basiert darauf, den Skalar Bit für Bit zu lesen

Algorithmus 27 Punktaddition auf $E : y^2 = x^3 + ax^2 + b$ über K mit $\mathrm{char}(K) = 3$, Jacobische Koordinaten

Eingabe: Projektive Punkte $P_1 = (X_1 : Y_1 : Z_1)$ und $P_2 = (X_2 : Y_2 : Z_2)$ auf $E : Y^2 = X^3 + aX^2Z^4 + bZ^6$ mit $P_1 \neq \pm P_2$ und $P_1, P_2 \neq \mathcal{O}$

Ausgabe: Ein projektiver Punkt $R = (X_3 : Y_3 : Z_3)$ auf E mit $R = P + Q$

$r_1 = {Z_2}^2$
$r_2 = X_1 r_1$
$r_3 = Y_1 Z_2$
$r_3 = r_3 r_1$
$r_1 = {Z_1}^2$
$r_5 = X_2 r_1$
$r_6 = Y_2 Z_1$
$r_6 = r_1 r_6$
$r_5 = r_5 - r_2$
$r_7 = Z_2 Z_1$
$Z_3 = r_7 r_5$
$r_6 = r_6 - r_3$
$r_8 = r_6$
$r_1 = {r_5}^2$
$r_4 = {r_6}^2$
$r_2 = r_2 r_1$
$r_5 = r_5 r_1$
$r_4 = r_4 - r_5$
$r_1 = 2r_2$
$r_4 = r_4 - r_1$
$r_2 = r_2 - r_4$
$r_6 = r_6 r_2$
$r_1 = r_3 r_5$
$r_1 = r_6 - r_1$
$r_9 = a{Z_3}^2$
$X_3 = r_4 - r_9$
$Y_3 = r_1 + r_9 r_8$
return $R = (X_3 : Y_3 : Z_3)$

Algorithmus 28 Punktverdopplung auf $E : y^2 = x^3 + ax^2 + b$ über K mit $\mathrm{char}(K) = 3$, Jacobische Koordinaten

Eingabe: Ein projektiver Punkt $P = (X_1 : Y_1 : Z_1)$ auf $E : Y^2 = X^3 + aX^2Z^4 + bZ^6$
Ausgabe: Ein projektiver Punkt $R = (X_3 : Y_3 : Z_3)$ auf E mit $R = 2P$

```
if (Y_1 = 0 oder Z_1 = 0) then
    return O
else
    r_1 = X_1^2
    r_2 = Z_1^2
    r_3 = Y_1^2
    r_4 = r_3^2
    r_5 = r_2^2
    Z_3 = Y_1 Z_1
    X_3 = a(r_1 r_5 a − Z_3^2) − 2 X_1 r_3
    Y_3 = a r_2 (r_1 r_3 − X_1 X_3) − r_4
    return R = (X_3 : Y_3 : Z_3)
end if
```

if ($Y_1 = 0$ oder $Z_1 = 0$) **then**
  return $\mathcal{O}$
else
  $r_1 = {X_1}^2$
  $r_2 = {Z_1}^2$
  $r_3 = {Y_1}^2$
  $r_4 = {r_3}^2$
  $r_5 = {r_2}^2$
  $Z_3 = Y_1 Z_1$
  $X_3 = a(r_1 r_5 a - {Z_3}^2) - 2X_1 r_3$
  $Y_3 = a r_2 (r_1 r_3 - X_1 X_3) - r_4$
  return $R = (X_3 : Y_3 : Z_3)$
end if

Algorithmus 29 Punktaddition auf $E : y^2 = x^3 + ax^2 + b$ über K mit $\mathrm{char}(K) = 3$, Jacobische – affine Koordinaten

Eingabe: Projektive Punkte $P_1 = (X_1 : Y_1 : Z_1)$ und $P_2 = (X_2 : Y_2 : 1)$ auf $E : Y^2 = X^3 + aX^2Z^4 + bZ^6$ mit $P_1 \neq \pm P_2$ und $P_1 \neq \mathcal{O}$
Ausgabe: Ein projektiver Punkt $R = (X_3 : Y_3 : Z_3)$ auf E mit $R = P + Q$

$r_1 = {Z_1}^2$
$r_2 = Z_1 r_1$
$r_3 = X_2 r_1$
$r_4 = Y_2 r_2$
$r_5 = r_3 - X_1$
$r_6 = r_4 - Y_1$
$r_7 = {r_5}^2$
$r_8 = r_7 r_5$
$r_9 = X_1 r_7$
$Z_3 = Z_1 r_5$
$X_3 = {r_6}^2 - (r_8 + 2r_9) - a{Z_3}^2$
$Y_3 = r_6(r_9 - X_3) - Y_1 r_8$
return $R = (X_3 : Y_3 : Z_3)$

und durch eine geeignete Abfolge von Verdopplungen und Additionen das Vielfache des Gruppenelementes zu berechnen. Algorithmus 30 zeigt, wie dies geht, wenn der Skalar von rechts nach links gelesen wird, Algorithmus 31 liest den Skalar von links nach rechts.

Algorithmus 30 Double and Add, Lesen des Skalars von rechts nach links

Eingabe: $k = (k_{t-1}, \dots, k_0), P$ Erzeuger einer zyklischen additiven Gruppe
Ausgabe: $k \cdot P$

$Q \leftarrow \mathcal{O}$
for $(i = 0, i \leq t-1, i \leftarrow i+1)$ **do**
    if $(k_i = 1)$ **then**
        $Q \leftarrow Q + P$
    end if
    $P \leftarrow 2P$
end for
return Q

Algorithmus 31 Double and Add, Lesen des Skalars von links nach rechts

Eingabe: $k = (k_{t-1}, \dots, k_0), P$ Erzeuger einer zyklischen additiven Gruppe
Ausgabe: $k \cdot P$

1: $Q \leftarrow \mathcal{O}$
2: **for** $(i = t-1, i \geq 0, i--)$ **do**
3:     $Q \leftarrow 2Q$
4:     **if** $(k_i = 1)$ **then**
5:         $Q \leftarrow Q + P$
6:     **end if**
7: **end for**
8: **return** Q

Beim Lesen des Skalars von links nach rechts ist bei der Addition eines der beiden Elemente immer P (Zeile 5 in Algorithmus 31), handelt es sich bei der Gruppe um die Gruppe von Punkten auf einer elliptischen Kurve, so wie im vorigen Abschnitt vorgestellt und werden die Punkte in projektiven Koordinaten dargestellt, lohnt es sich also, vor der Vervielfachung die affine Darstellung dieses Punktes zu berechnen, da dies zu deutlich schnelleren Additionsalgorithmen führt, wie wir im vorigen Abschnitt gesehen haben.
Betrachtet man den Aufwand von Algorithmus 30 und 31, so stellt man fest, dass die Anzahl der Verdopplungen in der Gruppe gerade die Anzahl von Bits des Skalars k ist, die Anzahl der Additionen ist genau die Anzahl der Bits von k die Eins sind, also im Schnitt gerade $n/2$, wobei n die Bitlänge von k ist. Durch eine andere Darstellung des Skalars k kann nun die Anzahl der Additionen reduziert werden, diese Darstellung heißt *non-adjacent-form* oder kurz NAF:

Definition 3.12. [14, Definition 3.28] Die *non-adjacent form* (NAF) einer natürlichen

Zahl k ist ein Ausdruck

$$k = \sum_{i=0}^{l-1} k_i 2^i, \text{ wobei } k_i \in \{-1, 0, 1\}, \; k_{l-1} = 1$$

und von jeweils zwei aufeinanderfolgenden Stellen k_j und k_{j+1} ist mindestens eine gleich Null. Die Länge der NAF ist l.

Diese Darstellung hat einige sehr wichtige Eigenschaften ([14, Theorem 3.29]):

- Die NAF einer natürlichen Zahl k ist eindeutig, sie wird mit NAF(k) bezeichnet
- Die Länge der NAF hat maximal eine Stelle mehr, als die binäre Darstellung (siehe [14, Theorem 3.33])
- Betrachtet man alle NAF-Darstellungen einer festen Länge l, so ist die durchschnittliche Dichte von Stellen, die ungleich Null sind, unter diesen NAF-Darstellungen ungefähr 1/3.

Algorithmus 32 zeigt nun, wie die NAF einer natürlichen Zahl effizient berechnet werden kann, in Algorithmus 33 wird gezeigt, wie man sie verwendet, um den Double and Add Algorithmus so zu modifiziren, dass weniger Additionen in der Gruppe benötigt werden, um das Vielfache eines Gruppenelementes zu berechnen.

Algorithmus 32 Berechnung der NAF einer natürlichen Zahl k

Eingabe: Eine natürliche Zahl k
Ausgabe: Die non-adjacent form von k, NAF(k)

```
i ← 0
while (k ≥ 1) do
    if (k ist ungerade) then
        k_i ← 2 − (k mod 4)
        k ← k − k_i
    else
        k_i = 0
    end if
    k ← k/2
    i ← i + 1
end while
return NAF(k) = (k_{i−1}, k_{i−2}, ..., k_1, k_0)
```

Da der durchschnittliche Anteil von Stellen, die ungleich Null sind, in der NAF von k nur 1/3 beträgt, ist die Anzahl der Additionen in der Regel deutlich geringer, als bei der Verwendung von Algorithmus 30 oder 31, wo dieser Anteil bei 1/2 liegt. Um diese Reduktion zu erreichen, muss allerdings sowohl die NAF von k, als auch das Element $-P$ zuvor berechnet werden, dieser Mehraufwand lohnt sich erst für große k.
Die Idee, den Double and Add Algorithmus dadurch zu beschleunigen, dass man für den

Algorithmus 33 Punktmultiplikation mit der NAF

Eingabe: $k \in \mathbb{N}, P$ Erzeuger einer zyklischen additiven Gruppe
Ausgabe: $k \cdot P$

```
Berechne NAF(k) = Σ_{i=0}^{l-1} k_i 2^i
Q ← O
for (i = l − 1, i ≥ 0, i − −) do
    Q ← 2Q
    if (k_i = 1) then
        Q ← Q + P
    end if
    if (k_i = −1) then
        Q ← Q − P
    end if
end for
return Q
```

Skalar eine Darstellung wählt, die möglichst viele Nullen enthält, wird durch die sog. *w-NAF* weitergeführt:

Definition 3.13. [14, Definition 3.32] Sei $w \geq 2 \in \mathbb{N}$. Die *w-NAF* einer natürlichen Zahl k ist ein Ausdruck der Form

$$k = \sum_{i=0}^{l-1} k_i 2^i,$$

wobei alle k_i entweder Null oder ungerade sind und $|k_i| < 2^{w-1}$, $k_{l-1} \neq 0$ und von jeweils w aufeinanderfolgenden Stellen ist maximal eine ungleich Null. Die Länge der w-NAF ist l. Die Zahl w ist die Fensterbreite der w-NAF.

Auch die w-NAF-Darstellung hat einige wichtige Eigenschaften (siehe [14, Theorem 3.33]):

- Jede natürliche Zahl k hat für jedes $w \in \mathbb{N}$ eine eindeutige w-NAF Darstellung, sie wird mit $\mathrm{NAF}_w(k)$ bezeichnet.
- $\mathrm{NAF}_2(k) = \mathrm{NAF}(k)$
- Die w-NAF einer natürlichen Zahl k ist um maximal eine Stelle länger als ihre Binärdarstellung.
- Betrachtet man alle w-NAF-Darstellungen einer festen Länge l, so ist die durchschnittliche Dichte von Stellen, die ungleich Null sind, unter diesen w-NAF-Darstellungen ungefähr $1/(w+1)$.

Die Ideen, die schon bei der NAF vorgestellt wurden, werden nun auf die w-NAF verallgemeinert, hierzu stellt Algorithmus 34 vor, wie sich die w-NAF effizient berechnen lässt, in Algorithmus 35 ist beschrieben, wie man diese Darstellung zur effizienten Vervielfachung von Gruppenelementen verwendet:

Algorithmus 34 Berechnung der w-NAF einer natürlichen Zahl k

Eingabe: Eine natürliche Zahl k, Fensterbreite w
Ausgabe: Die w-NAF von k, $\mathrm{NAF}_w(k)$

```
i ← 0
while (k ≥ 1) do
    if (k ist ungerade) then
        k_i ← k mod 2^w
        if (k_i > 2^(w−1)) then
            k_i ← k_i − 2^w
        end if
        k ← k − k_i
    else
        k_i = 0
    end if
    k ← k/2
    i ← i + 1
end while
return NAF_w(k) = (k_{i−1}, k_{i−2}, ..., k_1, k_0)
```

Algorithmus 35 Punktmultiplikation mit der w-NAF

Eingabe: $k \in \mathbb{N}, P$ Erzeuger einer zyklischen additiven Gruppe und $w \in \mathbb{N}$
Ausgabe: $k \cdot P$

```
Berechne NAF_w(k) = ∑_{i=0}^{l−1} k_i 2^i
Berechne P_i = i · P ∀ i ∈ {1, 3, 5, ... 2^(w−1) − 1}
Q ← O
for (i = l − 1, i ≥ 0, i − −) do
    Q ← 2Q
    if (k_i ≠ 0) then
        if (k_i > 0) then
            Q ← Q + P_{k_i}
        end if
        if (k_i < 0) then
            Q ← Q − P_{−k_i}
        end if
    end if
end for
return Q
```

Dadurch, dass der durchschnittliche Anteil von Stellen, die ungleich Null sind, in der w-NAF Darstellung nur $1/(w+1)$ beträgt, kann die Anzahl der Additionen also weiter reduziert werden, dies geschieht allerdings auf Kosten einiger Vorberechnungen und damit auch auf Kosten erhöhten Speicherbedarfs.
Weitere Verfahren beschleunigen die Berechnung des Vielfachen eines Gruppenelementes einer zyklischen additiven Gruppe weiter, einige davon basieren auf speziellen Eigenschaften der Gruppe, so finden sich etwa für Gruppen von Punktemengen elliptischer Kurven über Körpern der Charakteristik 2 die sog. Montgomery Punktmultiplikation (siehe [14, Algorithmus 3.40]) oder auch Verfahren, die auf der Halbierung von Elementen basieren (siehe [14, Abschnitt 3.6]).
Andere Verfahren basieren darauf, dass der Erzeuger P der additiven Gruppe schon zur Zeit der Erstellung der entsprechenden Implementierung bekannt ist (siehe [14, Abschnitt 3.3.2]) oder, dass effizient berechenbare Kurvenendomorphismen existieren, die geeignet sind, die Berechnung zu beschleunigen (siehe [14, Abschnitt 3.5]).

4 Hyperelliptische Kurven

4.1 Mathematischer Hintergrund

Auch wenn etwa in [9, Abschnitt 4.4.2b] begründet wird, warum elliptische Kurven kein Spezialfall hyperelliptischer Kurven sind, sind diese beiden Strukturen so sehr verwandt, dass man an anderen Stellen in der Literatur auch die Formulierung findet, dass elliptische Kurven gerade hyperelliptische Kurven vom Geschlecht 1 sind [25].
Auch basierend auf hyperelliptischen Kurven lassen sich endliche abelsche Gruppen definieren, Abbildung 4.1 zeigt allerdings, dass sich auf der Menge der Punkte einer hyperelliptischen Kurve nicht wie bei elliptischen Kurven durch die Sehnen- und Tangenten-Methode eine Gruppenoperation definieren lässt. Dieser Abschnitt soll nun den mathematischen Hintergrund darstellen, der die Grundlage von auf hyperelliptischen Kurven basierenden endlichen abelschen Gruppen bildet. Er basiert weitestgehend auf [25].

4.1.1 Hyperelliptische Kurven

Definition 4.1. [25] Sei K ein Körper, $\overline{K}$ der algebraische Abschluss von K. Sei weiter $2 \leq g \in \mathbb{N}, f(x), h(x) \in K[x]$ mit $\deg(f) = 2g+1$ und $\deg(h) \leq g$. Dann definiert eine Gleichung der Form

$$C : y^2 + h(x)y = f(x) \quad \in K[x,y] \tag{4.1}$$

eine *hyperelliptische Kurve vom Geschlecht g*, wenn es keinen Punkt $(u,v) \in \overline{K} \times \overline{K}$ gibt, der gleichzeitig die Gleichung C sowie beide partiellen Ableitungen $2y + h(x) = 0$ und $h'(x)y - f'(x) = 0$ erfüllt.

Definition 4.2. [25] Sei $L \subseteq \overline{K}$ ein Erweiterungskörper von K. Dann heißt die Menge

$$C(L) = \{(u,v) \in L \times L \mid v^2 + h(u)v = f(u)\} \cup \{\mathcal{O}\}$$

die *Menge L-rationaler Punkte auf C*.
Das Element $\mathcal{O} \in C(L)$ wird als *Punkt im Unendlichen* bezeichnet, alle anderen Punkte heißen endliche Punkte. Die Menge $\overline{K}$-rationaler Punkte auf C ist gerade C, also $C(\overline{K}) = C$

Definition 4.3. [25] Sei $P = (u,v)$ ein endlicher Punkt auf der hyperelliptischen Kurve C. Dann heißt $\tilde{P} = (u, -v-h(u))$ der P *gegenüberliegende Punkt*. Der $\mathcal{O}$ gegenüberliegende Punkt ist $\mathcal{O}$ selbst, also $\mathcal{O} = \tilde{\mathcal{O}}$. Gilt für einen Punkt P, dass $P = \tilde{P}$, dann nennt man P einen *speziellen Punkt*, sonst ist P ein *gewöhnlicher Punkt*.

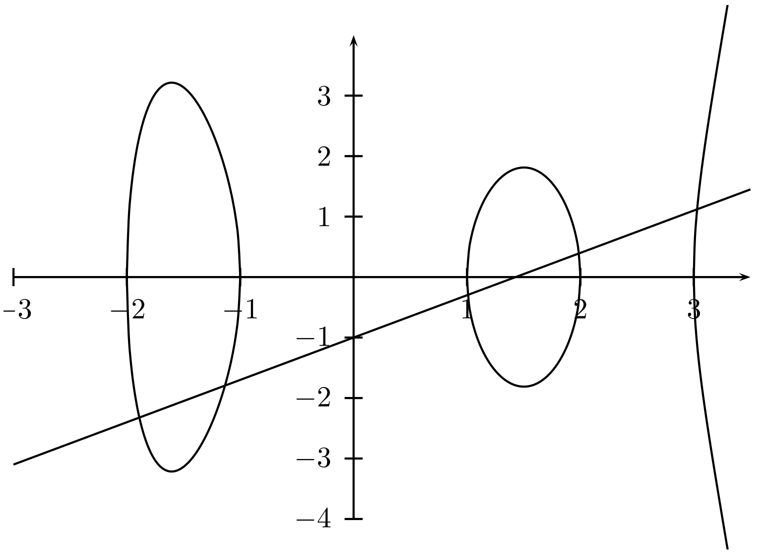

Abbildung 4.1: Schnitt einer Geraden mit einer hyperelliptischen Kurve über $\mathbb{R}$

4.1.2 Polynomfunktionen und rationale Funktionen

Definition 4.4. [25] Der *Koordinatenring von C über K* (Bezeichnung: $K[C]$) ist der Quotientenring

$$K[C] = K[x,y]/(y^2 + h(x)y - f(x)),$$

wobei $(y^2+h(x)y-f(x))$ das von dem Polynom $y^2+h(x)y-f(x)$ erzeugte Ideal bezeichnet. Analog definiert man den Koordinatenring von C über $\overline{K}$ als

$$\overline{K}[C] = \overline{K}[x,y]/(y^2 + h(x)y - f(x)),$$

Die Elemente von $\overline{K}(C)$ bezeichnet man als Polynomfunktionen auf C.

Um im weiteren Verlauf den Quotientenkörper der Polynomfunktionen zu konstruieren, benötigen wir das folgende Lemma:

Lemma 4.5. *[25] $K[C]$ und $\overline{K}[C]$ sind Integritätsbereiche.*

Beweis. Da $K[x,y]$ bzw. $\overline{K}[x,y]$ faktorielle Ringe sind, genügt es zu zeigen, dass $y^2 + h(x)y - f(x)$ über $\overline{K}[x,y]$ irreduzibel ist.
Wäre das Polynom $y^2+h(x)y-f(x)$ nicht irreduzibel, würde es zerfallen in $(y-a(x))(y-b(x))$. Dann müsste $f(x)$ das Produkt aus $a(x)$ und $b(x)$ und $h(x)$ die Summe aus $a(x)$ und $b(x)$ sein. Da f ungeraden Grad hat, können $a(x)$ und $b(x)$ nicht vom gleichen Grad sein. Also folgt aus $\deg(h(x)) \leq g$, dass der Grad von $a(x)$ und von $b(x)$ auch kleiner oder gleich g ist. Dann kann das Produkt nicht von Grad $2g+1 = \deg(f(x))$ sein. Dies ist ein Widerspruch, also ist $y^2 + h(x)y - f(x)$ irreduzibel über $\overline{K}[x,y]$ und insbesondere über $K[x,y]$. □

Durch sukzessives Ersetzen von y^2 durch $f(x) - h(x)y$ erhält man für jede Polynomfunktion G eine eindeutige Darstellung der Form $G = a(x) - b(x)y$, diese Darstellung wird im folgenden verwendet.
Häufig ist es hilfreich, Polynome in einer Variablen zu betrachten um Aussagen über Polynome in zwei Variablen zu gewinnen. Hierfür verwendet man die Norm einer Polynomfunktion, die mit den folgenden Definitionen und Lemmata eingeführt werden soll:

Definition 4.6. [25] Sei $G = a(x)-b(x)y$ eine Polynomfunktion aus $\overline{K}[C]$. *Die Konjugierte von* G ist die Polynomfunktion $\overline{G} = a(x) + b(x)(h(x) + y)$.

Definition 4.7. [25] Sei $G = a(x) - b(x)y$ eine Polynomfunktion aus $\overline{K}[C]$. Dann heißt $N(G) = G\overline{G}$ die *Norm von* G.

Lemma 4.8. *[25]*

1. $N(G) \in \overline{K}[x]$.
2. $N(\overline{G}) = N(G)$
3. $N(GH) = N(G)N(H)$

Beweis. Hier soll nur Teil 1 gezeigt werden, der Beweis für Teil 2 und 3 findet sich in [25]. Teil 1 sieht man wie folgt ein:

$$N(G) = G \cdot \overline{G} = (a - by)(a + b(h + y)) = a^2 + abh - b^2 \underbrace{(y^2 + hy)}_{=f} \in \overline{K}[x]$$

□

Definition 4.9. [25] Sei $G(x,y) = a(x)-b(x)y$ eine Polynomfunktion aus $\overline{K}[C]$ mit $G \neq 0$. Der *Grad von* G ist:

$$\deg(G) = \max\{2\deg_x(a),\ 2g + 1 + 2\deg_x(b)\} = \deg_x(N(G))$$

Den Beweis für die letzte Gleichung findet man in [25]. Es gilt, dass $\deg(GH) = \deg(G) + \deg(H)$, sowie, dass $\deg(G) = \deg(\overline{G})$.
Da $K[C]$ und $\overline{K}[C]$ Integritätsbereiche sind, kann man auch den Quotientenkörper von $K[C]$ sowie von $\overline{K}(C)$ betrachten. Dieser soll nun mit der folgenden Definition eingeführt werden:

Definition 4.10. [25] Der *Funktionenkörper* $K(C)$ *von* C *über* K ist der Quotientenkörper von $K[C]$. Genauso definiert man den Funktionenkörper $\overline{K}(C)$ als den Quotientenkörper von $\overline{K}[C]$. Die Elemente von $\overline{K}(C)$ nennt man rationale Funktionen.

Setzt man Punkte aus C in rationale Funktionen ein, so lassen sich Werte dieser Funktionen definieren. Hierzu benötigt man wiederum einige Definitionen und Aussagen:

Definition 4.11. [25] Sei $R \in \overline{K}(C)$ und sei $P \in C, P \neq \mathcal{O}$. Man sagt R *ist definiert in* P, wenn es $G, H \in \overline{K}[C]$ gibt mit $H(P) \neq 0$ und $R = \frac{G}{H}$, sonst ist R nicht definiert in P. Ist R definiert in P, so ist der Wert von R in P definiert als $R(P) = \frac{G(P)}{H(P)}$.

Man benötigt weiter die Definition des Wertes einer rationalen Funktion in $\mathcal{O}$:

Definition 4.12. [25] Sei $R = \frac{G}{H} \in \overline{K}(C)$ eine rationale Funktion. Ist $\deg(G) < \deg(H)$, so ist $R(\mathcal{O}) = 0$, ist $\deg(G) > \deg(H)$, so ist $R(\mathcal{O}) = \infty$. Gilt $\deg(G) = \deg(H)$, so berechnet sich der Wert von R in $\mathcal{O}$ als der Quotient der führenden Koeffizienten von G und H. Der führende Koeffizient ist hierbei der Koeffizient der höchsten Potenz von x in $a(x)$, wenn $2\deg_x(a) > 2g + 1 + \deg_x(b)$ bzw. der Koeffizient der höchsten Potenz von x in $b(x)$ sonst (siehe auch Definition 4.9).

4.1.3 Nullstellen und Pole

In diesem Abschnitt werden Nullstellen von rationalen Funktionen betrachtet. Die hier aufgeführten Definitionen und Sätze werden schließlich zusammengesetzt, um die Ordnung einer rationalen Funktion in einem Punkt zu definieren, die sich als Nullstellenvielfachheit interpretieren lässt. Eine deutlich ausführlichere Darstellung der hier dargestellten Zusammenhänge findet sich in [30, Kapitel 1].

Definition 4.13. [25] Sei $R \in \overline{K}(C), R \neq 0$ und sei $P \in C$. Ist $R(P) = 0$, so hat R eine *Nullstelle* in P, ist R in P nicht definiert, so hat R eine *Polstelle* in P, man schreibt dann $R(P) = \infty$.

Lemma 4.14. *[28, II, Prop 1.1],[30, I. Prop. 1.5(a)] Betrachtet man nun einen Punkt $P \in C$. Dann bildet die Menge $O_P = \{R \in \overline{K}(C) \mid R \text{ ist in } P \text{ definiert } \}$ einen Teilring von $\overline{K}(C)$.*
Dieser Ring O_P ist ein diskreter Bewertungsring von $\overline{K}(C)$ mit maximalem Ideal $M_P = \{R \in O_P \mid R(P) = 0\}$.

Sei nun $U \in M_P$ ein Primelement, also ein Erzeuger des maximalen Ideals von O_P. Dann lässt sich nach der Definition eines diskreten Bewertungsringes jedes $G \neq 0 \in \overline{K}(C)$ schreiben als $G = U^d \cdot S$ mit $S \in O_P \setminus M_P$. Mit Hilfe dieser Darstellung kann man nun die diskrete Bewertung von $K(C)$ definieren:

Definition 4.15. [30, I. Definition 1.11] Sei $P \in C$ und U Primelement von M_P. Die Abbildung

$$\begin{aligned}\nu_P : \overline{K}(C) \to \mathbb{Z} \cup \{\infty\} : U^d \cdot S &\mapsto d,\\ 0 &\mapsto \infty\end{aligned}$$

heißt *Bewertung des Funktionenkörpers* $K(C)$. Diese Abbildung ist unabhängig von der Wahl von U.

Dass ν_P tatsächlich die Eigenschaften einer diskreten Bewertungsfunktion hat, wird im Beweis von [30, Theorem I.1.12] gezeigt.
Nun kann man die Ordnung einer rationalen Funktion G in einem Punkt P wie folgt definieren:

Definition 4.16. Sei $R \in \overline{K}(C)$ eine rationale Funktion, $R \neq 0$ und sei $P \in C$. Die *Ordnung von R in P* ist:

$$\mathrm{ord}_P(R) = \nu_P(R)$$

In [28, II. Prop. 1.2] und [28, II. Prop. 3.1 (b)] findet man die folgenden beiden wichtigen Sätze:

Satz 4.17. *Sei $R \in \overline{K}(C)$. Dann existieren nur endlich viele $P \in C$ mit $R(P) = 0$ oder $R(P) = \infty$. G hat also nur endlich viele Pole und Nullstellen.*

Satz 4.18. *Sei $R \in \overline{K}(C)^*$. Dann gilt:*

$$\sum_{P \in C} \mathrm{ord}_P(R) = 0$$

Bei der Betrachtung der Ordnung von Polynomfunktionen im Punkt $\mathcal{O}$ hilft folgendes Lemma:

Lemma 4.19. *Sei G eine Polynomfunktion, dann ist* $\mathrm{ord}_{\mathcal{O}}(G) = -\deg(G)$.

Beweis. Es ist:

$$\begin{aligned} O_{\mathcal{O}} &= \{R = G_R/H_R \in \overline{K}(C) \mid G_R, H_R \in \overline{K}[C] \text{ und } \deg(H_R) \geq \deg(G_R)\} \text{ und} \\ M_{\mathcal{O}} &= \{R = G_R/H_R \in \overline{K}(C) \mid G_R, H_R \in \overline{K}[C] \text{ und } \deg(H_R) > \deg(G_R)\} \end{aligned}$$

Ein Erzeuger U des maximalen Ideals $M_{\mathcal{O}}$ muss nun also die Form $U = G_U/H_U$ haben, mit $G_U, H_U \in \overline{K}[C]$ und $\deg(H_U) = \deg(G_U) + 1$, da sonst durch Multiplikation mit Elementen aus $O_{\mathcal{O}}$ keine Elemente $W = G_W/H_W$ erzeugt werden könnten, für die gilt, dass $G_W, H_W \in \overline{K}[C]$ und $\deg(H_W) = \deg(G_W) + 1$. Solche Elemente W sind allerdings in $M_{\mathcal{O}}$ enthalten.
Sei nun $G = U^d \cdot S$ eine Zerlegung der Polynomfunktion G mit $S \in O_{\mathcal{O}} \setminus M_{\mathcal{O}}$. Dann hat S die Form $S = G_S/H_S$ mit $G_S, H_S \in \overline{K}[C]$ und $\deg(G_S) = \deg(H_S)$ (siehe hierzu Definition 4.12). Da G eine Polynomfunktion ist, muss also, damit $\deg(G) = \deg(U^d \cdot S)$ ist, d gerade $-\deg(G)$ sein, also ist nach Definition 4.16 $\mathrm{ord}_{\mathcal{O}}(G) = -\deg(G)$ □

In [25] werden explizit Erzeuger von M_P für beliebiges P hergeleitet. Für gewöhnliche Punkte $P = (x_1, y_1) \in C$ ist dies $U = (x - x_1)$, für endliche spezielle Punkte (x_1, y_1) ist es $U = (y - y_1)$ und für den Punkt $\mathcal{O}$ stellt (x^g/y) ein Primelement dar.

4.1.4 Divisoren

Dieser Abschnitt setzt die Ergebnisse aus den vorigen Abschnitten zusammen um eine Struktur zu definieren, die schließlich zu der endlichen abelschen Gruppe führen wird, die für die kryptographischen Anwendungen verwendet wird. Diese Struktur ist die Gruppe der Divisoren einer hyperelliptischen Kurve C.

Definition 4.20. [28] Die Gruppe der *Divisoren* einer hyperelliptischen Kurve C ist die von den Punkten von C erzeugte freie abelsche Gruppe, sie wird mit $\mathbb{D}$ bezeichnet. Die Elemente dieser Gruppe sind also formale Summen der Form:

$$D = \sum_{P \in C} n_P P$$

mit $n_P \in \mathbb{Z}$ und nur endlich viele n_P sind ungleich 0.

Weiter definiert man den *Grad eines Divisors* als die Summe $\deg(D) = \sum_{P \in C} n_P$ und die *Ordnung eines Divisors D in einem Punkt P* als $\mathrm{ord}_P(D) = n_P$.
Die Menge aller Divisoren mit Grad 0 bildet eine Untergruppe von $\mathbb{D}$ und wird mit $\mathbb{D}^0$ bezeichnet.

Definition 4.21. [25] Seien $D_1 = \sum_{P \in C} m_P P$ und $D_2 = \sum_{P \in C} n_P P$ aus $\mathbb{D}$. Der *größte gemeinsame Divisor* von D_1 und D_2 ist:

$$\gcd(D_1, D_2) = \sum_{P \in C} \min(m_P, n_P) P - \left(\sum_{P \in C} \min(m_P, n_P) \right) \mathcal{O}$$

Nun betrachtet man die von rationalen Funktionen erzeugten Divisoren:

Definition 4.22. [25] Sei $R \in \overline{K}(C),\ R \neq 0$. Der *Divisor von R* ist

$$\mathrm{div}(R) = \sum_{P \in C} (\mathrm{ord}_P(R)) P$$

Nach Satz 4.18 ist der Grad eines solchen Divisors 0.

Definition 4.23. [25], [28, II Prop. 3.1] Ein Divisor $D \in \mathbb{D}^0$ heißt *Hauptdivisor*, wenn es eine rationale Funktion $R \in \overline{K}(C)^*$ gibt mit $D = \mathrm{div}(R)$. Die Menge aller Hauptdivisoren ist eine Untergruppe von $\mathbb{D}^0$, sie wird mit $\mathbb{P}$ bezeichnet. Die Quotientengruppe $\mathbb{J} = \mathbb{D}^0 / \mathbb{P}$ heißt Jacobische Varietät der Kurve C.
$D_1, D_2 \in \mathbb{D}^0$ heißen äquivalent ($D_1 \sim D_2$), wenn $D_1 - D_2 \in \mathbb{P}$.

Beispiel 4.24. [25] Sei $P = (u, v) \in C$ ein gewöhnlicher Punkt auf C. Dann ist $\mathrm{div}(x - u) = P + \tilde{P} - 2\mathcal{O}$.
Ist $P = (u, v) \in C$ ein spezieller endlicher Punkt, so ist $\mathrm{div}(x - u) = 2P - 2\mathcal{O}$.

4.1.5 Darstellung von Divisorklassen

Um in der Jacobischen Varietät einer hyperelliptischen Kurve zu rechnen, benötigt man eine effiziente Darstellung der Divisorklassen, die die Elemente von $\mathbb{J}$ bilden. Hierzu wählt man aus jeder Klasse einen Repräsentanten, der einige Eigenschaften erfüllt, die ihn besonders günstig für die Berechnung der Gruppenoperation machen. Diese Eigenschaften, sowie der Nachweis, dass der entsprechende Divisor einer Divisorklasse durch sie eindeutig definiert ist, zeigt dieser Abschnitt.
Ein erster Schritt auf diesem Weg sind semi-reduzierte Divisoren, die in den beiden folgenden Definitionen vorgestellt werden:

Definition 4.25. [25] Sei $D = \sum_{P \in C} m_P P$ ein Divisor. Der *Träger von D* ist die Menge $\mathrm{supp}(D) = \{P \in C \mid m_P \neq 0\}$

Definition 4.26. [25] Ein Divisor D der Form

$$D = \sum_{P \in C \setminus \{\mathcal{O}\}} m_P P - \left(\sum_{P \in C \setminus \{\mathcal{O}\}} m_P \right) \mathcal{O}$$

heißt *semireduziert*, wenn $m_P \geq 0 \ \forall P \in C \setminus \{\mathcal{O}\}$, wenn für alle gewöhnlichen Punkte P gilt, dass: $P \in \mathrm{supp}(D) \Rightarrow \tilde{P} \notin \mathrm{supp}(D)$ und für alle endlichen speziellen Punkte Q gilt, dass: $Q \in \mathrm{supp}(D) \Rightarrow m_Q = 1$.

Lemma 4.27. *[25] Für jedes $D \in \mathbb{D}^0$ gibt es einen semireduzierten Divisor $D' \in \mathbb{D}^0$ mit $D \sim D'$.*

Beweis. Sei $D = \sum_{P \in C} m_P P$ und sei (C_1, C_2) eine Partition der Menge der gewöhnlichen Punkte von C, so dass

- $P \in C_1 \Rightarrow \tilde{P} \in C_2$
- $P \in C_1 \Rightarrow m_P \geq m_{\tilde{P}}$

Sei C_0 die Menge der endlichen speziellen Punkte auf C. Dann lässt sich D schreiben als:

$$D = \sum_{P \in C_1} m_P P + \sum_{P \in C_2} m_P P + \sum_{P \in C_0} m_P P - m\mathcal{O}$$

mit m so, dass $\sum_{P \in C \setminus \{\mathcal{O}\}} m_P = m$.
Setze nun

$$D' = D - \sum_{P=(u,v) \in C_2} m_P \mathrm{div}(x-u) - \sum_{P=(u,v) \in C_0} \lfloor \frac{m_P}{2} \rfloor \mathrm{div}(x-u)$$

Offensichtlich ist nun $D - D' \in \mathbb{P}$, es gilt also $D \sim D'$, weiterhin gilt mit Beispiel 4.24:

$$\begin{aligned} D' &= \sum_{P \in C_1} m_P P + \sum_{P \in C_2} m_P P - \sum_{P=(u,v) \in C_2} m_P \cdot (P + \tilde{P} - 2\mathcal{O}) \\ &\quad + \sum_{P \in C_0} m_P P - \sum_{P \in C_0} \lfloor \frac{m_P}{2} \rfloor (2P - 2\mathcal{O}) - m\mathcal{O} \\ &= \sum_{P \in C_1} m_P P - \sum_{P \in C_2} m_P \tilde{P} + \sum_{P \in C_0} (m_P - 2\lfloor \frac{m_P}{2} \rfloor) P - m'\mathcal{O} \\ &= \sum_{P \in C_1} (m_P - m_{\tilde{P}}) P - \sum_{P \in C_0} \left(m_P - 2\lfloor \frac{m_P}{2} \rfloor \right) P - m'\mathcal{O} \end{aligned}$$

für ein geeignetes m'. Also ist D' semireduziert. □

Schon semireduzierte Divisoren besitzen eine effizient speicherbare Darstellung, die sog. Mumford-Darstellung:

Satz 4.28. *Mumford Darstellung [25]*
Sei $D = \sum m_i P_i - (\sum m_i)\mathcal{O}$ *ein semireduzierter Divisor und* $P_i = (x_i, y_i)$. *Sei* $a(x) = \prod (x - x_i)^{m_i}$. *Dann existiert genau ein Polynom* $b(x) \in \overline{K}[x]$ *mit den folgenden Eigenschaften:*

- $\deg(b(x)) < \deg(a(x))$
- $b(x_i) = y_i \quad \forall (x_i, y_i) \in \operatorname{supp}(D)$
- $a(x) \mid b(x)^2 + b(x)h(x) - f(x)$

Dann gilt: $D = \gcd(\operatorname{div}(a(x)), \operatorname{div}(b(x) - y))$
Als Kurzschreibweise verwendet man häufig $D = \operatorname{div}(a, b)$, *diese Darstellung eines Divisors durch zwei Polynome aus* $\overline{K}[x]$ *heißt Mumford-Darstellung.*

Beweis. siehe [25] □

Lemma 4.29. *[25] Seien* $a(x), b(x) \in \overline{K}[x]$ *mit* $\deg(a(x)) > \deg(b(x))$ *und* $a(x) \mid b(x)^2 + b(x)h(x) - f(x)$. *Dann ist* $\operatorname{div}(a, b)$ *semireduziert.*

Offensichtlich ist, dass die Darstellung eines Divisors durch zwei Polynome recht einfach ist, das Problem, was noch besteht, ist, dass Satz 4.28 keinerlei Aussagen über den Grad der Polynome $a(x)$ und $b(x)$ macht.
Im folgenden werden reduzierte Divisoren eingeführt, für die der Grad der Polynome $a(x)$ und $b(x)$ durch das Geschlecht der Kurve begrenzt ist. Außerdem haben sie die Eigenschaft, dass es in jeder Divisorklasse der Jacobischen Varietät genau einen reduzierten Divisor gibt.

Definition 4.30. [25] Sei $D = \sum_{P \in C} m_P P$ ein Divisor. Die *Norm von D* ist definiert als:

$$|D| = \sum_{P \in C \setminus \{\mathcal{O}\}} |m_P|$$

Definition 4.31. [25] Sei D ein semi-reduzierter Divisor. Wenn $|D| \leq g$, dann heißt D reduziert.

Um die Existenz eines reduzierten Divisors in jeder Divisorklasse von $\mathbb{J}$ zu beweisen, benötigt man ein vorbereitendes Lemma:

Lemma 4.32. *Sei* $G \in \overline{K}[C]$ *eine Polynomfunktion. Dann gilt:*

$$\operatorname{ord}_P(G) \geq 0 \ \forall P \in C \setminus \mathcal{O}$$

.
Weiter gilt: $|\operatorname{div}(G)| = \deg(G)$.

Beweis. Der erste Teil der Aussage folgt direkt aus Definition 4.15 und Definition 4.16, der zweite Teil folgt aus dem ersten zusammen mit Lemma 4.19 und der Tatsache, dass $\deg(\mathrm{div}(G)) = 0$. □

Satz 4.33. *[25] Sei $D \in \mathbb{D}^0$. Dann existiert genau ein reduzierter Divisor $D_1 \in \mathbb{D}^0$ mit $D_1 \sim D$.*

Beweis. Hier soll nur die Existenz eines solchen reduzierten Divisors D_1 gezeigt werden, den vollständigen Beweis findet man in [25].
Sei also $D \in \mathbb{D}^0$ und D' ein semi-reduzierter Divisor mit $D \sim D'$ mit $|D'| \leq |D|$ (dieser existiert nach dem Beweis von 4.27). Wenn $|D'| \leq g$, so ist D' schon reduziert.
Ansonsten betrachte die Punkte $P_1, \ldots, P_{g+1} \in \mathrm{supp}(D')$, wobei jeder Punkt P in dieser Liste nicht häufiger vorkommt, als $\mathrm{ord}_P(D')$ mal. Sei weiter $\mathrm{div}(a(x), b(x))$ die Mumford-Darstellung des Divisors $P_1 + P_2 + \cdots + P_{g+1} - (g+1)\mathcal{O}$.
Da der Grad von $b(x)$ kleiner ist als g, gilt, dass $\deg(b(x) - y) = 2g + 1$ und also:

$$\mathrm{div}(b(x) - y) = P_1 + P_2 + \cdots + P_{g+1} + Q_1 + \cdots + Q_g - (2g+1)\mathcal{O}$$

für endliche Punkte $Q_1 \ldots Q_g$. Subtrahiert man nun diesen Divisor von D', so erhält man einen Divisor D'', für den gilt, dass: $D'' \sim D' \sim D$ und $|D''| < |D'|$. Aus diesem Divisor D'' konstruiert man nach dem Verfahren aus dem Beweis von 4.27 einen semi-reduzierten Divisor D''', für den gilt $|D'''| \leq |D''|$. Dieses Verfahren iteriert man, bis man einen reduzierten Divisor $D^{(n)}$ hat, für den dann gilt, dass $D^{(n)} \sim D$. □

Die bisherigen Betrachtungen bezogen sich immer auf die unendliche Gruppe $\mathbb{J}$, für kryptographische Anwendungen benötigt man allerdings eine endliche Gruppe (siehe Definition 1.1). Die folgenden Aussagen stammen aus [25] und führen zu einer Konstruktion eben solch einer endlichen Gruppe. Hierzu betrachten wir hyperelliptische Kurven über *endlichen* Körpern K.

Lemma 4.34. *[25] Sei C eine hyperelliptische Kurve über K und sei σ ein Automorphismus von $\overline{K}$ über K. Sei $P = (u, v)$ ein Punkt auf C, dann ist auch $P^\sigma = (\sigma(u), \sigma(v))$ ein Punkt auf C.*

Beweis. Es gilt $\sigma(0) = 0$, also $\sigma((v^2) + h(u)v - f(u)) = 0$. Da σ ein Automorphismus ist folgt:

$$(\sigma(v))^2 + h(\sigma(u))\sigma(v) - f(\sigma(u)) = 0$$

□

Definition 4.35. Ein Divisor $D = \sum_{P \in C} m_P P$ heißt definiert über K, wenn für alle Automorphismen σ von $\overline{K}$ über K gilt, dass:

$$D^\sigma = \sum_{P \in C} m_P P^\sigma = D.$$

.

Die Menge $\mathbb{J}(K)$ der Divisorklassen, die einen Divisor enthalten, der in K definiert ist, bildet eine Untergruppe von $\mathbb{J}$, bei einem endlichen Körper K ist sie gerade die gewünschte endliche Gruppe. Jedes Element aus $\mathbb{J}(K)$ enthält genau einen reduzierten Divisor $D_{red} = \mathrm{div}(a, b)$, wobei a und b Polynome aus $K[x]$ sind.

4.2 Implementierung hyperelliptischer Kurven

Nun ist die Struktur der über hyperelliptische Kurven definierten endlichen Gruppe bekannt. Die Darstellung der Gruppenelemente über zwei Polynome vom Grad maximal g bzw. $g-1$ mit Koeffizienten aus einem endlichen Körper K wurde ebenfalls eingeführt. Nun stellt sich die Frage, wie die Gruppenoperation effizient berechnet werden kann.
Dieser Abschnitt geht zunächst auf einen Algorithmus ein, der Elemente aus $\mathbb{J}(K)$ für beliebige hyperelliptische Kurven addiert. Anschließend soll vorgestellt werden, wie man wesentlich effizienter in $\mathbb{J}(K)$ rechnen kann, wenn man zusätzliche Anforderungen an die zugrunde liegende Kurve C stellt.

4.2.1 Cantors Algorithmus

Cantors Algorithmus ist ein Verfahren zum Addieren von Divisorklassen beliebiger hyperelliptischer Kurven. Es arbeitet in zwei Schritten: Der erste Schritt erwartet als Eingabe zwei semi-reduzierte Divisoren D_1 und D_2 in Mumford-Darstellung und liefert als Ausgabe einen semi-reduzierten Divisor D_3 in Mumford-Darstellung, so dass $D_3 = D_1 + D_2$. Algorithmus 36 beschreibt diesen ersten Schritt.
Im zweiten Schritt wird der semi-reduzierte Divisor, der im ersten berechnet wurde, reduziert, so dass man beide Schritte zusammen verwenden kann, um Divisorklassen mittels ihrer reduzierten Divisoren zu addieren. Der zweite Schritt wird in Algorithmus 37 dargestellt.
Die beiden Teile von Cantors Algorithmus stammen aus [25], hier findet sich auch ein Beweis für ihre Korrektheit.

Algorithmus 36 Addition zweier semi-reduzierter Divisoren in Mumford-Darstellung

Eingabe: Zwei semi-reduzierte Divisoren $D_1 = \mathrm{div}(a_1, b_1)$ und $D_2 = \mathrm{div}(a_2, b_2)$
Ausgabe: Der semi-reduzierte Divisor D_3, so dass $D = D_1 + D_2$

$d_1 \leftarrow \gcd(a_1, a_2) \qquad [d_1 = e_1 a_1 + e_2 a_2]$
$d \leftarrow \gcd(d_1, b_1 + b_2 + h) \qquad [d = c_1 d_1 + c_2(b_1 + b_2 + h)]$
$s_1 \leftarrow c_1 e_1$
$s_2 \leftarrow c_1 e_2$
$s_3 \leftarrow c_2$
$a \leftarrow \frac{a_1 a_2}{d^2}$
$b \leftarrow \frac{s_1 a_1 b_2 + s_2 a_2 b_1 + s_3(b_1 b_2 + f)}{d} \mod a$
return $D_3 = \mathrm{div}(a, b)$

4.2.2 Kurven vom Geschlecht 2 über Körpern ungerader Charakteristik

Für kryptographische Anwendungen kommen in der Regel nur hyperelliptische Kurven vom Geschlecht 2 oder 3 zum Einsatz, da für größeres Geschlecht Index-Calculus-Attacken gegen das DLP existieren, die schneller sind, als Pollard's ρ-Algorithmus. In [12] wird gezeigt, dass dies sogar schon für Geschlecht 3 der Fall ist. Dieser Abschnitt stellt nun

Algorithmus 37 Reduktion eines semi-reduzierten Divisors in Mumford-Darstellung

Eingabe: Einen semi-reduzierten Divisor $D = \mathrm{div}(a, b)$
Ausgabe: Der reduzierte Divisor D' mit $D \sim D'$
repeat
  $a' \leftarrow \frac{f-bh-b^2}{a}$
  $b' \leftarrow (-h - b) \mod a'$
  $a \leftarrow a'$
  $b \leftarrow b'$
until $\deg a \leq g$
Normiere a'
return $D' = \mathrm{div}(a', b')$

vor, wie hyperelliptische Kurven vom Geschlecht 2 über Körpern ungerader Charakteristik effizient implementiert werden können. Ein erster Schritt hierzu ist eine Vereinfachung der Kurvengleichung durch eine Variablentransformation: Sei C eine hyperelliptische Kurve vom Geschlecht 2 über einem Körper K mit $\mathrm{char}(K) \notin \{2, 5\}$ definiert durch:

$$C : y^2 + (h_2x^2 + h_1x + h_0)y = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0.$$

Durch die Variablentransformation $x \mapsto x - f_4/5$ erhält man $f_4 = 0$ und die Variablentransformation $y \mapsto y - (h_2x^2 + h_1x + h_0)/2$ führt zu einer Gleichung mit $h(x) = 0$. Im Folgenden gehen wir also von einer Kurvengleichung der Form

$$y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0 \tag{4.2}$$

aus. Ist wie in dieser Gleichung $h(x) = 0$, so ist es sehr einfach zu überprüfen, ob es einen Punkt $(u, v) \in \overline{K} \times \overline{K}$ gibt, der sowohl die Kurvengleichung, als auch beide partiellen Ableitungen erfüllt: Hierzu berechnet man den größten gemeinsamen Teiler von $f(x)$ und $f'(x)$, ist dieser nicht in K, so existiert ein solcher Punkt und die Gleichung $y^2 = f(x)$ definiert keine hyperelliptische Kurve.

Bei der Darstellung eines reduzierten Divisors $D = \mathrm{div}(a, b)$ beschränkt man sich nun auf den weitaus häufigsten Fall, dass der Grad von a gleich 2 ist, und gelangt so zu einer Darstellung über vier Elemente aus K wie folgt: $[a_1, a_0, b_1, b_0]$ repräsentiert den Divisor $D = \mathrm{div}(x^2 + a_1x + a_0, b_1x + b_0)$, diese Darstellung bezeichnet man auch als Darstellung in *affinen Koordinaten*

Nun verwendet man die Idee der projektiven Koordinaten von Punkten auf elliptischen Kurven um auch hier zu effizienteren Darstellungen zu gelangen. Ein erster Schritt hierzu ist die Darstellung eines reduzierten Divisors über ein Fünftupel $[a_1, a_0, b_1, b_0, Z]$, das den Divisor $D = \mathrm{div}(x^2 + a_1/Zx + a_0/Z, b_1/Zx + b_0/Z)$ repräsentiert. Diese Darstellung heißt aufgrund ihrer Ähnlichkeit mit den projektiven Koordinaten bei elliptischen Kurven auch *Darstellung in projektiven Koordinaten*.

Tanja Lange erweiterte diese Idee in [20], indem sie auch das Konzept der Gewichtung der projektiven Koordinaten von Punkten auf elliptischen Kurven in gewisser Weise auf die Darstellung von Divisoren übertrug. Dies führte zu sogenannten *neuen Koordinaten*, bei denen ein weiteres Element aus K zu der Darstellung hinzugefügt wird, so dass

man ein Sechstupel $[a_1, a_0, b_1, b_0, Z_1, Z_2]$ erhält, das den Divisor $D = \mathrm{div}(x^2 + a_1/Z_1^2 x + a_0/Z_1^2, b_1/(Z_1^3 Z_2)x + b_0/(Z_1^3 Z_2))$ darstellt. Häufig erweitert man auch diese Darstellung noch um zwei Einträge $z_1 = Z_1^2$ und $z_2 = Z_2^2$, da diese Quadrate von Z_1 und Z_2 in den Algorithmen zur Addition und Verdopplung ohnehin berechnet werden und deren Speicherung zwei Körperoperationen in den folgenden Berechnungen einspart.
Es lassen sich nun Algorithmen für die Addition und Verdopplung von Divisorklassen herleiten, die auf dieser Darstellung basieren, bei der Entwicklung dieser Algorithmen werden verschiedene Fälle unterschieden (siehe [9, Abschnitt 14.3.1]); für den weitaus häufigsten Fall finden sich in [20] effiziente Algorithmen. Die hier vorgestellten Algorithmen 38, 39 und 40 entsprechen diesen weitestgehend, an entsprechenden Stellen finden sich allerdings zusätzlich Abfragen, die sicherstellen, dass tatsächlich der Fall vorliegt, für den diese Algorithmen geeignet sind, und dass andernfalls mit einem anderen Verfahren (etwa Cantors Algorithmus, siehe Algorithmen 36 und 37) fortgefahren wird.
Bei der Implementierung ist weiterhin zu beachten, dass zwar für die meisten reduzierten Divisoren $D = \mathrm{div}(a, b)$ gilt, dass der Grad von a gleich 2 ist, dies muss allerdings vor der Verwendung der Algorithmen für beide Eingabedivisoren überprüft werden. Für Divisoren $D = \mathrm{div}(a, b)$ mit $\deg(a) < 2$ finden sich in [9, Abschnitt 14.3.2] spezielle Algorithmen, ich verwende in diesem seltenen Fall ebenfalls den generischen Algorithmus von Cantor.
Für die Addition zweier Divisoren in neuen Koordinaten benötigt Algorithmus 38 sieben Quadrierungen und 47 Multiplikationen in dem zugrundeliegenden endlichen Körper, liegt einer der Divisoren in affinen Koordinaten vor, so lässt sich die Summe mit Algorithmus 40 mit nur 5 Quadrierungen und 36 Multiplikationen berechnen. Für die Verdopplung eines Divisors in neuen Koordinaten benötigt Algorithmus 39 7 Quadrierungen und 34 Multiplikationen.
Die Anzahl der Körperoperationen ist also hier wesentlich höher als bei elliptischen Kurven (siehe Abschnitt 3.2), für die Implementierung von Kryptosystemen kann allerdings der endliche Körper deutlich kleiner gewählt werden, hierdurch lässt sich jede einzelne dieser Körperoperationen schneller berechnen.

Algorithmus 38 Addition zweier reduzierter Divisoren in neuen Koordinaten ($C : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0$)

Eingabe: Zwei Divisoren $D_1 = [a_{11}, a_{10}, b_{11}, b_{10}, Z_{11}, Z_{12}, z_{11}, z_{12}]$ und $D_2 = [a_{21}, a_{20}, b_{21}, b_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}]$ mit $D_1 \neq D_2$

Ausgabe: Ein Divisor $D_3 = [a_{31}, a_{30}, b_{31}, b_{30}, Z_{31}, Z_{32}, z_{31}, z_{32}] = D_1 + D_2$, falls ein solcher existiert

$z_{13} \leftarrow Z_{11}Z_{12},\ z_{23} \leftarrow Z_{21}Z_{22},\ z_{12} \leftarrow z_{11}z_{13},\ z_{22} \leftarrow z_{21}z_{23}$
$\tilde{a}_{21} \leftarrow a_{21}z_{11},\ \tilde{a}_{20} \leftarrow a_{20}z_{11},\ \tilde{b}_{21} \leftarrow b_{21}z_{12},\ \tilde{b}_{20} \leftarrow b_{20}z_{12}$
$inv_1 \leftarrow a_{11}z_{21} - \tilde{a}_{21},\ y_2 \leftarrow \tilde{a}_{20} - a_{10}z_{21},\ inv_0 \leftarrow a_{11}inv_1 + y_2z_{11}$
$r \leftarrow y_2inv_0 + {inv_1}^2a_{10}$
if $(r = 0)$ **then**
  Verwende einen anderen Algorithmus, um $D_1 + D_2$ zu berechnen.
end if
$Z_{32} \leftarrow Z_{11}Z_{21},\ \tilde{Z}_2 \leftarrow Z_{12}Z_{22},\ Z_1 \leftarrow {Z_{32}}^2,\ \tilde{Z}_2 \leftarrow \tilde{Z}_2Z_1,\ \tilde{Z}_2 \leftarrow \tilde{Z}_2r$
$Z_{32} \leftarrow Z_{32}\tilde{Z}_2,\ \tilde{Z}_2 \leftarrow \tilde{Z}_2^2,\ z_{32} \leftarrow Z_{32}^2$
$w_0 \leftarrow b_{10}z_{22} - \tilde{b}_{20},\ w_1 \leftarrow b_{11}z_{22} - \tilde{b}_{21},\ w_2 \leftarrow inv_0w_0,\ w_3 \leftarrow inv_1w_1$
$s_1 \leftarrow (inv_0 + z_{11}inv_1)(w_0 + w_1) - w_2 - w_3(z_{11} + a_{11})$
if $(s_1 = 0)$ **then**
  Verwende einen anderen Algorithmus, um $D_1 + D_2$ zu berechnen.
end if
$s_0 \leftarrow w_2 - a_{10}w_3$
$S_1 \leftarrow {s_1}^2,\ S_0 \leftarrow s_0Z_1,\ Z_{31} \leftarrow s_1Z_1,\ S \leftarrow Z_{31}S_0,\ S_0 \leftarrow {S_0}^2$
$R \leftarrow rZ_{31},\ s_0 \leftarrow s_0Z_{31},\ s_1 \leftarrow s_1Z_{31},\ z_{31} \leftarrow {Z_{31}}^2$
$l_2 \leftarrow s_1\tilde{a}_{21},\ l_0 \leftarrow s_0\tilde{a}_{20},\ l_1 \leftarrow (s_0 + s_1)(\tilde{a}_{20} + \tilde{a}_{21}) - l_0 - l_2$
$l_2 \leftarrow l_2 + S$
$b_{31} \leftarrow R\tilde{b}_{21}$
$a_{30} \leftarrow S_0 + inv_1(S_1(inv_1 + \tilde{a}_{21}) - 2s_0) + y_2s_1 + 2b_{31} + (2\tilde{a}_{21} + inv_1)\tilde{Z}_2$
$a_{31} \leftarrow 2S - inv_1s_1 - z_{32}$
$l_2 \leftarrow l_2 - a_{31},\ w_0 \leftarrow l_2a_{30},\ w_1 \leftarrow l_2a_{31}$
$b_{31} \leftarrow w_1 - z_{31}(l_1 + b_{31} - a_{30})$
$b_{30} \leftarrow w_0 - z_{31}(l_0 + R\tilde{b}_{20})$
return $D_3 = [a_{31}, a_{30}, b_{31}, b_{30}, Z_{31}, Z_{32}, z_{31}, z_{32}]$

Algorithmus 39 Verdopplung eines reduzierten Divisors in neuen Koordinaten ($C : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0$)

Eingabe: Ein Divisor $D_1 = [a_{11}, a_{10}, b_{11}, b_{10}, Z_{11}, Z_{12}, z_{11}, z_{12}]$
Ausgabe: Ein Divisor $D_3 = [a_{31}, a_{30}, b_{31}, b_{30}, Z_{31}, Z_{32}, z_{31}, z_{32}] = 2D_1$, falls ein solcher existiert

$\tilde{a}_0 \leftarrow a_{10}z_{11}$, $w_0 \leftarrow b_{11}^2$, $w_1 \leftarrow a_{11}^2$
$inv_0 \leftarrow b_{10}z_{11} - a_{11}b_{11}$, $r \leftarrow w_0a_{10} + b_{10}inv_0$
if $(r = 0)$ **then**
  Verwende einen anderen Algorithmus, um $D_1 + D_2$ zu berechnen.
end if
$\tilde{Z}_2 \leftarrow Z_{12}r$, $\tilde{Z}_2 \leftarrow \tilde{Z}_2z_{11}$, $Z_{32} = 2\tilde{Z}_2Z_{11}$, $\tilde{Z}_2 \leftarrow \tilde{Z}_2^2$
$inv_1 \leftarrow -b_{11}$
$z_3 \leftarrow z_{11}^2$, $w_3 \leftarrow f_3z_3 + w_1$
$k_1 \leftarrow z_{12}(2(w_1 - \tilde{a}_0) + w3)$, $z_3 \leftarrow z_3z_{11}$
$k_0 \leftarrow z_2(a_{11}(4\tilde{a}_0 - w_3) + z_3f_2) - w_0$
$w_0 \leftarrow k_0inv_0$, $w_1 \leftarrow k_1inv_1$
$s_1 \leftarrow (inv_0 + inv_1)(k_0 + k_1) - w_0 - w_1(1 + a_{11})$
if $(s_1 = 0)$ **then**
  Verwende einen anderen Algorithmus, um $D_1 + D_2$ zu berechnen.
end if
$s_0 \leftarrow w_0 - w_1\tilde{a}_0$
$S_0 \leftarrow {s_0}^2$, $Z_{31} \leftarrow s_1z_{11}$, $z_{31} \leftarrow {Z_{31}}^2$, $S \leftarrow s_0Z_{31}$
$R \leftarrow rZ_{31}$, $z_{32} \leftarrow {Z_{32}}^2$, $s_0 \leftarrow s_0s_1$, $s_1 \leftarrow Z_{31}s_1$
$l_2 \leftarrow s_1a_{11}$, $l_0 \leftarrow s_0a_{10}$, $l_1 \leftarrow (s_0 + s_1)(a_{10} + a_{11}) - l_0 - l_2$
$l_2 \leftarrow l_2 + S$
$b_{31} \leftarrow Rb_{11}$
$a_{30} \leftarrow S_0 + 4(b_{31} + 2\tilde{Z}_2a_{11})$
$a_{31} \leftarrow 2S - z_{32}$
$l_2 \leftarrow l_2 - a_{31}$, $w_0 \leftarrow l_2a_{30}$, $w_1 \leftarrow l_2a_{31}$
$b_{31} \leftarrow w_1 - z_{31}(l_1 + 2b_{31} - a_{30})$
$b_{30} \leftarrow w_0 - z_{31}(l_0 + 2Rb_{10})$
return $D_3 = [a_{31}, a_{30}, b_{31}, b_{30}, Z_{31}, Z_{32}, z_{31}, z_{32}]$

Algorithmus 40 Addition eines reduzierten Divisoren in neuen Koordinaten mit einem reduzierten Divisors in affinen Koordinaten ($C: y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0$)

Eingabe: Zwei Divisoren $D_1 = [a_{11}, a_{10}, b_{11}, b_{10}, 1, 1, 1, 1]$ und $D_2 = [a_{21}, a_{20}, b_{21}, b_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}]$ mit $D_1 \neq D_2$

Ausgabe: Ein Divisor $D_3 = [a_{31}, a_{30}, b_{31}, b_{30}, Z_{31}, Z_{32}, z_{31}, z_{32}] = D_1 + D_2$, falls ein solcher existiert

$z_{23} \leftarrow Z_{21}Z_{22}$, $z_{22} \leftarrow z_{21}z_{23}$, $inv_1 \leftarrow a_{11}z_{21} - a_{21}$
$y_2 \leftarrow a_{20} - a_{10}z_{21}$, $inv_0 \leftarrow a_{11}inv_1 + y_2$, $r \leftarrow y_2inv_0 + {inv_1}^2a_{10}$
if $(r = 0)$ **then**
  Verwende einen anderen Algorithmus, um $D_1 + D_2$ zu berechnen.
end if
$w_0 \leftarrow b_{10}z_{22} - b_{20}$, $w_1 \leftarrow b_{11}z_{22} - b_{21}$, $w_2 \leftarrow inv_0w_0$, $w_3 \leftarrow inv_1w_1$
$s_1 \leftarrow (inv_0 + inv_1)(w_0 + w_1) - w_2 - w_3(a_{11} + 1)$
if $(s_1 = 0)$ **then**
  Verwende einen anderen Algorithmus, um $D_1 + D_2$ zu berechnen.
end if
$s_0 \leftarrow w_2 - a_{10}w_3$
$\tilde{Z}_2 \leftarrow rz_{23}$, $Z_{32} \leftarrow \tilde{Z}_2Z_{21}$, $\tilde{Z}_2 \leftarrow \tilde{Z}_2^2$, $Z_{31} \leftarrow s_1Z_{21}$, $R \leftarrow rs_1$
$\tilde{s}_0 \leftarrow s_0z_{21}$, $S_0 \leftarrow s_0s_1$, $S \leftarrow S_0z_{21}$, $S_1 \leftarrow {s_1}^2$, $z_{31} \leftarrow {Z_{31}}^2$, $z_{32} \leftarrow {Z_{32}}^2$
$l_2 \leftarrow S_1a_{21}$, $l_0 \leftarrow S_0a_{20}$, $l_1 \leftarrow (S_1 + S_0)(a_{21} + a_{20}) - l_0 - l_2$
$l_2 \leftarrow l_2 + S$
$b_{31} \leftarrow Rb_{21}$
$a_{30} \leftarrow (s_0 - a_{11}s_1)(\tilde{s}_0 - inv_1s_1) + l_1 - a_{10}z_{31} + 2b_{31} + (2a_{21} + inv_1)\tilde{Z}_2$
$a_{31} \leftarrow 2S - inv_1S_1 - z_{32}$
$l_2 \leftarrow l_2 - a_{31}$, $w_0 \leftarrow l_2a_{30}$, $w_1 \leftarrow l_2a_{31}$
$b_{31} \leftarrow w_1 - z_{31}(l_1 + b_{31} - a_{30})$
$b_{30} \leftarrow w_0 - z_{31}(l_0 - Rb_{20})$
return $D_3 = [a_{31}, a_{30}, b_{31}, b_{30}, Z_{31}, Z_{32}, z_{31}, z_{32}] = D_1 + D_2$

4.2.3 Hyperelliptische Kurven vom Geschlecht 2 über Körpern der Charakteristik 2

Für hyperelliptische Kurven über Körpern der Charakteristik 2 habe ich keinerlei spezielle Algorithmen implementiert. In der Literatur (etwa in [9, Abschnitt 14.5]) finden sich allerdings solche Algorithmen. Die Ideen, die dort verwendet werden, sind denen die im vorigen Abschnitt vorgestellt wurden sehr ähnlich und sollen nun kurz dargestellt werden.
Auch bei der Implementierung von hyperelliptischen Kurven vom Geschlecht 2 über Körpern der Charakteristik 2 ist der erste Schritt die Vereinfachung der Kurvengleichung. Auch hier erhält man durch die Variablentransformation $x \mapsto x - f_4/5$ eine Kurvengleichung mit $f_4 = 0$, die weitere Vereinfachung zu $h(x) = 0$ ist hier allerdings nicht möglich, wie Lemma 4.36 besagt, man erhält also eine Kurvengleichung der Form

$$C : y^2 + (h_2x^2 + h_1x + h_0)y = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0.$$

Lemma 4.36. *[25] Sei C eine hyperelliptische Kurve über K mit $h(x) = 0$. Dann ist* $\mathrm{char}(K) \neq 2$.

Beweis. Angenommen $h(x)$ ist Null und $\mathrm{char}(K) = 2$. Dann sind die partiellen Ableitungen von Gleichung 4.1 gerade $0 = 0$ und $f'(x) = 0$. Sei $x_1 \in \overline{K}$ eine Nullstelle von $f'(x)$ und sei $y_1 \in \overline{K}$ eine Nullstelle von $y^2 - f(x_1)$. Dann erfüllt der Punkt (x_1, y_1) sowohl die Kurvengleichung als auch beide partiellen Ableitungen. Also folgt, dass $h(x) \neq 0$ wenn $\mathrm{char}(K) = 2$. □

Auch für den Fall, dass die Charakteristik von K gleich 2 ist, finden sich verschiedene Vorschläge für die Darstellung der Divisoren, Algorithmen für die Addition von Divisoren in neuen Koordinaten finden sich in [20], hier wird auch der Speziallfall behandelt, dass eine Kurve vorliegt, in deren Kurvengleichung $h_2 = 0$ ist. Für diesen Fall lässt sich die Verdopplung von Divisoren durch die Verwendung sogenannter *neuester Koordinaten* (*recent coordinates*) noch etwas schneller berechnen, die Addition ist in diesem Falle aber etwas langsamer (siehe [9, Abschnitt 14.5.5.b]).
Neben der Addition von Divisoren gibt es hier auch die Möglichkeit der Halbierung (siehe [6]), die dann ähnlich wie bei elliptischen Kurven für die Vervielfachung von Divisoren verwendet werden kann.

5 Abschließende Betrachtungen

5.1 Struktur der Implementierung

Ziel der Implementierung war es, auf beliebigen elliptischen und hyperelliptischen Kurven rechnen zu können. Die Fälle die für kryptographische Anwendungen von Bedeutung sind, sollten bei der Optimierung im Vordergrund stehen. Durch den objektorientierten Ansatz sollte eine leichte Erweiterbarkeit erreicht werden. Alle in der Arbeit vorgestellten Algorithmen sind, wenn nicht im Text anders angegeben, in der Bibliothek implementiert, außer der Punktmultiplikation mit der w-NAF (Algorithmus 35).
Den Kern meiner Implementierung bilden Klassen, die die mathematischen Strukturen repräsentieren, die in den vorigen Kapiteln vorgestellt wurden. Hierbei gibt es jeweils eine Klasse, die die Struktur selbst und eine Klasse, die deren Elemente darstellt. Die durch entsprechende Klassen erfassten Strukturen sind im einzelnen:

- der Körper $\mathbb{F}_2$ sowie dessen Elemente
- der Körper $\mathbb{F}_3$ sowie dessen Elemente
- Körper $\mathbb{F}_p$ sowie deren Elemente
- Körper $\mathbb{F}_{2^k}$ sowie deren Elemente
- Körper $\mathbb{F}_{3^k}$ sowie deren Elemente
- Körper $\mathbb{F}_{p^k}$ sowie deren Elemente
- elliptische Kurven $E : y^2 = x^3 + ax + b$ über $\mathbb{F}_{p^k}$, $p \notin \{2, 3\}$ sowie Punkte auf solchen Kurven
- nicht supersinguläre elliptische Kurven $E : y^2 + xy = x^3 + ax^2 + b$ über $\mathbb{F}_{2^k}$, sowie Punkte auf solchen Kurven
- supersinguläre elliptische Kurven $E : y^2 + cx = x^3 + ax + b$ über $\mathbb{F}_{2^k}$, sowie Punkte auf solchen Kurven
- nicht supersinguläre elliptische Kurven $E : y^2 = x^3 + ax^2 + b$ über F_{3^k}, sowie Punkte auf solchen Kurven
- supersinguläre elliptische Kurven $E : y^2 = x^3 + ax + b$ über $\mathbb{F}_{3^k}$, sowie Punkte auf solchen Kurven
- beliebige hyperelliptische Kurven $C : y^2 + h(x)y = f(x)$ und deren Divisoren

- hyperelliptische Kurven $C : y^2 = f(x)$ über $\mathbb{F}_{p^k}$, $p \notin \{2, 5\}$ vom Geschlecht 2 und deren Divisoren

Darüberhinaus gibt es eine Klasse, die den Datentyp `mpz_t` aus der GnuMP-Bibliothek kapselt und so komfortable Zugriffsmöglichkeiten auf große Ganzzahlen bietet, eine Klasse, die mit Hilfe eines Templates die Konstruktion von Polynomen über beliebigen Körpern erlaubt, eine Spezialisierung dieses Templates für Polynome über $\mathbb{F}_2$, die intern den Datentyp `dynamic_bitset` aus der boost-Bibliothek verwendet sowie einige weitere Klassen (etwa für die Darstellung der NAF und der w-NAF).
Die Korrektheit der Implementierung habe ich überprüft, indem ich mit der Computer-Algebra-Suite Magma [3] zufällige Kurven erzeugt, auf jeder dieser Kurven mehrere zufällige Punkte angelegt und jeden dieser Punkte mit 60 zufälligen Werten multipliziert habe. Anschließend habe ich die gleichen Berechnungen auch von einem Testprogramm durchführen lassen, das auf meiner Bibliothek basiert und die Ausgaben dann mit denen von Magma verglichen. Die Testfälle waren so gewählt, dass für jede der Klassen, die eine Kurve repräsentiert, mindestens zwei verschiedene Instanzen angelegt wurden.

5.2 Anwendungsbereich der Implementierung

In erster Linie soll meine Implementierung eine Sammlung von Algorithmen und Methoden darstellen, die bei der Implementierung elliptischer und hyperelliptischer Kurven für kryptographische Zwecke zum Einsatz kommen. Um tatsächlich auf elliptischen oder hyperelliptischen Kurven basierende sicherheitskritische kryptographische Protokolle zu implementieren, kann die Bibliothek dennoch nicht ohne weiteres zum Einsatz kommen. Zum einen liegt dies daran, dass die vorgestellten Algorithmen für eine solche Anwendung so angepasst werden sollten, dass sie Schutz gegen Seitenkanalattacken bieten, die häufig eine recht effiziente Möglichkeit darstellen konkrete Implementierungen anzugreifen. Einen Überblick darüber, welche Seitenkanalattacken es gegen Kryptosysteme gibt, die auf elliptischen Kurven basieren, und welche Gegenmaßnahmen gegen solche Attacken getroffen werden sollten, gibt [13].
Der zweite Grund, weswegen die Bibliothek nicht für sicherheitskritische Anwendungen eingesetzt werden kann, ist die Art der Erzeugung von Zufallszahlen. Es ist zwar mit meiner Implementierung möglich, zufällige Elemente aus $\mathbb{Z}$ sowie aus den implementierten endlichen Körpern zu erzeugen, diese Zufallswerte basieren allerdings auf einem leicht rekonstruierbaren Initialwert (*random seed*), nämlich der Systemzeit. Somit sind die erzeugten Zufallszahlen ebenfalls rekonstruierbar, was sie für kryptographische Protokolle keinesfalls sein dürfen.
Es wäre sicherlich mit relativ geringem Aufwand möglich, in der Bibliothek sowohl kryptographisch sichere Zufallszahlen zu erzeugen als auch Gegenmaßnahmen gegen Seitekanalattacken zu implementieren, dennoch wäre es dann nicht sinnvoll, ein auf meiner Bibliothek basierendes kryptographisches Protokoll umzusetzen, da man sich für ein solches Protokoll meist eine Kurve fest vorgeben und eine speziell für diese Kurve optimierte Implementierung verwenden kann. Daher sehe ich den Anwendungsbereich meiner Bibliothek im nicht sicherheitskritischen Umfeld kryptographischer Anwendungen. So wird sie zum Beispiel

in einem Programm verwendet, das basierend auf dem in [5] vorgestellten Algorithmus elliptische Kurven über Primkörpern konstruiert, die besonders gut für paarungsbasierte Kryptographie geeignet sind.
Durch den objektorientierten Ansatz ist die Bibliothek sowohl einfach zu verwenden, als auch leicht zu erweitern, daher ist sie gut geeignet um neue Ideen und Algorithmen zu testen. Hierfür lassen sich in der Regel auch Computer-Algebra-Programme verwenden, die Ergebnisse eines Geschwindigkeitsvergleiches eines auf meiner Bibliothek basierenden Programmes mit der kommerziellen Suite Magma sind in den Tabellen 5.1, 5.2 und 5.3 dargestellt. Die Tests habe ich auf die für kryptographische Anwendungen interessanten Fälle beschränkt. Man sieht, dass Magma in allen Fällen schneller rechnet, die deutlichsten Unterschiede treten bei Multiplikationen in Körpern der Charakterstik 2 auf, was wohl meiner Entscheidung, diese selber zu programmieren, geschuldet ist — bei der Verwendung einer über lange Zeit entwickelten und entsprechend optimierten Bibliothek wie der NTL [4] wären für Operationen in diesen Körpern sicher kürzere Berechnungszeiten erreichbar.
Die gemessenen Zeiten zeigen allerdings auch, dass die Berechnungen meiner Implementierung so schnell sind, dass sie durchaus für die oben beschriebenen Anwendungen eingesetzt werden kann. Kommen etwa bei einer Anwendung viele Zugriffe auf Vektoren oder Felder zum Einsatz, so kann eine auf meiner Bibliothek basierende Anwendung durchaus schneller sein, als ein entsprechendes Programm in Magma, da diese Operationen dort recht viel Zeit kosten. Weitere Vorteile gegenüber Computer-Algebra-Programmen ergeben sich aus der Implementierung als C++ Bibliothek: So ist es kein Problem, die Klassen und Algorithmen in komplexere Software einfließen zu lassen und hierbei auch beliebige andere der zahlreichen existierenden C++ Bibliotheken zu verwenden.

Körper	**libecce**	**Magma**
$\mathbb{F}_p$ der Größenordnung 2^{160}	ca. 0.35 s	ca. 0.18 s
$\mathbb{F}_p$ der Größenordnung 2^{192}	ca. 0.37 s	ca. 0.22 s
$\mathbb{F}_p$ der Größenordnung 2^{256}	ca. 0.41 s	ca. 0.27 s
$\mathbb{F}_{2^k}$ der Größenordnung 2^{160}	ca. 5.2 s	ca. 0.29 s
$\mathbb{F}_{2^k}$ der Größenordnung 2^{192}	ca. 6.9 s	ca. 0.33 s
$\mathbb{F}_{2^k}$ der Größenordnung 2^{256}	ca. 10.5 s	ca. 0.46 s

Tabelle 5.1: Zeitaufwand für 100000 Multiplikationen in verschiedenen endlichen Körpern in meiner Implementierung (libecce) und Magma

5.3 Ausblick

Die in Abschnitt 5.2 beschriebenen Geschwindigkeitsmessungen zeigen, dass die im Rahmen dieser Diplomarbeit entwickelte Bibliothek in einigen Fällen nur wenig langsamer ist, als über lange Zeit optimierte kommerzielle Software. Andere Teile, auch durchaus solche, die für kryptographische Anwendungen von besonderer Bedeutung sind, sind allerdings deutlich langsamer, eine naheliegende Fortsetzung der Arbeit ist es also, gerade diese Fälle

Kurve	**libecce**	**Magma**
$E : y^2 = x^3 + ax + b$ über $\mathbb{F}_p$ der Größenordnung 2^{160}	ca. 1,27 s	ca. 0,37 s
$E : y^2 = x^3 + ax + b$ über $\mathbb{F}_p$ der Größenordnung 2^{192}	ca. 1,55 s	ca. 0,5 s
$E : y^2 = x^3 + ax + b$ über $\mathbb{F}_p$ der Größenordnung 2^{256}	ca. 2,25 s	ca. 0,9 s
$E : y^2 + xy = x^3 + ax^2 + b$ über $\mathbb{F}_{2^k}$ der Größenordnung 2^{160}	ca. 18,1 s	ca. 1,2 s
$E : y^2 + xy = x^3 + ax^2 + b$ über $\mathbb{F}_{2^k}$ Größenordnung 2^{192}	ca. 28,5 s	ca. 1,8 s
$E : y^2 + xy = x^3 + ax^2 + b$ über $\mathbb{F}_{2^k}$ der Größenordnung 2^{256}	ca. 1 min	ca. 3,65 s

Tabelle 5.2: Zeitaufwand für 100 Punktmultiplikationen für verschiedene elliptische Kurven in meiner Implementierung (libecce) und Magma

Kurve	**libecce**	**Magma**
$C : y^2 = f(x), \deg(f) = 5$ $\mathbb{F}_p$ der Größenordnung 2^{80}	ca. 4,5 s	ca. 1,34 s
$C : y^2 = f(x), \deg(f) = 5$ $\mathbb{F}_p$ der Größenordnung 2^{96}	ca. 5,3 s	ca. 1,65 s
$C : y^2 = f(x), \deg(f) = 5$ $\mathbb{F}_p$ der Größenordnung 2^{128}	ca. 7,3 s	ca. 2,64 s

Tabelle 5.3: Zeitaufwand für 100 Divisorklassenvervielfachungen für verschiedene hyperelliptische Kurven in meiner Implementierung (libecce) und Magma

(insbesondere sind hier die Operationen auf Polynomen über dem $\mathbb{F}_2$ zu nennen) weiter zu optimieren.
Die in Abschnitt 4.2.3 nur kurz dargestellten Ideen zur Implementierung hyperelliptischer Kurven vom Geschlecht 2 über binären Körpern sind nicht in meine Implementierung eingeflossen, eine konsequente Weiterentwicklung sollte diese in entsprechenden Klassen darstellen und in die Bibliothek aufnehmen.
Über die reine Weiterentwicklung der im Rahmen dieser Arbeit entstandenen Implementierung hinaus bieten sich Anknüpfungsmöglichkeiten im Bereich der Hardwareimplementierung der beschriebenen Strukturen, sowie der Untersuchung, wie die vorgestellten Algorithmen verbessert werden können, wenn man sich auf eine spezielle Kurve festlegen kann. Zu beiden Themenbereichen existiert bereits umfangreiche Literatur, im Bereich der Hardwareimplementierung sind hier zum Beispiel die Arbeiten von Thomas Wollinger (siehe [31] oder [32]), sowie [14, Abschnitt 5.2] zu nennen, zu zweiterem findet man etwa in [14, Abschnitt 3.3.2] ausführliche Beschreibungen.

Index

Literaturverzeichnis

[1] boost c++ libraries. `http://www.boost.org/`.

[2] Gnu Multiple Precision Arithmetic Library. `http://www.swox.com/gmp/`.

[3] Magma — high performance software for algebra, number theory, and geometry. `http://magma.maths.usyd.edu.au/`.

[4] Ntl: A library for doing number theory. `http://www.shoup.net/ntl/`.

[5] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography – SAC'2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer-Verlag, 2006.

[6] Peter Birkner. Efficient Divisor Class Halving on Genus Two Curves. Cryptology ePrint Archive, Report 2006/257, 2006. `http://eprint.iacr.org/`.

[7] Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart, editors. *Advances in Elliptic Curve Cryptography*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2005.

[8] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[9] Henri Cohen, Gerhard Frey, Christophe Douche, Roberto M. Avanzi, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete Mathematics and its Applications. Chapman & Hall/CRC, Taylor & Francis Group, Boca Raton, 2006.

[10] A. Enge and P. Gaudry. A General Framework for Subexponential Discrete Logarithm Algorithms, 2000. `http://citeseer.ifi.unizh.ch/enge00general.html`.

[11] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[12] P. Gaudry, E. Thomé, N. Thériault, and C. Diem. A double large prime variation for small genus hyperelliptic index calculus. Cryptology ePrint Archive, Report 2004/153, 2004. `http://eprint.iacr.org/`.

[13] Louis Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *Proceedings of PKC 2003, LNCS 2567*, pages 199–211. Springer-Verlag Berlin Heidelberg, 2003.

[14] D. Hankerson, A. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, New York, 2004.

[15] Juraj Hromkovic. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics.* Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[16] Antoine Joux. A one round protocol for tripartite diffie-hellman. In *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag.

[17] Neal Koblitz. Elliptic curve cryptosystems. In *Mathematics of Computation, Vol. 48*, pages 203–209, 1987.

[18] Neal Koblitz. Hyperelliptic cryptosystems. *Journal of Cryptology*, 1(3):139–150, 1989. http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/BF02252872.

[19] Neal Koblitz. *Algebraic Aspects of Cryptography.* Springer, Berlin, 1998.

[20] Tanja Lange. Weighted Coordinates on Genus 2 Hyperelliptic Curves. Cryptology ePrint Archive, Report 2002/153, 2002. http://eprint.iacr.org/.

[21] Tanja Lange. A Note on López-Dahab Coordinates. Cryptology ePrint Archive, Report 2004/323, 2004. http://eprint.iacr.org/.

[22] Henrik W. Lenstra. Factoring Integers with Elliptic Curves. *Annals of mathematics*, 126(2):649–673, 1987.

[23] Rudolf Lidl and Harald Niederreiter. *Finite Fields.* Cambridge University Press, Cambridge, 1997.

[24] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1996. http://www.cacr.math.uwaterloo.ca/hac.

[25] Alfred J. Menezes, Yi-Hong Wu, and Robert J. Zuccherato. An Elementary Introduction to Hyperelliptic Curves. In *[19]*.

[26] Victor S. Miller. Uses of Elliptic Curves in Cryptography. In *Proceedings of CRYPTO '85, LNCS 218*, pages 417–426. Springer-Verlag, 1986.

[27] I.A. Semaev. Evaluation of Discrete Logarithms in a Group of p-Torsion Points of an Elliptic Curve in Characteristic p. *Mathematics of Computation*, 67(221):353–356, 1998.

[28] J. H. Silverman. *The Arithmetic of Elliptic Curves.* Springer-Verlag, New York, 1986.

[29] Joseph H. Silverman and John Tate. *Rational Points on Elliptic Curves.* Springer Science+Business Media Inc., 1992.

[30] Henning Stichtenoth. *Algebraic Function Fields and Codes.* Springer, Berlin Heidelberg, 1993.

[31] Thomas Wollinger. Computer Architectures for Cryptosystems Based on Hyperelliptic Curves. Master's thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, May 2001.

[32] Thomas Wollinger. *Software and Hardware Implementation of Hyperelliptic Curve Cryptosystems.* PhD thesis, Ruhr-Universität Bochum, Mai 2004.