Bestimmung von Weil-Funktionen auf elliptischen Kurven

—

Meromorphe Funktionen
mit Divisor $n\,[P] - n\,[\mathcal{O}]$
für $n \in \{4, \dots, 10, 12\}$

—

Miller-Algorithmus und Faktorisierung von x^n

Patrick Reichert

30. August 2016

Zusammenfassung

Für $n \in \{4, \dots, 10, 12\}$ wird für jede elliptische Kurve $(E, \mathcal{O})$ mit n-Torsionspunkt $P = (0, 0) \in E$ die Gleichung der Weil-Funktion bestimmt, also der meromorphen Funktion $f\colon E \to \overline{\mathbb{C}}$ mit Divisor $n\,[P]-n\,[\mathcal{O}]$. Die berechneten Funktionen werden mit Hilfe der charakteristischen Eigenschaft $f(Q) \cdot f(-Q) = (-1)^n x^n$ für alle Punkte $Q = (x, y) \in E$ zusätzlich verifiziert.

Mathematical Subject Classification 2010: 14H52 (primary)

Inhaltsverzeichnis

1 Einleitung

André Weil (1906–1998) hat 1940 die Weil-Paarung für n-Torsionspunkte einer elliptischen Kurve eingeführt [Wei40], mit deren Hilfe man jedem n-Torsionspunkt einer elliptischen Kurve eine Weil-Funktion [Mil04] zuordnen kann.

2 Existenz und Konstruktion von Weil-Funktionen

Theorem 2.1 (Existenz von Weil-Funktionen, [Sil86], Korollar III.3.5) *Sei E eine über $\mathbb{C}$ definierte elliptische Kurve mit Basispunkt $\mathcal{O}$. Sei $P \in E$ ein n-Torsionspunkt, d.h. $nP = \mathcal{O}$, $P \neq \mathcal{O}$, $n \geq 2$.*

Dann gibt es eine meromorphe Funktion $f\colon E \to \overline{\mathbb{C}}$ mit Divisor

$$\operatorname{div}(f) = n\,[P] - n\,[\mathcal{O}].$$

*Die Divisorschreibweise soll ausdrücken, dass die Funktion f im Punkt P eine n-fache Nullstelle, im Basispunkt $\mathcal{O}$ einen n-fachen Pol und ansonsten keine weiteren Nullstellen und Polstellen besitzt. Eine solche Funktion f wird **Weil-Funktion** zum n-Torsionspunkt P der elliptischen Kurve E genannt.*

Beweis. Die Existenz einer solchen Funktion folgt aus Jacobi-Abel wegen $n \cdot P - n \cdot \mathcal{O} = \mathcal{O}$. □

Theorem 2.2 (Konstruktion von Weil-Funktionen, [Mil04], Proposition 2 und Lemma 2) *Sei E eine über $\mathbb{C}$ definierte elliptische Kurve mit Basispunkt $\mathcal{O}$. Dann gilt:*

(a) *Für zwei Punkte $Q, R \in E$ sei $L_{Q,R}\colon E \to \overline{\mathbb{C}}$ eine meromorphe Funktion, so dass $L_{Q,R} = 0$ die Gleichung der Gerade durch die Punkte Q und R ist (bzw. die Gleichung der Tangente an E im Punkt Q, falls $Q = R$ gilt). Diese Funktion besitzt den Divisor*

$$\operatorname{div}(L_{Q,R}) = [Q] + [R] + [-(Q+R)] - 3\,[\mathcal{O}].$$

(b) *Sei $P \in E$ ein n-Torsionspunkt, d.h. $nP = \mathcal{O}$, $P \neq \mathcal{O}$, $n \geq 2$. Definiert man für $m \in \mathbb{N}$ die meromorphen Funktionen $f_m\colon E \to \overline{\mathbb{C}}$ rekursiv gemäß*

$$\begin{aligned} f_0 &\equiv 1, \\ f_1 &\equiv 1, \\ f_{a+b} &= f_a \cdot f_b \cdot \frac{L_{aP,bP}}{L_{(a+b)P,-(a+b)P}} \quad \textit{für alle } a, b \in \mathbb{N}, \end{aligned}$$

so gilt für den Divisor

$$\operatorname{div}(f_m) = m\,[P] - [mP] - (m-1)\,[\mathcal{O}] \quad \textit{für alle } m \geq 1.$$

Insbesondere folgt daraus für $m = n$ wegen $nP = \mathcal{O}$ sofort

$$\operatorname{div}(f_n) = n\,[P] - n\,[\mathcal{O}].$$

Damit ist f_n eine Weil-Funktion zum n-Torsionspunkt $P \in E$.

Beweis.

(a) Die Aussage folgt direkt aus der Definition der Addition von Punkten auf der elliptischen Kurve E: Die Gerade durch die Punkte Q und R schneidet die elliptische Kurve E nur noch im Punkt $-(Q+R)$; der Basispunkt $\mathcal{O}$ muss dann ein dreifacher Pol der Geradengleichung sein.

(b) Die Aussage wird induktiv bewiesen. Nach (a) gilt für die Divisoren der Geradengleichungen

$$\begin{aligned} \operatorname{div}(L_{aP,bP}) &= [aP] + [bP] + [-(a+b)P] - 3\,[\mathcal{O}] \quad \text{und} \\ \operatorname{div}(L_{(a+b)P,-(a+b)P}) &= [(a+b)P] + [-(a+b)P] - 2\,[\mathcal{O}]. \end{aligned}$$

Als Induktionsvoraussetzung gelte

$$\begin{aligned} \operatorname{div}(f_a) &= a\,[P] - [aP] - (a-1)\,[\mathcal{O}] \quad \text{und} \\ \operatorname{div}(f_b) &= b\,[P] - [bP] - (b-1)\,[\mathcal{O}]. \end{aligned}$$

Daraus erhält man durch Einsetzen die Induktionsbehauptung

$$\begin{aligned} \operatorname{div}(f_{a+b}) &= \operatorname{div}(f_a) + \operatorname{div}(f_b) + \operatorname{div}(L_{aP,bP}) - \operatorname{div}(L_{(a+b)P,-(a+b)P}) \\ &= (a+b)\,[P] - [(a+b)P] - (a+b-1)\,[\mathcal{O}] \end{aligned}$$ □

Algorithmus 2.3 (Miller-Algorithmus zur Bestimmung der Weil-Funktion, [Mil04], Algor. 1)

- ***Eingabe:*** *E – elliptische Kurve über $\mathbb{C}$ mit Basispunkt $\mathcal{O}$*

  P – n-Torsionspunkt von E

n – Ordnung des Torsionspunkts P, $n \geq 2$

- ***Ausgabe:*** *f_n – Weil-Funktion von P, $f_n \colon E \to \overline{\mathbb{C}}$ mit Divisor $\operatorname{div}(f_n) = n\,[P] - n\,[\mathcal{O}]$*
- ***Algorithmus:*** *Die Weil-Funktion f_n wird durch einen Double-And-Add-Algorithmus mit Hilfe der Binärdarstellung der Zahl n berechnet.*
  1. ***Initialisierung.*** *Setze $f_1 := 1$, $m := 1$.*
  2. ***Schleife.*** *Iteriere absteigend über die Bits von n, beginne mit dem Bit hinter dem höchstwertigen 1-Bit.*
     (a) ***Verdopplung.*** *Setze*
     $$f_{2m} := f_m^2 \cdot \frac{L_{mP,mP}}{L_{2mP,-2mP}}$$
     $$m := 2m$$
     (b) ***Inkrementierung.*** *Ist das aktuelle Bit 1, dann setze*
     $$f_{m+1} := f_m \cdot \frac{L_{mP,P}}{L_{(m+1)P,-(m+1)P}}$$
     $$m := m + 1$$
     (c) ***Abbruchtest.*** *Sobald $m = n$ gilt, ist der Algorithmus beendet und f_n ist berechnet. Ansonsten wird die Iteration fortgesetzt.*

Beweis. Der Algorithmus verwendet direkt die Formeln aus Theorem 2.2(b). Um die Anzahl der Iterationsschritte möglichst gering zu halten, wird nicht jede einzelne Funktion $f_1, f_2, f_3, \ldots, f_n$ bestimmt. Stattdessen werden anhand der Binärdarstellung von n geeignete Verdopplungsschritte durchgeführt. □

Beispiel 2.4 (Weil-Funktion für $X_1(5)$) *Elliptische Kurven mit 5-Torsionspunkt $P = (0,0)$ werden über $\mathbb{C}$ durch die modulare Kurve $X_1(5)$ parametrisiert und besitzen die Form*

$$E_c \colon y^2 + (c+1)xy + cy = x^3 + cx^2$$

mit dem komplexen Parameter $c \in \mathbb{C} \setminus \{0, -\frac{11}{2} \pm \frac{5}{2}\sqrt{5}\}$. Der Torsionspunkt besitzt die Vielfachen $2P = (-c, c^2)$, $3P = (-c, 0)$ und $4P = (0, -c)$. Mit Algorithmus 2.3 soll die Weil-Funktion $f_5 \colon E_c \to \overline{\mathbb{C}}$ zum 5-Torsionspunkt $P \in E_c$ bestimmt werden. Die im Algorithmus benötigten Geradengleichungen lauten:

Funktion $L \colon E_c \to \overline{\mathbb{C}}$ für Gerade $L(x,y) = 0$	*Divisor $\operatorname{div}(L)$ gemäß Theorem 2.2(a)*
$L_{P,P} = y$	$\operatorname{div}(L_{P,P}) = 2\,[P] + [3P] - 3\,[\mathcal{O}]$
$L_{2P,-2P} = x + c$	$\operatorname{div}(L_{2P,-2P}) = [2P] + [3P] - 2\,[\mathcal{O}]$
$L_{2P,2P} = cx + y$	$\operatorname{div}(L_{2P,2P}) = 2\,[2P] + [P] - 3\,[\mathcal{O}]$
$L_{4P,-4P} = x$	$\operatorname{div}(L_{4P,-4P}) = [P] + [4P] - 2\,[\mathcal{O}]$
$L_{4P,P} = x$	$\operatorname{div}(L_{4P,P}) = [P] + [4P] - 2\,[\mathcal{O}]$
$L_{5P,-5P} = 1$	$\operatorname{div}(L_{5P,-5P}) = 0$

Der Miller-Algorithmus 2.3 führt die folgenden Schritte durch:

m	*Meromorphe Funktion $f_m \colon E_c \to \overline{\mathbb{C}}$*	*Divisor $\operatorname{div}(f_m)$ gemäß Theorem 2.2(b)*
1	$f_1 = 1$	$\operatorname{div}(f_1) = 0$
2	$f_2 = f_1^2 \cdot \frac{L_{P,P}}{L_{2P,-2P}} = \frac{y}{x+c}$	$\operatorname{div}(f_2) = 2\,[P] - [2P] - [\mathcal{O}]$
4	$f_4 = f_2^2 \cdot \frac{L_{2P,2P}}{L_{4P,-4P}} = \frac{y^2}{(x+c)^2} \cdot \frac{cx+y}{x}$	$\operatorname{div}(f_4) = 4\,[P] - [4P] - 3\,[\mathcal{O}]$
5	$f_5 = f_4 \cdot \frac{L_{4P,P}}{L_{5P,-5P}} = \frac{y^2}{(x+c)^2} \cdot \frac{cx+y}{x} \cdot \frac{x}{1}$	$\operatorname{div}(f_5) = 5\,[P] - 5\,[\mathcal{O}]$

Die Weil-Funktion zum 5*-Torsionspunkt* $P = (0,0)$ *lautet also* $f_5(x,y) = \frac{y^2(cx+y)}{(x+c)^2}$. *Ergänzt man den Zähler formal um das Vielfache*

$$(-x+y-c)(x^3+cx^2-y^2-(c+1)xy-cy)$$

der Kurvengleichung, so lässt sich der Bruch kürzen und man erhält die nennerfreie Darstellung der Weil-Funktion

$$f_5(x,y) = -x^2+xy+y. \qquad \square$$

3 Berechnete Weil-Funktionen

Theorem 3.1 (Gleichungen für Weil-Funktionen) *Die nachfolgende Tabelle gibt für* $n \in \{4,\ldots,10,12\}$ *die allgemeine Darstellung einer elliptischen Kurve mit* n*-Torsionspunkt* $P = (0,0)$ *und die zugehörige Weil-Funktion an:*

n	*Darstellung und Diskriminante der elliptischen Kurve* E_c *mit* n*-Torsionspunkt* $P=(0,0)$, *Weil-Funktion* $f_n\colon E_c \to \overline{\mathbb{C}}$ *mit Divisor* $\operatorname{div}(f_n) = n\,[P] - n\,[\mathcal{O}]$
4	$E_c\colon y^2+xy+cy = x^3+cx^2$, $c \in \mathbb{C}\setminus\left\{0, \frac{1}{16}\right\}$, $\operatorname{disc}(E_c) = -(16c-1)\,c^4$, $f_4(x,y) = x^2-y$
5	$E_c\colon y^2+(c+1)xy+cy = x^3+cx^2$, $c \in \mathbb{C}\setminus\left\{0, -\frac{11}{2} \pm \frac{5}{2}\sqrt{5}\right\}$, $\operatorname{disc}(E_c) = -(c^2+11c-1)\,c^5$, $f_5(x,y) = -x^2+xy+y$
6	$E_c\colon y^2+(1-c)xy-c(c+1)y = x^3-c(c+1)x^2$, $c \in \mathbb{C}\setminus\left\{0, -1, -\frac{1}{9}\right\}$, $\operatorname{disc}(E_c) = (c+1)^3(9c+1)c^6$, $f_6(x,y) = y^2-(c+1)\,xy-(c+1)^2\,y+(c+1)^2\,x^2$
7	$E_c\colon y^2+(1-c-c^2)xy+(c+1)c^2y = x^3+(c+1)c^2x^2$, $c \in \mathbb{C}$ *mit* $\operatorname{disc}(E_c) \neq 0$, $\operatorname{disc}(E_c) = -(c^3+8c^2+5c-1)(c+1)^7c^7$, $f_7(x,y) = (c-1)\,y^2+x^2y-c^3\,xy+c^4\,y-c^4\,x^2$
8	$E_c\colon y^2+(1-2c^2)xy-(2c+1)(c+1)^3cy = x^3-(2c+1)(c+1)^2cx^2$, $c \in \mathbb{C}\setminus\left\{0, -1, -\frac{1}{2}, -\frac{1}{2} \pm \frac{1}{4}\sqrt{2}\right\}$, $\operatorname{disc}(E_c) = (8c^2+8c+1)(2c+1)^4(c+1)^8c^8$, $f_8(x,y) = xy^2+(2c+3)(c+1)^3\,y^2-2(c+1)^2\,x^2y-(2c+1)^2(c+1)^4\,xy$ $\qquad -(2c+1)^2(c+1)^7\,y+(2c+1)^2(c+1)^6\,x^2$

n	*Darstellung und Diskriminante der elliptischen Kurve E_c mit n-Torsionspunkt $P=(0,0)$, Weil-Funktion f_n mit Divisor $\operatorname{div}(f_n) = n\,[P] - n\,[\mathcal{O}]$*
9	$E_c\colon y^2 + (c^3+c^2+1)xy + (c^2+c+1)(c+1)c^2 y = x^3 + (c^2+c+1)(c+1)c^2x^2$, $c \in \mathbb{C}$ *mit* $\operatorname{disc}(E_c) \neq 0$, $\operatorname{disc}(E_c) = -(c^3+6c^2+3c-1)(c^2+c+1)^3(c+1)^9c^9$, $f_9(x,y) = y^3 + (c-1)(c^2+c+1)\,xy^2 + (c^2+c+1)^2(c^3+2c-1)\,y^2$ $- (2c-1)(c^2+c+1)^2\,x^2y + c^4(c^2+c+1)^3\,xy + c^4(c^2+c+1)^4\,y$ $- c^4(c^2+c+1)^4\,x^2$
10	$E_c\colon y^2 + (-c^3-2c^2+4c+4)xy + (c+1)(c+2)(c^2+6c+4)c^3y$ $= x^3 + (c+1)(c+2)c^3x^2$, $c \in \mathbb{C} \setminus \left\{0, -1, -2, -\frac{1}{2} \pm \frac{1}{2}\sqrt{5}, -3 \pm \sqrt{5}\right\}$, $\operatorname{disc}(E_c) = -(c^2+6c+4)^2(c^2+c-1)(c+2)^{10}(c+1)^5c^{10}$, $f_{10}(x,y) = 2(c^2-2c-2)\,y^3 + x^2y^2 - (2c+1)c^4\,xy^2 + (c^3+16c^2+22c+8)c^6\,y^2$ $- (3c+2)c^6\,x^2y + (c+1)^2c^{10}\,xy - (c+1)^2(c^2+6c+4)c^{12}\,y$ $+ (c+1)^2c^{12}\,x^2$
12	$E_c\colon y^2 + (6c^4-8c^3+2c^2+2c-1)xy - c(c-1)^5(2c-1)(2c^2-2c+1)(3c^2-3c+1)y$ $= x^3 - c(c-1)^2(2c-1)(2c^2-2c+1)(3c^2-3c+1)x^2$, $c \in \mathbb{C} \setminus \left\{0, 1, \frac{1}{2}, \frac{1}{2} \pm \frac{1}{2}i, \frac{1}{2} \pm \frac{1}{6}\sqrt{3}i, \frac{1}{2} \pm \frac{1}{6}\sqrt{3}\right\}$, $\operatorname{disc}(E_c) = (6c^2-6c+1)(3c^2-3c+1)^4(2c^2-2c+1)^3(2c-1)^6(c-1)^{12}c^{12}$, $f_{12}(x,y) = y^4 + (6c^2-8c+3)(2c^2-2c+1)\,xy^3$ $+ (c-1)^4(2c^2-2c+1)^2(36c^4-90c^3+96c^2-49c+10)\,y^3$ $+ 3(c-1)^2(5c^2-6c+2)(2c^2-2c+1)^2\,x^2y^2$ $+ (c-1)^4(14c^2-16c+5)(3c^2-3c+1)^2(2c^2-2c+1)^3\,xy^2$ $+ (c-1)^8(3c^2-3c+1)^2(2c^2-2c+1)^4(12c^4-42c^3+57c^2-33c+7)\,y^2$ $+ 2(c-1)^6(9c^2-10c+3)(3c^2-3c+1)^2(2c^2-2c+1)^4\,x^2y$ $+ (2c-1)^2(c-1)^8(3c^2-3c+1)^4(2c^2-2c+1)^5\,xy$ $- (2c-1)^2(c-1)^{13}(3c^2-3c+1)^4(2c^2-2c+1)^6\,y$ $+ (2c-1)^2(c-1)^{10}(3c^2-3c+1)^4(2c^2-2c+1)^6\,x^2$

Beweis. Die Weil-Funktionen wurden mit Hilfe des Miller-Algorithmus 2.3 bestimmt; die erstellte SAGE-Bibliothek ist in Kapitel A beschrieben. □

4 Verifikation der Berechnungen

Die mit Hilfe der SAGE-Implementierung des Miller-Algorithmus berechneten Weil-Funktionen sollen durch Nachweis einer charakteristischen Eigenschaft verifiziert werden: Jede Weil-Funktion $f_n(x,y)\colon E \to \overline{\mathbb{C}}$ ist ein Teiler des Polynoms x^n im von der Kurvengleichung von E aufgespannten Ideal.

Theorem 4.1 (Charakteristische Eigenschaft von Weil-Funktionen) *Sei $E\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ eine elliptische Kurve über $\mathbb{C}$ mit Basispunkt $\mathcal{O}$. Seien $P = (x_P, y_P), Q \in E$ zwei n-Torsionspunkte, d.h. $nP = \mathcal{O} = nQ$, $P \neq \mathcal{O} \neq Q$, $n \geq 2$. Seien $f_P, f_Q, f_{P+Q}\colon E \to \overline{\mathbb{C}}$ Weil-Funktionen von P, Q bzw. $P+Q$. Dann gilt:*

(a) *Es existiert eine meromorphe Funktion $h\colon E \to \overline{\mathbb{C}}$ mit*

$$\frac{f_P f_Q}{f_{P+Q}} = h^n.$$

(b) *Speziell für $Q = -P$ gilt*

$$\begin{aligned} f_Q(x,y) &= f_P(x, -a_1 x - y - a_3), \\ f_{P+Q}(x,y) &\equiv 1 \quad \textit{und} \\ h(x,y) &= \alpha\,(x - x_P) \end{aligned}$$

für eine Konstante $\alpha \in \mathbb{C} \setminus \{0\}$.

(c) *Ist $P = (0,0) \in E$, so gilt*

$$f_P(x,y) \cdot f_P(x, -a_1 x - y - a_3) = \alpha^n x^n.$$

Die Konstante $\alpha \in \mathbb{C} \setminus \{0\}$ hängt dabei nur von der gewählten Weil-Funktion f_P ab.

Beweis.

(a) Aus den Divisoren der Weil-Funktionen

$$\begin{aligned} \operatorname{div}(f_P) &= n\,[P] - n\,[\mathcal{O}], \\ \operatorname{div}(f_Q) &= n\,[Q] - n\,[\mathcal{O}], \\ \operatorname{div}(f_{P+Q}) &= n\,[P+Q] - n\,[\mathcal{O}] \end{aligned}$$

folgt für den Divisor des Quotienten

$$\operatorname{div}\left(\frac{f_P f_Q}{f_{P+Q}}\right) = n\,[P] + n\,[Q] - n\,[P+Q] - n\,[\mathcal{O}].$$

Wegen $P + Q - (P+Q) - \mathcal{O} = \mathcal{O}$ existiert nach [Sil86], Korollar III.3.5 eine meromorphe Funktion $h\colon E \to \overline{\mathbb{C}}$ mit Divisor

$$\operatorname{div}(h) = [P] + [Q] - [P+Q] - [\mathcal{O}]$$

und Potenz

$$h^n = \frac{f_P f_Q}{f_{P+Q}}.$$

(b) Auf der elliptischen Kurve $E\colon y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$ besitzt ein Punkt $(x,y) \in E$ das additive Inverse $-(x,y) = (x, -a_1 x - y - a_3)$ und somit gilt für $Q = -P$:

$$f_Q(x,y) = f_P(x, -a_1 x - y - a_3)$$

und

$$\operatorname{div}(h) = [P] + [-P] - 2\,[\mathcal{O}].$$

Nach Theorem 2.2(a) ist $h\colon E \to \overline{\mathbb{C}}$ somit die Gleichung der Gerade durch die Punkte $P = (x_P, y_P)$ und $-P = (x_P, -a_1 x_P - y_P - a_3)$ und besitzt damit die Gleichung $h(x,y) = \alpha(x - x_P)$ für eine Konstante $\alpha \in \mathbb{C} \setminus \{0\}$.

(c) Die Aussage folgt aus (a) und (b) für $P = (x_P, y_P) = (0,0)$. □

Theorem 4.2 (Verifikation der Weil-Funktionen) *Sei $E_n\colon y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$ eine der in Theorem 3.1 angegebenen elliptischen Kurven und f_n die zugeordnete Weil-Funktion für den n-Torsionspunkt $P = (0,0) \in E_n$ für $n \in \{4, \ldots, 10, 12\}$. Dann gilt*

$$f_n(x,y) \cdot f_n(x, -a_1 x - y - a_3) = (-1)^n x^n$$

für alle Punkte $(x,y) \in E_n$.

Beweis. Die Aussage lässt sich mit den SAGE-Routinen aus Kapitel C nachweisen, indem man folgende Schritte durchführt:

1. Die gemäß Theorem 4.1(c) existierende Konstante $\alpha \in \mathbb{C} \setminus \{0\}$ mit

   $$f_n(x,y) \cdot f_n(x, -a_1x - y - a_3) = \alpha^n x^n \quad \text{für alle } (x,y) \in E_n$$

   lässt sich durch SAGE bestimmen, indem $x = 1$ gesetzt wird:

   $$\alpha^n = f_n(1,y) \cdot f_n(1, -a_1 - y - a_3) \quad \text{für } (1,y) \in E_n.$$

   Dieser Ausdruck wird berechnet, indem er erst im Ideal $\mathbb{Q}[x,y,c]/\langle y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6\rangle$ reduziert wird und anschließend alle Potenzen y^m für $m \geq 2$ mit Hilfe der elliptischen Kurvengleichung und $x = 1$ ersetzt werden:

   $$\underbrace{y^m}_{\text{Grad } m} = y^{m-2} \cdot y^2 = \underbrace{y^{m-2}(-a_1y - a_3y + a_2 + a_4 + a_6 + 1)}_{\text{Grad } m-1}.$$

   Als Ergebnis erhält man $\alpha^n = (-1)^n$ für alle Weil-Funktionen aus Theorem 3.1.

2. Die Polynomdifferenz

   $$f_n(x,y) \cdot f_n(x, -a_1x - y - a_3) - \alpha^n x^n$$

   wird im Ideal $\mathbb{Q}[x,y,c]/\langle y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6\rangle$ reduziert. Als Ergebnis erhält man für alle Weil-Funktionen aus Theorem 3.1 die Differenz 0. $\square$

A SAGE-Bibliothek zur Berechnung von Weil-Funktionen

Entwickelt wurde eine Bibliothek für [SAGE] 7.0, mit der für eine elliptische Kurve E über $\mathbb{C}$ mit Basispunkt $\mathcal{O}$ und n-Torsionspunkt $P \in E$ eine Weil-Funktion $f\colon E \to \overline{\mathbb{C}}$ bestimmt werden kann, so dass $\operatorname{div}(f) = n\,[P] - n\,[\mathcal{O}]$ gilt.

A.1 Verwendung der Bibliothek

Der nachfolgende Quellcode verdeutlicht die Benutzung der Bibliothek, indem die Weilfunktionen für alle elliptischen Kurven mit 10-Torsionspunkt bestimmt werden:

```
# Berechnet die Weil-Funktion für X_1(10), reduziert die Potenzen x^3 und
# höher bzgl. der Gleichung der elliptischen Kurve und faktorisiert die
# Koeffizienten, um eine kompakte Darstellung der Weil-Funktion zu erhalten.
def CalcWeilFunction10():
    # Elliptische Kurve X_1(10) und 10-Torsionspunkt P definieren
    x, y, c = var('x y c');
    E_10 = EllipticCurve([-c^3 - 2*c^2 + 4*c + 4,
                          (c+1) * (c+2) * c^3,
                          (c+1) * (c+2) * (c^2+6*c+4) * c^3,
                          0, 0]);
    P_10 = E_10(0, 0);

    # Weil-Funktion mittels Miller-Algorithmus berechnen
    # (Funktion wird gleich im Quotientenring reduziert,
    #  damit eine nennerfreie Darstellung entsteht)
    f = WeilFunction(E_10, P_10, 10)

    # Potenzen x^n für n >= 3 ersetzen durch x^2, x, 1
    f = ReplaceHighXMonoms(f, E_10)

    # f als Polynom in x,y sehen und Koeffizienten faktorisieren
    f_str = FactorCoefficientsXYPoly(f)
    return f_str;
```

```
%time Weil10_str = CalcWeilFunction10();

+ ( (2) * (c^2 - 2*c - 2) ) *  y^3
+ ( 1 ) *  x^2*y^2
+ ( (-2) * (c + 1/2) * c^4 ) *  x*y^2
+ ( c^6 * (c^3 + 16*c^2 + 22*c + 8) ) *  y^2
+ ( (-3) * (c + 2/3) * c^6 ) *  x^2*y
+ ( (c + 1)^2 * c^10 ) *  x*y
+ ( (-1) * (c + 1)^2 * c^12 * (c^2 + 6*c + 4) ) *  y
+ ( (c + 1)^2 * c^12 ) *  x^2
CPU times: user 12.8 s, sys: 89 ms, total: 12.9 s
```

Elliptische Kurven mit 10-Torsionspunkt $P = (0,0)$ werden über $\mathbb{C}$ durch die modulare Kurve $X_1(10)$ parametrisiert und besitzen die Form

$$E_c\colon y^2 + (-c^3 - 2c^2 + 4c + 4)\,xy + (c+1)(c+2)(c^2+6c+4)c^3\,y = x^3 + (c+1)(c+2)c^3\,x^2$$

mit dem komplexen Parameter $c \in \mathbb{C} \setminus \left\{0, -1, -2, -\frac{1}{2} \pm \frac{1}{2}\sqrt{5}, -3 \pm \sqrt{5}\right\}$ und der Diskriminante

$$\operatorname{disc}(E_c) = -(c^2+6c+4)^2(c^2+c-1)(c+2)^{10}(c+1)^5c^{10}.$$

Der obige Quellcode *CalcWeilFunction10* führt die folgenden Schritte durch:

1. Berechnung der Weil-Funktion für den 10-Torsionspunkt $P = (0,0)$ auf der elliptischen Kurve E_c durch den Miller-Algorithmus 2.3.

2. Die vom Miller-Algorithmus gelieferte Weil-Funktion f_{10} liegt im Quotientenraum $\mathbb{Q}(x,y,c)$. Mit Hilfe der elliptischen Kurvengleichung $x^3 + a_2\,x^2 + a_4\,x + a_6 = y^2 + a_1\,xy + a_3\,y$ wird die Weil-Funktion im Ideal $\mathbb{Q}[x,y,c]/\langle y^2 + a_1\,xy + a_3\,y - x^3 - a_2\,x^2 - a_4\,x - a_6\rangle$ reduziert, so dass eine nennerfreie Darstellung $f_{10} \in \mathbb{Q}[x,y,c]$ entsteht.

3. Mit Hilfe der elliptischen Kurvengleichung lassen sich alle Potenzen x^m für $m \geq 3$ durch eine Linearkombination von 1, x und x^2 ersetzen:

$$\underbrace{x^m}_{\text{Grad } m} = x^{m-3}\,x^3 = \underbrace{x^{m-3}\,(-a_2\,x^2 - a_4\,x - a_6 + y^2 + a_1\,xy + a_3\,y)}_{\text{Grad } m-1}$$

4. Um eine kompakte Darstellung der Weil-Funktion zu erreichen, wird das Polynom f_{10} als Element von $\mathbb{Q}[c][x,y]$ angesehen und die Koeffizienten in $\mathbb{Q}[c]$ faktorisiert. Man erhält die Weil-Funktion

$$\begin{aligned} f_{10}(x,y) = 2(c^2-2c-2)\,y^3 &+ x^2y^2 - (2c+1)c^4\,xy^2 \\ &+ (c^3+16c^2+22c+8)c^6\,y^2 - (3c+2)c^6\,x^2y \\ &+ (c+1)^2c^{10}\,xy - (c+1)^2(c^2+6c+4)c^{12}\,y + (c+1)^2c^{12}\,x^2. \end{aligned}$$

Analoge Aufrufe können für elliptische Kurven mit n-Torsionspunkt für $n \in \{4,\dots,9\}$ verwendet werden.

A.2 Bibliotheksroutine *WeilFunction*

Die bisher vorhandene SAGE-Implementierung des Miller-Algorithmus wurde leicht umformuliert, um für eine natürliche Zahl $n \geq 2$ und eine elliptische Kurve E mit Punkt $P \in E$ eine rationale Funktion $f\colon E \to \overline{\mathbb{C}}$ in den Variablen x, y mit komplexem Parameter c zu bestimmen, die den Divisor

$$\operatorname{div}(f) = n\,[P] - [nP] - (n-1)\,[\mathcal{O}]$$

besitzt. Falls P ein n-Torsionspunkt der elliptischen Kurve E ist, hat f also den Divisor

$$\operatorname{div}(f) = n\,[P] - n\,[\mathcal{O}]$$

und ist damit eine Weil-Funktion von P. Der Miller-Algorithmus liefert zunächst ein Element des Quotientenraumes $\mathbb{Q}(x, y, c)$, welches im Ideal $\mathbb{Q}[x, y, c]/\langle y^2 + a_1\, xy + a_3\, y - x^3 - a_2\, x^2 - a_4\, x - a_6\rangle$ reduziert wird, so dass eine nennerfreie Darstellung $f \in \mathbb{Q}[x, y, c]$ entsteht:

```
# Vorlage: _miller_ in ell_point.py auf github.com/sagemath
# @param E .. elliptische Kurve; die Koeffizienten a1,a2,a3,a4 und a6
#             dürfen Parameter c enthalten
# @param P .. Punkt der elliptischen Kurve (Anwendungsfall: P ist
#             n-Torsionspunkt von E)
# @param n .. natürliche Zahl (Anwendungsfall: n ist die Ordnung von P)
# @return Gleichung für rationale Funktion f:E -> CC in den Variablen x, y
#         mit Parameter c mit Divisor div(f) = n[P] - [nP] - (n-1)[OO].
#         Falls P ein n-Torsionspunkt von E ist, hat f also den Divisor
#         div(f) = n[P] - n[OO].
def WeilFunction(E, P, n):

    if not P.curve() is E:
        raise ValueError("point must be on the elliptic curve")

    if P == E(0):
        raise ValueError("P cannot be the basepoint of the elliptic curve")

    if n.is_zero():
        raise ValueError("n must be nonzero.")

    n_is_negative = False
    if n < 0:
        n = n.abs()
        n_is_negative = True

    one = E.base_field().one()
    t = one
    V = P
    S = 2*V
    nbin = n.bits()
    i = n.nbits() - 2
    while i > -1:
        S = 2*V
        ell = EllipticLine(E, V, V)
        vee = EllipticLine(E, S, -S)
        t = (t**2)*(ell/vee)
        V = S
        if nbin[i] == 1:
            S = V + P
            ell = EllipticLine(E, V, P)
            vee = EllipticLine(E, S, -S)
            t = t*(ell/vee)
            V = S
        i = i-1

    if n_is_negative:
        vee = EllipticLine(E, V, -V)
        t = 1/(t*vee)

    # Im Quotientenring Q(x,y,c) reduzieren, damit eine nennerfreie
    # Darstellung entsteht
    return ReduceInQuotientRing(t, E)
```

A.3 Bibliotheksroutine *ReduceInQuotientRing*

Um ein Element des Quotientenrings $\mathbb{Q}(x, y, c)$ mit Hilfe einer elliptischen Kurvengleichung zu vereinfachen und im besten Fall als nennerfreies Polynom aus $\mathbb{Q}[x, y, c]$ darzustellen, wurde folgende Funktion implementiert:

```
# Reduziert den Ausdruck <term> im Quotienten-Ring modulo der Gleichung der
# elliptischen Kurve.
# @param E .. elliptische Kurve; Parameter a1,a2,a3,a4 und a6 dürfen c
#             enthalten
# @param term .. Element aus Q(x,y,c) = Quotient aus zwei Polynomen mit
#                Variablen x und y und Parameter c
# @returns term modulo Q(x,y,c)/I, wobei Q(x,y,c) = Polynom/Polynom ist
#          und I das Ideal der Gleichung der elliptischen Kurve E
def ReduceInQuotientRing(term, E):
    a1, a2, a3, a4, a6 = E.a_invariants();

    Q.<x,y,c> = QQ['x','y','c'];
    J = Q.ideal(y^2 + a1*x*y + a3*y - x^3 - a2*x^2 - a4*x - a6);
    R.<X,Y,C> = QuotientRing(Q, J);

    # In <term> werden die Variablen ersetzt gemäss x -> X, y -> Y, c -> C.
    termXYZ = sage_eval(str(term), locals={'x':X, 'y':Y, 'c':C});

    # Reduzieren im Ideal modulo der Gleichung der elliptischen Kurve.
    reduc = R(termXYZ).reduce(J.gens());

    # Rückbenennung X -> x, Y -> y, C -> c
    return sage_eval(str(reduc), locals={'X':x, 'Y':y, 'C':c});
```

Im Beispiel 2.4 wurde die rationale Funktion $f_5\colon E_5 \to \overline{\mathbb{C}}$, $f_5(x, y) = \frac{y^2(cx+y)}{(x+c)^2}$ mit Hilfe der elliptischen Kurvengleichung $E_5\colon y^2 + (c+1)xy + cy = x^3 + cx^2$ durch die formale Erhöhung des Zählers um $(-x+y-c)(x^3+cx^2-y^2-(c+1)xy-cy)$ in die nennerfreie Darstellung $f_5(x, y) = -x^2+xy+y$ gebracht. Diese Operation lässt sich in SAGE wie folgt durchführen:

```
x, y, c = var('x y c');
E_5 = EllipticCurve([c + 1, c, c, 0, 0]);
ReduceInQuotientRing(y^2 * (c * x + y) / (x + c)^2, E_5);
--> -x^2 + x*y + y
```

A.4 Bibliotheksroutine *EllipticLine*

Die in SAGE enthaltene Funktion `_line_` wurde leicht modifiziert, um die Gleichung der Gerade zu bestimmen, die durch zwei Punkte $Q, R \in E$ einer elliptischen Kurve E geht. Für $Q = R$ wird die Gleichung der Tangente an E im Punkt Q zurückgegeben:

```
# Bestimmt die Gleichung der Gerade durch die Punkte Q und R, die auf der
# elliptischen Kurve E liegen.
# Vorlage: _line_ in ell_point.py auf github.com/sagemath
# @param E   .. elliptische Kurve; die Koeffizienten a1,a2,a3,a4 und a6
#               dürfen Parameter c enthalten
# @param Q,R .. Zwei Punkte der elliptischen Kurve E
def EllipticLine(E, Q, R):
    if (Q == E(0)) or (R == E(0)):
        if Q == R:
            return E.base_field().one()
```

```
        if Q == E(0):
            return x - R[0]
        if R == E(0):
            return x - Q[0]
    elif Q != R:
        if Q[0] == R[0]:
            return x - Q[0]
        else:
            m = (R[1] - Q[1])  / (R[0] - Q[0])
            return y - Q[1] - m * (x - Q[0])
    else:
        a1, a2, a3, a4, a6 = E.a_invariants()
        numerator = (3*Q[0]**2 + 2*a2*Q[0]+a4-a1*Q[1])
        denominator = (2*Q[1] + a1*Q[0] + a3)
        if denominator == 0:
            return x - Q[0]
        else:
            m = numerator / denominator
            return y - Q[1] - m * (x - Q[0])
```

A.5 Bibliotheksroutinen *ReplaceHighXMonoms*, *XPower*, *ReplaceSubTerm*

Um in einem Polynom aus $\mathbb{Q}[x, y, c]$ alle Vorkommnisse von x^n für $n \geq 3$ durch Linearkombinationen aus x^2, x und 1 zu ersetzen, kann die elliptische Kurvengleichung in der folgenden Form verwendet werden:

$$\underbrace{x^n}_{\text{Grad } n} = x^{n-3} \cdot x^3 = \underbrace{x^{n-3}\left(-a_2\, x^2 - a_4\, x - a_6 + y^2 + a_1\, xy + a_3\, y\right)}_{\text{Grad } n-1}.$$

Der Quellcode führt die Ersetzung von x^n iterativ durch und beginnt mit der höchsten x-Potenz:

```
# Drückt x^n für n >= 3 durch kleinere x-Potenzen aus.
# @param E   .. elliptische Kurve; die Koeffizienten a1,a2,a3,a4 und a6
#               dürfen Parameter c enthalten
# @param n   .. natürliche Zahl
def XPower(n, E):
    a1, a2, a3, a4, a6 = E.a_invariants();

    if n <= 2:
        return x^n
    if n >= 3:
        return expand(x^(n-3) * (y^2 + a1*x*y + a3*y - a2*x^2 - a4*x - a6))

# Ersetzt in <term> alle Vorkommnisse von <search> durch (<repl>).
# Unterstützt werden die Variablen x, y und c.
def ReplaceSubTerm(term, search, repl):
    term_str = str(term);
    term_repl = term_str.replace(str(search), "(" + str(repl) + ")");
    return sage_eval(term_repl, locals={'x':x, 'y':y, 'c':c})

# Ersetzt in <term> alle Vorkommnisse von <x^n> durch Potenzen x^2, x, 1.
# @param E    .. elliptische Kurve; die Koeffizienten a1,a2,a3,a4 und a6
#                dürfen Parameter c enthalten
# @param term .. Polynom in x, y und c.
def ReplaceHighXMonoms(term, E):
    # Polynomring als "Monom-Wörterbuch"
    R.<x,y,c> = sage.rings.polynomial.multi_polynomial_ring.
                MPolynomialRing_polydict(QQ, 3, order='lex');
```

```
        # term als Polynom in x, y, c interpretieren
        poly = R(term)

        # Maximalgrad von x im Polynom <term> bestimmen.
        max_x_degree = poly.degree(x)

        # Iteration von max_x_degree bis 3
        for e in range(max_x_degree, 2, -1):
            # Ersetze x^e durch kleinere Potenzen.
            term = ReplaceSubTerm(term, x^e, XPower(e, E));
            # Ausmultiplizieren, damit im nächsten Schritt x^(e-1) ersetzt
            # werden kann.
            term = expand(term);
        return term;
```

A.6 Bibliotheksroutine *FactorCoefficientsXYPoly*

Die nachfolgende Routine interpretiert ein Polynom aus $\mathbb{Q}[x, y, c]$ als Element von $\mathbb{Q}[c][x, y]$ und faktorisiert die Koeffizienten in $\mathbb{Q}[c]$:

```
# Interpretiert <term> als Polynom in x und y und faktorisiert die
# Koeffizienten dieses Polynoms.
# Rückgabe erfolgt als String.
# @param term .. Polynom in x, y und c
def FactorCoefficientsXYPoly(term):
    # Polynomring als "Monom-Wörterbuch"
    R.<x,y,c> = sage.rings.polynomial.multi_polynomial_ring.
                MPolynomialRing_polydict(QQ, 3, order='lex');

    # term als Polynom in x, y, c interpretieren
    poly = R(term)

    # Höchsten Grad von x und y bestimmen
    degree_x = poly.degree(x)
    degree_y = poly.degree(y)

    # Leeres Polynom und leeren Polynom-String anlegen.
    new_poly = 0
    new_poly_str = ""

    # Über alle Monome x^a y^b iterieren
    for b in range(degree_y, -1, -1):
        for a in range(degree_x, -1, -1):
            # Koeffizient von x^a y^b lesen
            coeff = poly.coefficient({x:a, y:b})

            # Koeffizient faktorisieren, wenn nicht 0.
            if (coeff != 0):
                coeff = factor(coeff);

                # Neues Polynom um Monom x^a y^b mit faktorisiertem
                # Koeffizienten erweitern.
                new_poly = new_poly + coeff * x^a * y^b;

                # Polynomstring um Monom x^a y^b mit faktorisiertem
                # Koeffizienten erweitern
                new_poly_str = new_poly_str + "+ (" + str(coeff) + ") * "
                               + str(x^a * y^b)
```

```
                # Übersichtliche formatierte Konsolenausgabe erzeugen
                print "+ (", coeff, ") * ", x^a * y^b

        # Sicherstellen, dass die beiden Polynome übereinstimmen.
        diff = term - sage_eval(new_poly_str, locals={'x':x, 'y':y, 'c':c})
        if diff != 0:
            print "Error: Wrong polynomial calculated. Diff = ", diff

        return new_poly_str
```

A.7 Bibliotheksroutine *CalcWeilFunction12*

Um die Weil-Funktion für elliptische Kurven mit 12-Torsion zu bestimmen, müssen Zwischenergebnisse im Miller-Algorithmus zusammengefasst werden, um die große Anzahl von Monomen verarbeiten zu können:

```
    # Berechnet die Weil-Funktion für X_1(12), reduziert die Potenzen x^3 und
    # höher bzgl. der Gleichung der elliptischen Kurve und faktorisiert die
    # Koeffizienten, um eine kompakte Darstellung der Weil-Funktion zu erhalten.
    def CalcWeilFunction12():
        # Elliptische Kurve X_1(12) und 12-Torsionspunkt P definieren
        x, y, c = var('x y c');
        E_12 = EllipticCurve([6*c^4 - 8*c^3 + 2*c^2 + 2*c - 1,
                              -c*(c-1)^2*(2*c-1)*(2*c^2-2*c+1)*(3*c^2-3*c+1),
                              -c*(c-1)^5*(2*c-1)*(2*c^2-2*c+1)*(3*c^2-3*c+1),
                              0, 0]);
        P = E_12(0, 0);

        # Miller-Algorithmus durchführen
        f1 = 1;
        f2 = f1^2 * EllipticLine(E_12, P, P) / EllipticLine(E_12, 2*P, -2*P);
        f2 = factor(f2);
        f3 = f2 * EllipticLine(E_12, 2*P, P) / EllipticLine(E_12, 3*P, -3*P);
        f3 = factor(f3);
        f6 = f3^2 * EllipticLine(E_12, 3*P, 3*P) / EllipticLine(E_12, 6*P,-6*P);
        f6 = factor(f6);
        f12a = f6^2 * EllipticLine(E_12, 6*P, 6*P);
        f12a = factor(f12a);
        f12 = f12a / EllipticLine(E_12, 12*P, -12*P);
        f12 = factor(f12);

        # Im Quotientenring reduzieren, damit eine nennerfreie Darstellung
        # entsteht
        f = ReduceInQuotientRing(f12, E_12);

        # Potenzen x^n für n >= 3 ersetzen durch x^2, x, 1
        f = ReplaceHighXMonoms(f, E_12)

        # f als Polynom in x,y sehen und Koeffizienten faktorisieren
        f_str = FactorCoefficientsXYPoly(f)

        return f_str;
```

B SAGE-Datenbank für berechnete Weil-Funktionen

Es wurde eine SAGE-Datenbank erstellt, in der die berechneten Weil-Funktionen abgelegt wurden. Für jeden Wert $n \in \{4, \ldots, 10, 12\}$ werden die folgenden Informationen bereitgestellt:

- Gleichung der elliptischen Kurve $X_1(n)$
- n-Torsionspunkt $P = (0,0)$
- Weil-Funktion f_n mit Divisor $\operatorname{div}(f_n) = n\,[P] - n\,[\mathcal{O}]$
- Wohlformatierte String-Repräsentation der Weil-Funktion

B.1 Konstruktor

```
class WeilObject(object):
    # Konstruktor: Erzeugt eine elliptische Kurve inklusive Weil-Funktion.
    # @param n .. Natürliche Zahl aus dem Bereich {4,...,10,12}
    # self._E = Elliptische Kurve X_1(n) in den Variablen x, y und Parameter c
    # self._P = n-Torsionspunkt (0,0) auf E
    # self._n = natürliche Zahl n
    # self._f = Weil-Funktion E -> C mit Divisor div(f)=n[P] - n[OO]
    # self._s = Wohlformatierter String der Weil-Funktion f
    #
    def __init__(self, n):
        self._n = n
        x, y, c = var('x y c');

        if n == 4:
            self._E = EllipticCurve([1, c, c, 0, 0]);
            self._P = self._E(0,0);
            self._f = x^2 - y;
            self._s = "x^2 - y";
            return;

        if n == 5:
            self._E = EllipticCurve([c+1, c, c, 0, 0]);
            self._P = self._E(0,0);
            self._f = -x^2 + x*y + y;
            self._s = "-x^2 + x*y + y";
            return;

        if n == 6:
            self._E = EllipticCurve([1-c, -c*(c+1), -c*(c+1), 0, 0]);
            self._P = self._E(0,0);
            self._f = y^2 - (c + 1)*x*y - (c + 1)^2*y + (c + 1)^2*x^2;
            self._s = "y^2 - (c + 1)*x*y - (c + 1)^2*y + (c + 1)^2*x^2";
            return;

        if n == 7:
            self._E = EllipticCurve([1-c-c^2, c^2*(c+1), c^2*(c+1), 0, 0]);
            self._P = self._E(0,0);
            self._f = (c - 1)*y^2 + x^2*y - c^3*x*y + c^4*y - c^4*x^2;
            self._s = "(c - 1)*y^2 + x^2*y - c^3*x*y + c^4*y - c^4*x^2";
            return;

        if n == 8:
            self._E = EllipticCurve([1-2*c^2, -c*(2*c+1)*(c+1)^2,
                                    -c*(2*c+1)*(c+1)^3, 0, 0]);
            self._P = self._E(0,0);
            self._f = (x*y^2 + (2*c + 3)*(c + 1)^3*y^2 - 2*(c + 1)^2*x^2*y
                      - (2*c + 1)^2*(c + 1)^4*x*y - (2*c + 1)^2*(c + 1)^7*y
                      + (2*c + 1)^2*(c + 1)^6*x^2);
            self._s = ...
            return;
```

```
        if n == 9:
            self._E = EllipticCurve([c^3 + c^2 + 1, c^2 * (c+1) * (c^2+c+1),
                                     c^2 * (c+1) * (c^2+c+1), 0, 0]);
            self._P = self._E(0,0);
            self._f = (y^3 + (c - 1)*(c^2 + c + 1)*x*y^2
                       + (c^2 + c + 1)^2*(c^3 + 2*c - 1)*y^2
                       - (2*c - 1)*(c^2 + c + 1)^2*x^2*y
                       + c^4*(c^2 + c + 1)^3*x*y
                       + c^4*(c^2 + c + 1)^4*y - c^4*(c^2 + c + 1)^4*x^2);
            self._s = ...
            return;

        if n == 10:
            self._E = EllipticCurve([-c^3 - 2*c^2 + 4*c + 4, (c+1) * (c+2) * c^3,
                                     (c+1) * (c+2) * (c^2+6*c+4) * c^3, 0, 0]);
            self._P = self._E(0,0);
            self._f = (2*(c^2 - 2*c - 2)*y^3 + x^2*y^2 - (2*c + 1)*c^4*x*y^2
                       + (c^3 + 16*c^2 + 22*c + 8)*c^6*y^2 - (3*c + 2)*c^6*x^2*y
                       + (c + 1)^2*c^10*x*y - (c + 1)^2*(c^2 + 6*c + 4)*c^12*y
                       + (c + 1)^2*c^12*x^2);
            self._s = ...
            return;

        if n == 12:
            self._E = EllipticCurve([6*c^4 - 8*c^3 + 2*c^2 + 2*c - 1,
                      -c * (c-1)^2 * (2*c-1) * (2*c^2-2*c+1) * (3*c^2-3*c+1),
                      -c * (c-1)^5 * (2*c-1) * (2*c^2-2*c+1) * (3*c^2-3*c+1),
                      0, 0]);
            self._P = self._E(0,0);
            self._f = (y^4
              + (6*c^2-8*c+3)*(2*c^2-2*c+1)*x*y^3
              + (c-1)^4*(2*c^2-2*c+1)^2*(36*c^4-90*c^3+96*c^2-49*c+10)*y^3
              + 3*(c-1)^2*(5*c^2-6*c+2)*(2*c^2-2*c+1)^2*x^2*y^2
              + (c-1)^4*(14*c^2-16*c+5)*(3*c^2-3*c+1)^2*(2*c^2-2*c+1)^3*x*y^2
              + (c-1)^8*(3*c^2-3*c+1)^2*(2*c^2-2*c+1)^4
                       *(12*c^4-42*c^3+57*c^2-33*c+7)*y^2
              + 2*(c-1)^6*(9*c^2-10*c+3)*(3*c^2-3*c+1)^2*(2*c^2-2*c+1)^4*x^2*y
              + (2*c-1)^2*(c-1)^8*(3*c^2-3*c+1)^4*(2*c^2-2*c+1)^5*x*y
              - (2*c-1)^2*(c-1)^13*(3*c^2-3*c+1)^4*(2*c^2-2*c+1)^6*y
              + (2*c-1)^2*(c-1)^10*(3*c^2-3*c+1)^4*(2*c^2-2*c+1)^6*x^2);
            self._s = ...
            return;

        # Assert im Konstruktor auslösen, damit kein Objekt für
        # nicht unterstützte n-Werte erzeugt wird
        assert false;
```

B.2 String-Repräsentation, String-Ausgabe, Evaluation

```
    # Gibt die String-Repräsentation des Objektes zurück.
    def __repr__(self):
        str="Weil function info object of elliptic curve with "
        str=str+ "%i-torsion point (0,0):\n- "%(self._n)
        str=str+ self._E.__repr__() + "\n"
        str=str+ "- %i-torsion point P = "%(self._n) + self._P.__repr__() + "\n"
        str=str+ "- Weil function f(x,y) = " + self._f.__repr__() + "\n"
        return str
```

```
    # Gibt die Weil-Funktion als wohlformatierten String auf der Konsole aus.
    def PrintNiceString(self):
        print self._s

    # Gibt den Wert der Weilfunktion f im Punkt (xval, yval) zurück.
    def f(self, xval, yval):
        return self._f(x = xval, y = yval)
```

C Verifikation der Weil-Funktionen mit SAGE

Die mit dem Miller-Algorithmus berechneten und in der Datenbank abgelegten Weil-Funktionen sollen verifiziert werden, indem eine charakteristische Eigenschaft der Weil-Funktionen nachgewiesen wird.

C.1 Berechnung von α^n

Der nachfolgende Quellcode bestimmt für jede Weil-Funktion f_n die Konstante α^n, für die die Beziehung

$$f_n(x, y) \cdot f_n(x, -a_1x - y - a_3) = \alpha^n x^n$$

für alle Punkte (x, y) der elliptischen Kurve gilt. Die Existenz von $\alpha \in \mathbb{C}$ wurde in Theorem 4.1(c) nachgewiesen.

```
    # Bestimmt die komplexe Zahl alpha^n, die die folgende Eigenschaft besitzt:
    # f((x,y)) * f(-(x,y)) = alpha^n * x^n, wobei mit -(x,y) das Inverse bzgl.
    # der Addition auf der elliptischen Kurve gemeint ist.
    # -> Wenn alles korrekt ist, muss alpha^n = (-1)^n gelten.
    #
    def CalcAlphaHochN(self):
        # Minus-Funktion g der Weil-Funktion berechnen, g(Q) = f(-Q) für alle Q
        g(x, y) = self.GetMinusWeilFunction()

        # Produkt aus Weil-Funktion und Weil-Minus-Funktion bestimmen
        p(x, y) = self.f(x, y) * g(x, y)

        # alpha^n berechnen, indem spezieller Wert x = 1 eingesetzt wird
        alpha_hoch_n = p(x = 1, y = y)

        # ausmultiplizieren
        alpha_hoch_n = expand(alpha_hoch_n)

        # Potenzen y^m für m >= 2 ersetzen durch Linearkombination aus 1 und y;
        # die Variable y muss aus dem Term komplett herausfallen, so dass eine
        # Konstante übrig bleibt
        alpha_hoch_n = self.ReplaceHighYMonomsX1(alpha_hoch_n)

        # Den Wert (alpha^n) zurückgeben.
        return alpha_hoch_n
```

C.2 Verifikation der charakteristischen Eigenschaft

Um zu ermitteln, ob die Bibliothek die korrekten Weil-Funktionen enthält, wird die Gültigkeit der charakteristischen Eigenschaft

$$f_n(Q) \cdot f_n(-Q) = \alpha^n x^n \quad \text{für alle Punkte } Q = (x, y) \in E_n$$

überprüft. Die einzelnen Schritte dieses Algorithmus sind im Beweis von Theorem 4.2 erläutert.

```
# Überprüft die Integrität des Objektes:
# - Ist P wirklich ein n-Torsionspunkt von E ?
# - Gilt wirklich alpha^n = (-1)^n ?
# - Gilt wirklich f((x,y)) * f(-(x,y)) = alpha^n * x^n ?
def Verify(self):
    # 1. Ist P wirklich ein n-Torsionspunkt von E ?
    assert (self._n * self._P == self._E(0));

    # 2. Gilt wirklich alpha^n = (-1)^n ?
    alpha_hoch_n = self.CalcAlphaHochN();
    assert (alpha_hoch_n == (-1)^self._n);

    # 3. Gilt wirklich f((x,y)) * f(-(x,y)) = alpha^n * x^n ?

    # 3a. Minus-Funktion g der Weil-Funktion berechnen, g(Q) = f(-Q)
    g(x, y) = self.GetMinusWeilFunction()

    # 3b. Produkt aus Weil-Funktion und Weil-Minus-Funktion bestimmen
    p(x, y) = self.f(x, y) * g(x, y)

    # 3c. Differenz p - (alpha^n * x^n) im Ideal reduzieren
    diff = p(x,y)-(alpha_hoch_n * x^self._n)
    diff = self.ReduceInQuotientRing(diff)

    # 3d. Differenz muss Null sein
    assert (diff == 0);

    # 4. Alles OK.
    return true;
```

Die eigentliche Verifikation erfolgt durch Aufruf dieser Methode *Verify*:

```
WeilObject(4).Verify();
WeilObject(5).Verify();
WeilObject(6).Verify();
WeilObject(7).Verify();
WeilObject(8).Verify();
WeilObject(9).Verify();
WeilObject(10).Verify();
WeilObject(12).Verify();
```

C.3 Routine *GetMinusWeilFunction*

Die nachfolgende Routine bestimmt das Bild des additiven Inversen $-(x, y) = (x, -a_1x - a_3 - y)$ eines Punktes (x, y) unter der Weil-Funktion einer elliptischen Kurve $E\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$.

```
# Gibt f(-(x,y)) zurück, wobei mit -(x,y) das Inverse bzgl. der Addition
# auf der elliptischen Kurve gemeint ist.
def GetMinusWeilFunction(self):
    a1, a2, a3, a4, a6 = self._E.a_invariants();
    return self.f(x, -a1 * x - a3 - y);
```

C.4 Routinen *YPowerX1*, *ReplaceHighYMonomsX1*

Die nachfolgenden Routinen ersetzen Potenzen y^m für $m \geq 2$ mit Hilfe der elliptischen Kurvengleichung gemäß

$$\underbrace{y^m}_{\text{Grad } m} = y^{m-2} \cdot y^2 = \underbrace{y^{m-2}(-a_1 y - a_3 y + a_2 + a_4 + a_6 + 1)}_{\text{Grad } m-1}.$$

Dies geschieht unter der Einschränkung, dass $x = 1$ gilt.

```
# Drückt y^m für m >= 2 durch kleinere y-Potenzen aus, FALLS x = 1 gilt.
def YPowerX1(self, m):
    a1, a2, a3, a4, a6 = self._E.a_invariants();
    if m < 2:
        return y^m
    if m >= 2:
        return expand(y^(m-2) * ((-a1-a3)* y + a2 + a4 + a6 + 1)) # x = 1 !

# Ersetzt in <term> alle Vorkommnisse von <y^m> durch Potenzen y^2, y, 1,
# FALLS x = 1 gilt.
# Parameter term: Polynom in y und c.
def ReplaceHighYMonomsX1(self, term):
    # Polynomring als "Monom-Wörterbuch"
    R.<x,y,c> = sage.rings.polynomial.multi_polynomial_ring
                    .MPolynomialRing_polydict(QQ, 3, order='lex');

    # term als Polynom in y, c interpretieren
    poly = R(term)

    # Maximalgrad von y im Polynom <term> bestimmen.
    max_y_degree = poly.degree(y)

    # Iteration von max_y_degree bis 2
    for e in range(max_y_degree, 1, -1):
        # Ersetze y^e durch kleinere Potenzen.
        term = self.ReplaceSubTerm(term, y^e, self.YPowerX1(e));
        # Ausmultiplizieren, damit im nächsten Schritt y^(e-1)
        # ersetzt werden kann.
        term = expand(term);

    return term;
```

Literatur

[Mil04] Victor S. Miller, *The Weil pairing and its efficient calculation*, J. Cryptology **17** (2004), pp. 235–261

[SAGE] The Sage Notebook v7.0, Computer-Algebra-System, `http://www.sagenb.org/`

[Sil86] Joseph H. Silverman, *The Arithmetic of Elliptic Curves*, Springer Verlag, New York, 1986

[Wei40] André Weil, *Sur les fonctions algebriques à corps de constantes finis*, C. R. Acad. Sci. Paris, **210** (1940), pp. 592–594 (enthalten in Oeuvres Scientifiques, Volume I, pp. 257–259)