Universität Bremen
FB 3 - Institut für Mathematik

Wintersemester 2010 / 2011
Seminar der WE AlZAGK:
Faktorisierung und diskreter Logarithmus
Betreuer: Prof. Jens Gamst

Schriftliche Ausarbeitung zum Thema:

Faktorisierung mit Elliptischen Kurven

Isabelle Beckmann

icmb@gmx.de

Inhaltsverzeichnis

1 Einleitung

Bereits seit Euklid ist bekannt, dass jede natürliche Zahl eine eindeutige Primfaktorzerlegung besitzt. Dieses Problem ist also seit der Antike gelöst.

Die Frage, die sich direkt daraus ergibt, ist allerdings: Wie bekommt man diese Primfaktoren im konkreten Fall?
Um die Antwort auf diese Frage geht es in der vorliegenden Arbeit.

Es wurden einige verschiedene Faktorisierungsmethoden im Laufe der Zeit entwickelt.
Unter dem Schlagwort *klassische Faktorisierungsmethoden* fasse ich im Folgenden vor allem drei zusammen: Als erstes die naive Division der zu faktorisierenden Zahl durch Primzahlen unter einer bestimmten Schranke, als zweites die (p-1)- und als drittes die (p+1)-Methode.
All diese Verfahren liefern für relativ kleine Zahlen gute Ergebnisse, allerdings versagen sie bei größeren Zahlen.

Im Jahre 1987 veröffentlichte der Niederländische Mathematiker Hendrik Lenstra (Jr.) in den *Annals of Mathematics* einen Artikel mit dem Titel „Factoring Integers with Elliptic Curves“.[1]
Das Verfahren, das er dort vorstellte, sollte einen deutlichen Fortschritt in den Möglichkeiten der Faktorisierung bringen. Genau dieses Verfahren soll in der vorliegenden Arbeit behandelt werden.

Im Abschnitt 2 wiederhole ich kurz einige Voraussetzungen, die in den folgenden Teilen der Arbeit benutzt werden, beziehungsweise als bekannt vorausgesetzt werden.

Die Abschnitte 3 und 4 bilden den Hauptteil dieser Arbeit.

Im 3. Abschnitt wird die eigentliche Methode vorgestellt und anschließend mit Leben gefüllt. Das heißt, ich beschreibe, wie man die benötigten *zufälligen Elliptischen Kurven* finden / konstruieren kann und stelle danach zwei Beispiele für konkrete Faktorisierungen vor. Anschließend mache ich einige Bemerkungen dazu, wann und warum Lenstras Faktorisierungsmethode gut funktioniert.

Im 4. Abschnitt geht es dann darum, die neue Methode mit den klassischen Methoden in einen Zusammenhang zu stellen. Dazu betrachte ich, was geschieht, wenn man sie nicht auf beliebige Elliptische Kurven, sondern auf singuläre anwendet.

Abschließend folgt noch eine Zusammenfassung, in der ich auch kurz auf Optimierungsmöglichkeiten des Verfahrens eingehe.

Da für das zweite Beispiel in Abschnitt 3 das Computerprogramm „Aribas“ [2] benutzt wurde, stelle ich die benutzen Codes im Anhang zusammen.

Zur besseren Lesbarkeit kürze ich „Lenstras Elliptische Kurven Faktorisierungsmethode“ mit „ECM“ ab.

[1] vgl. [Le] im Literaturverzeichnis
[2] vgl. [Fo2]

2 Vorbemerkungen

Im Folgenden möchte ich kurz auf einige Voraussetzungen eingehen, die ich im Laufe der Arbeit benötige.

2.1 Elliptische Kurven

2.1.1 Allgemeine Einführung

Eine Elliptische Kurve E ist definiert über die Weierstraß-Gleichung

$$y^2 = x^3 + Ax + C,$$

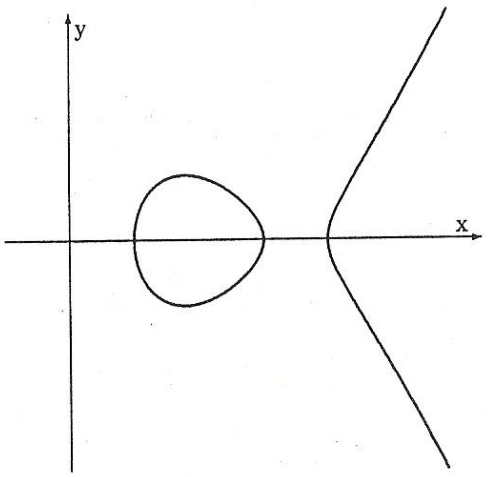
wobei $A, C \in \mathbb{K}$ und $\mathbb{K}$ ein Körper einer Charakteristik $\neq 2, 3$ sind.

Zusätzlich wird die sogenannte Diskriminantenbedingung gestellt, damit E singularitätenfrei ist:

$$\Delta := 4A^3 + 27C^2 \neq 0$$

Hiermit ist also gewährleistet, dass sich die Kurve nicht selbst schneidet und damit schneidet jede Gerade die Kurve in 3 Punkten.

Zur Veranschaulichung ein Bild einer elliptischen Kurve[3]:



2.1.2 Gruppenstruktur

Wir setzen im Folgenden voraus, dass die y-Koordinate der Punkte ungleich 0 ist.
Da dieser Sonderfall für unsere Faktorisierungsmethode vermeidbar ist, betrachten wir den Fall hier nicht und schließen ihn aus.

Betrachtet man die Elliptischen Kurven als projektive Kurven, gewinnt man den unendlich fernen Punkt $\mathfrak{O}$ hinzu. Durch eine geometrische Konstruktion kann dann auf der Menge der Lösungen der Elliptischen Kurven $E(\mathbb{K})$ und $\mathfrak{O}$ eine Operation definiert werden, die eine abelsche Gruppe bildet.
Die Vorschrift dafür lautet:

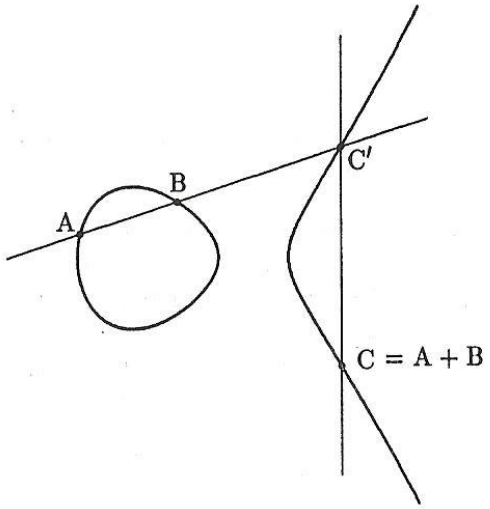
1. Das Nullelement von E sei der unendlich ferne Punkt $\mathfrak{O} = (0 : 0 : 1)$.
2. Wenn E von einer Gerade in den drei Punkten A, B, C geschnitten wird, dann sei

$$A + B + C = \mathfrak{O}.$$

[3]Alle Bilder dieses Abschnittes 2.1 stammen aus [Fo1], § 19

Auf einen Beweis, dass $(E(\mathbb{K}) \cup \mathfrak{O}, +)$ tatsächlich eine abelsche Gruppe definiert, möchte ich an dieser Stelle verzichten.

Ausgeführt[4] heißt das, dass die Addition $A + B = C$ wie folgt konstruiert wird:
Lege eine Gerade durch A und B (falls A=B ist, nehme man die Tangente an A), diese schneidet dann E in einem weiteren Punkt C'. Dann ist C die Spiegelung von C' an der x-Achse und C' ist das Inverse von C.



2.1.3 Explizite Formeln zur Addition von Punkten

Aus der in 2.1.2 geschilderten geometrischen Konstruktion der Addition kann man explizite Formeln für die Addition ableiten. Dabei bezeichnen (x_i, y_i) die Koordinaten der Punkte und m die Steigung:

Es sei $P = (x_1, y_1)$ ein Punkt auf E.

Dann ist

$$2 \cdot P = (x_3, y_3)$$

mit:

$$\begin{aligned} m &= \frac{3x_1^2 + A}{2y_1} \\ x_3 &= m^2 - 2x_1 \\ y_3 &= m(x_1 - x_3) - y_1 \end{aligned}$$

Für $P_1 \neq P_2$ ist

$$P_1 + P_2 = (x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

mit:

$$\begin{aligned} m &= \frac{y_2 - y_1}{x_2 - x_1} \\ x_3 &= m^2 - x_1 - x_2 \\ y_3 &= m(x_1 - x_3) - y_1 \end{aligned}$$

[4] Auf eine detaillierte Ausführung möchte ich an dieser Stelle aus Platz- und Gewichtungsgründen verzichten.

2.2 Klassische (p-1)-Faktorisierungsmethode

Noch einmal zur Erinnerung die Grundidee der (p-1)-Faktorisierungsmethode von Pollard:

Wähle eine zu faktorisierende Zahl $n \in \mathbb{Z}$.

Wähle dann ein beliebiges zu n teilerfremde $a \in \mathbb{Z}$ und berechne

$$a^k \mod n.$$

Wenn dann bereits

$$a^k \not\equiv 1 \mod n$$

gilt, folgt daraus, wegen

$$ggT\left(a^k - 1, n\right) \mid n,$$

offenbar auch:

$$ggT\left(a^k - 1, n\right) < n \mod n$$

Weiterhin hofft man, dass $ggT\left(a^k - 1, n\right) > 1$ ist, damit man einen echten Teiler von n gefunden hätte. Jenes erreicht man, indem k geschickt gewählt wird.

Falls p ein Primteiler von n und zusätzlich $a^k \equiv 1 \mod p$ ist, dann gilt:

$$p \mid ggT\left(a^k - 1, n\right)$$

Nun weiß man, dass

$$\left|(\mathbb{Z}/p)^\times\right| = p - 1 = \prod_{i=1}^{r} q_i^{e_i}$$

ist, wobei der letzte Term die Primfaktorzerlegung von $p-1$ bezeichnen soll.

Man wähle jetzt eine Schranke $B \in \mathbb{Z}$ mit

$$B \geq q_i^{e_i} \quad \forall q_i.$$

Man berechnet nun

$$a^{B!} \mod n.$$

Offenbar folgt aus der Wahl von B:

$$a^{B!} \equiv 1 \mod p$$

und man ist fertig, wenn zusätzlich gilt:

$$a^{B!} \not\equiv 1 \mod n$$

Der gesuchte Primfaktor von n wird dann gegeben durch:

$$ggT\left(a^{B!} - 1, n\right) \mid n$$

Da wir p nicht im Vornherein kennen, sind wir auf die geschickte Wahl von B angewiesen und darauf, dass $p-1$ aus kleinen Primpotenzfaktoren zusammengesetzt wird. Ist dies der Fall, funktioniert die (p-1)-Faktorisierungsmethode sehr gut. Im anderen Fall funktioniert das Verfahren leider nicht.

2.3 Big Prime Variation

Für den Fall, dass

$$a^k := \quad a^{B!} \not\equiv 1 \qquad \mod p$$

ist, scheitert das (p-1)-Faktorisierungsverfahren, wie in 2.2 gezeigt wurde. Allerdings gibt es eine Möglichkeit, doch noch zum Ziel zu kommen, indem man die sogenannte „Big Prime Variation“ benutzt.

Diese basiert auf der Tatsache,[5] dass $a^k \mod p$ Element einer Untergruppe X von $(\mathbb{Z}/p)^\times$ ist. Deren Ordnung s ist gegeben durch:

$$s := |X| = \frac{p-1}{ggT(k, p-1)}$$

Da $k = B!$ gesetzt wurde, enthält k offensichtlich alle Primzahlpotenzen unterhalb der Schranke B. Daraus folgt, dass es sehr wahrscheinlich ist, dass s eine Primzahl größer als B ist, da k auch alle Primzahlpotenzen aus $(p-1)$ enthält.
(Falls $p-1$ eine Primzahlpotenz $q^t > B$ enthält, mit $q \leq B$, kann s keine Primzahl größer als B sein.)

Wenn dieser wahrscheinlichere Fall eintritt, kann die (p-1)-Methode durch die sogenannte Big Prime Variation erweitert werden:

Es wird nun also vorausgesetzt, dass $s > B$ eine Primzahl ist.

Dann folgt daraus direkt, dass

$$\left(a^k\right)^s \equiv 1 \qquad \mod p$$

ist, also auch:

$$p \quad | \quad a^{ks} - 1$$

Und da sowieso

$$ggT(a^{ks} - 1, n) \quad | \quad n$$

gilt, haben wir unseren gesuchten Faktor p von n gefunden .

Das bedeutet zusammengefasst, dass man bei der Big Prime Variation alle Primzahlen s unterhalb einer neuen Grenze $C > s$ mit $s > B$ durchprobiert, um den gesuchten Faktor von n zu finden.

[5] Diese möchte ich an dieser Stelle ohne Beweis zitieren. Der Beweis ist beispielsweise bei [Fo1], § 14, Satz 14.1 nachzulesen.

3 Elliptische Kurven Faktorisierungsmethode

3.1 Beschreibung der Methode

Die Anfangsproblematik lautet: Wir haben ein ungerades $n \in \mathbb{Z}_+$ gegeben, von dem wir (beispielsweise durch verschiedene Primzahltests) wissen, dass n zusammengesetzt ist. Wir suchen einen Primfaktor von n.

Zuerst beschreibe ich die allgemeine Methode von Lenstra:

1. Wähle eine bestimmte Anzahl m von zufälligen Elliptischen Kurven E_i, $i \in \{1, \ldots, m\}$:

   $$y^2 = x^3 + A_i x + C_i \qquad \mod n$$

   und Punkte auf diesen Kurven

   $$P_i \qquad \mod n$$

2. Wähle eine Schranke $B \in \mathbb{Z}$ (in der Größenordnung 10^8).[6]
   Berechne

   $$(B!)P_i \quad \text{auf} \quad E_i \quad \forall i \in \{1, \ldots, m\}$$

3. Falls eine Steigung $\mod n$ bei der Berechnung der Punktvielfachheiten nicht existiert, missglückt Schritt 2 und dann haben wir einen Faktor von n gefunden.
   Es ist dann bei sehr günstiger Wahl[7] von B bereits: [8]

   $$ggT(\text{Nenner}, n) = p$$

   Im Allgemeinen gilt aber nur:

   $$ggT(\text{Nenner}, n) = f,$$

   wobei f hier einen echten Faktor der Ordnung von $E(\mathbb{F}_p)$ bezeichnen soll. Folgend wäre also wiederum für f zu prüfen, ob es eine Primzahl ist und wenn nicht, diese Zahl zu faktorisieren.

4. Falls Schritt 2 glückt, dann gibt es zwei Möglichkeiten:

   (a) Erhöhe B oder

   (b) wähle neue Kurven E_i und Punkte P_i und beginne von vorn.

Führt man diese Faktorisierung auf einem Computer durch, kann man Schritt 2 für alle Elliptischen Kurven E_i parallel behandeln.

[6] Begründung siehe Abschnitt 3.4

[7] „Günstig“ bedeutet hier, dass B in Bezug auf die Ordnung von $E(\mathbb{F}_p)$ weder zu groß noch zu klein ist.

[8] Dabei ist „Nenner“, der Nenner aus der Berechnung der Steigung während der Punktaddition gemäß 2.1.3.

3.2 Konstruktion der Elliptischen Kurven

In Schritt 1 des obigen Algorithmus sollen zufällige Elliptische Kurven gewählt werden. Wie man diese konkret erhält, beschreibe ich in diesem Abschnitt.
Das Problem an der Konstruktion ist, dass es für einen Computer ein äquivalentes Problem ist, eine Zahl zu faktorisieren und Quadratwurzeln $\mod n$ zu berechnen, da bei der Quadratwurzelberechnung ja gerade ein bestimmter Faktor gesucht wird. Das heißt also, dass es keine praktikable Möglichkeit ist, die Werte für A, B und u zu setzen und daraus v bestimmen zu wollen - wobei $P = (u, v)$ sein soll.
Glücklicherweise kann man die Elliptischen Kurven leicht definieren, indem man einfach eine Gleichung hinschreibt, und auf diese Weise die beschriebene Problematik umgeht. Diese Konstruktion beschreibe ich in diesem Abschnitt.[9]

Gefordert sind also zufällige Elliptische Kurven $\mod n$ und Punkte auf diesen Kurven.

Man beginne damit, Werte für

$$A \in \mathbb{Z} \quad , \quad A \mod n$$

und ein Paar

$$(u, v) \in \mathbb{Z}^2 \quad , \quad (u, v) \mod n$$

zu wählen.

Dann setze

$$C = v^2 - u^3 - Au \qquad \mod n.$$

Damit haben wir Elliptische Kurven

$$y^2 = x^3 + Ax + C$$

und Punkte darauf

$$P = (u, v).$$

[9] Wie so oft ist der „Trick“ inhaltlich mathematisch nicht besonders anspruchsvoll, sondern mutet eher trivial an, trotzdem sollte man die Konstruktion meiner Meinung nach einmal ausführlich beschreiben.

3.3 Beispiele

Im Folgenden werde ich zwei Zahlen mit der ECM faktorisieren, um zu demonstrieren, wie die Methode funktioniert und dass sie vorteilhafter ist, als andere Methoden.
Begonnen werden soll mit einem Beispiel, das noch von Hand zu lösen ist. Beim zweiten wird eine deutlich größere Zahl mit dem Computerprogramm ARIBAS faktorisiert werden.

3.3.1 n := 4453

Wir wollen $n := 4453$ faktorisieren:

Schritt 1 - Zur Auswahl der Elliptischen Kurven:
Wähle $P = (u, v) := (1, 3)$ und $A := 10$.

Dann setze

$$\begin{aligned} C &= v^2 - u^3 - Au \\ &= 3^2 - 1^3 - 10 \cdot 1 \\ &= -2 \end{aligned}$$

Daraus folgt, dass die Elliptische Kurve mit dem Punkt

$$P = (1, 3) \qquad \text{mod } 4453$$

folgende Form hat:

$$y^2 = x^3 + 10x - 2 \qquad \text{mod } 4453$$

Schritt 2 - Zur Berechnung der Punktvielfachen und Steigungen:
Es sei $B := 3$.

Berechne $2 \cdot P := (x_2, y_2)$:

Dann ist die Steigung m_2 gegeben durch:

$$m_2 = \frac{3u^2 + A}{2v} = \frac{3 \cdot 1^2 + 10}{2 \cdot 3} = \frac{13}{6}$$

Es ist

$$ggT(6, 4453) = 1 \quad \Rightarrow \quad 6^{-1} \equiv 3711 \qquad \text{mod } 4453,$$

also ist

$$m_2 = \frac{13}{6} \equiv 3713 \qquad \text{mod } 4453.^{10}$$

Die neuen Punktkoordinaten erhält man ebenfalls durch die in Abschnitt 2 vorgestellten Formeln:

$$\begin{aligned} x_2 &= m_2^2 - 2u \equiv 3713^2 - 2 \equiv 4332 \qquad \text{mod } 4453 \\ y_2 &= m_2(u - x_2) - v \equiv 3230 \qquad \text{mod } 4453 \end{aligned}$$

Berechne $3 \cdot P := (x_3, y_3)$:

Nun ist die Steigung m_3 gegeben durch:

$$m_3 = \frac{y_2 - v}{x_2 - u} = \frac{3230 - 3}{4332 - 1} = \frac{3227}{4331}$$

Allerdings ist

$$ggT(4331, 4453) = 61 \neq 1$$

[10] Nebenbemerkung: An dieser Stelle könnte man theoretisch schon aufhören, da nun bekannt ist, dass die Steigung m existiert. Allerdings werden die neuen Koordinaten von 2P ebenfalls berechnet, um sie für den nächsten Schritt benutzen zu können.

und deswegen ist $4331^{-1} \mod 4453$ nicht berechenbar und da man sofort sieht, dass 61 prim ist, folgt, dass unser von n gesuchter Primfaktor 61 ist.

Wir haben also gefunden:

$$4453 = 61 \cdot q$$

Und damit kann man sogar noch leicht bestimmen, dass $q = 73$ ist.

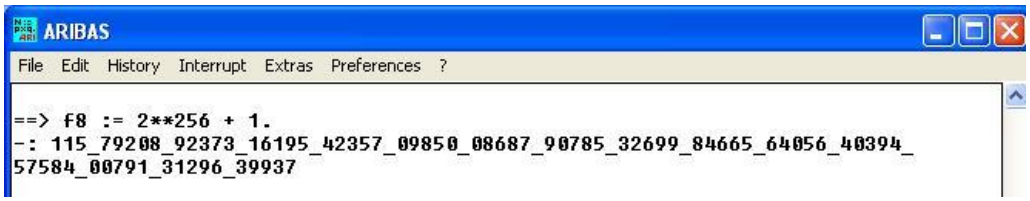
3.3.2 n := F_8

Als zweites wollen wir die 8. Fermat-Zahl $2^{2^8} + 1$ faktorisieren.

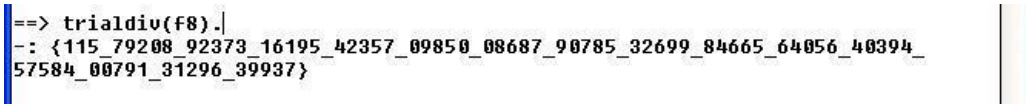
Da diese 78-stellige Zahl deutlich zu groß zum Faktorisieren per Hand ist, benutze ich das Programm ARIBAS von Otto Forster als Hilfe.[11]
Zur Veranschaulichung drucke ich im Folgenden die zugehörigen Screenshots mit den entsprechenden Kommentaren ab.

Zur Demonstration, dass die ECM tatsächlich einen deutlichen Vorteil gegenüber anderen Methoden hat, habe ich F_8 ebenfalls von zwei anderen Verfahren faktorisieren lassen.

Zuerst wird die 8. Fermat-Zahl $2^{256} + 1$ als Variable „f8“ definiert, sodass sie im Folgenden leicht aufgerufen werden kann:



Das erste Verfahren ist „trialdiv“. Hierbei findet eine einfache Divison von F_8 durch alle Primzahlen $p < 2^{16}$ statt. Die Ausgabe bedeutet, dass dabei kein Faktor gefunden wurde, da gerade wieder die eingegebene Zahl ausgegeben wird.

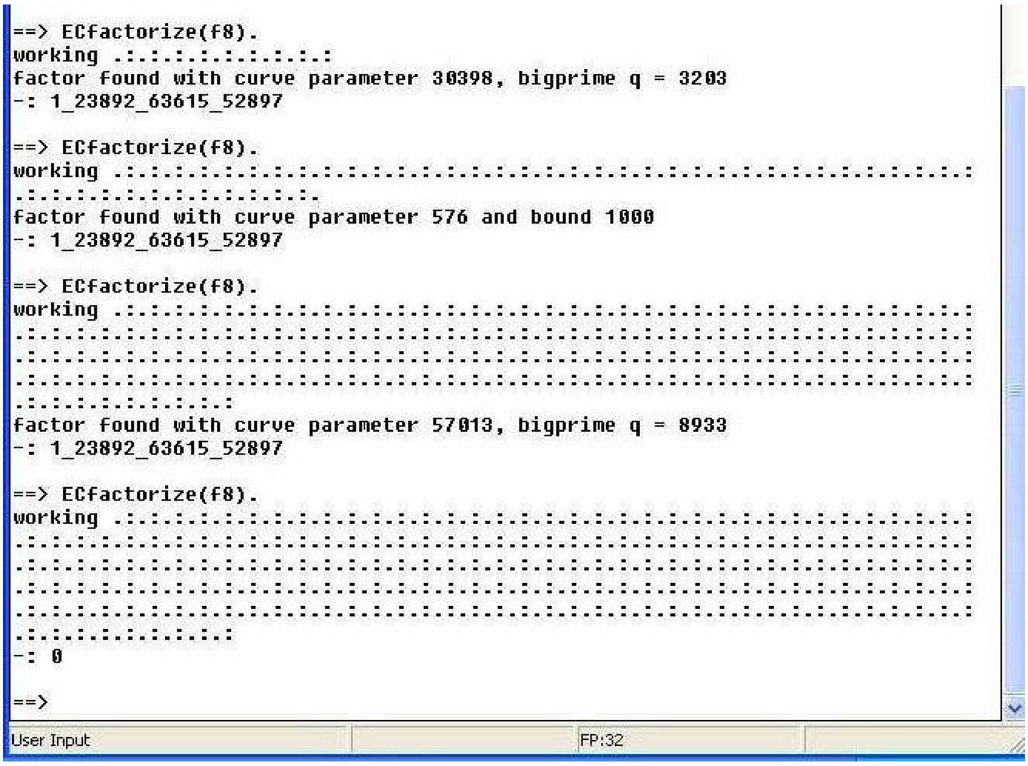


Das zweite Verfahren ist „p1_factorize“, also die (p-1)-Methode von Pollard. Auch hier lautet die Ausgabe „0“, das Verfahren kommt also zu keinem Ergebnis.
Möglich wäre es zwar mit diesem Verfahren einen Primfaktor zu finden, aber die Grenze in ARIBAS ist standardmäßig für diese Methode auf $q^k \leq 16000$ festgelegt - wobei q eine beliebige Primzahl und k einen Laufindex darstellt. Da $p - 1$ kein Produkt von Primpotenzen $q^k \leq 16000$ ist, wird innerhalb dieser Grenze also ebenfalls kein Primfaktor für F_8 gefunden.

Als letztes wurde die ECM mit ARIBAS mithilfe des Befehls „ECfactorize“ aufgerufen. Neben dem gefundenen Primfaktor gibt dieses Programm auch den Kurvenparameter und eine sogenannte „bigprime“ oder eine Grenze „bound“ aus.
Bei dem Kurvenparameter handelt es sich um das a aus der Darstellung $y^2 = x^3 + ax^2 + x$, das bedeutet also, dass Forster nicht die Weierstraß-Gleichung, sondern eine andere äquivalente Darstellung zugrunde legt.

[11] Die Programm-Codes befinden sich im Anhang.

Weiterhin bedeutet das Ende der Rechnung, bei der ein „bound“-Wert ausgegeben wird - gekennzeichnet durch einen einfachen Punkt - dass keine Big Prime Variation (gekennzeichnet durch einen Doppelpunkt) nötig war. Ausgegeben werden dann entsprechend die Schranke für Primfaktoren der verwendeten Elliptischen Kurven oder die analoge Schranke für die Big Prime Variation.[12]



Um zu zeigen, dass das Programm tatsächlich zufällige Kurven wählt, habe ich die Zahl insgesamt viermal faktorisieren lassen. Dabei wird schon an diesen Beispielen sehr deutlich, dass die Wahl der Kurven entscheidend für ein Gelingen der Methode und für die Rechenzeit des Programms ist.
Bei den ersten drei Wahlen kommt das Programm schließlich zum Faktor 1238926361552897, bei der letzten waren offenbar die Kurven ungeschickt gewählt oder die Grenze zu klein, sodass kein Faktor gefunden wurde.

Schließlich bleibt noch zu erwähnen, dass die „bound“-Schranke intern auf 1000 und die Anzahl der betrachteten Kurven auf 200 festgesetzt ist. Man könnte diese Werte zusätzlich von Hand variieren, worauf ich an dieser Stelle aber verzichte.

[12] Vgl. Abschnitt 2.3

3.4 Wann und warum funktioniert die EC-Methode?

Nachdem wir nun in den vorherigen Abschnitten die Faktorisierungsmethode mithilfe der Elliptischen Kurven kennengelernt und etwas damit experimentiert haben, möchte ich in diesem Abschnitt darauf eingehen, für welche Zahlen die Methode gut zu einem Ergebnis führt und im zweiten Teil soll darauf eingegangen werden, warum sie überhaupt funktioniert.

Allgemein hängt die Schranke für die Berechenbarkeit von Primfaktoren großer Zahlen bei dieser Methode weniger an den zu faktorisierenden Zahlen, als an der Größe des kleinsten Primfaktors, da sich die Rechenzeit des Algorithmus' an dem kleinsten Primfaktor der zu faktorisierenden Zahl orientiert.
Das bedeutet also, dass sich die angegebenen Grenzen nach der Rechenleistung von Computern richten. Sollten jene großer werden, müssten auch die Grenzen angepasst werden.

Konkret sollte für die zu faktorisierende Zahl n eine Zahl gewählt werden, die bis zu 100 Stellen haben darf. Gleichzeitig sollte der Primfaktor nicht mehr als 40 Stellen haben. Dies sind die Schranken, die Washington in seinem Buch[13] nennt, allerdings liegt seit Mai 2010 der Rekord für einen mit der ECM gefundenen Primfaktor bei einer 73-stelligen Zahl.
Es handelt sich dabei um einen Primfaktor der Mersenne-Zahl $2^{1181} - 1$ und einer der „Entdecker" ist einer der Brüder vom ECM-Entwickler Hendrik Lenstra, nämlich Arjen Lenstra.[14] Ein weiterer daran beteiligter bekannter Mathematiker ist Peter Montgomery.

Für die Anwendung in der praktisch durchgeführten Kryptographie ist damit leider die ECM nicht mehr zu gebrauchen, da hierfür Primzahlen mit mehr als 75 Stellen benutzt werden. Die Primfaktoren von $n = p \cdot q$, wobei $p, q > 10^{75}$ sind, zu finden, ist mit dieser Methode heutzutage noch nicht in akzeptabler Rechenzeit möglich.
Hierfür gibt es andere Methoden, beispielsweise das Zahlkörpersieb, innerhalb der die ECM ebenfalls benutzt wird.

Kommen wir nun zu der Antwort auf die zweite Frage:

Es sei dafür gegeben:

$$n = p \cdot q$$

Dann können wir eine Elliptische Kurve $E \mod n$ auch als $(E \mod p)$ und $(E \mod q)$ betrachten und nach dem Satz von Hasse gilt:

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

Es gilt sogar, dass jede ganze Zahl aus dem Intervall

$$(p + 1 - 2\sqrt{p} \quad , \quad p + 1 + 2\sqrt{p})$$

als Ordnung für eine Elliptische Kurve vorkommt.

Wenn unsere Schranke B nun sinnvoll gewählt wurde, hat das zur Folge, dass die Dichte der B-glatten ganzen Zahlen hoch genug und gleichzeitig die Verteilung der Ordnungen der Elliptischen Kurven E_i hinreichend gleich ist.
Als Konsequenz daraus folgt, dass bei der Wahl von verschiedenen zufälligen Elliptischen Kurven mindestens eine dabei sein wird, die eine B-glatte Ordnung hat.

Insbesondere bedeutet dies für einen Punkt P, der auf E liegt, dass es sehr wahrscheinlich ist, dass

$$(B!)P \equiv \infty \qquad \mod p$$

gilt und genauso unwahrscheinlich ist, dass gleichzeitig gilt:

$$(B!)P \equiv \infty \qquad \mod q$$

[13] vgl. [Wa], Kapitel 7
[14] vgl. [LM]

Sollte dies trotzdem einmal der Fall sein, war die Schranke zu hoch gewählt und man wiederhole die Untersuchung mit einem kleineren B.

Was bedeutet diese Erkenntnis für unseren Algorithmus?

Wenn wir

$$(B!)P \mod n$$

berechnen, erwarten wir eine Steigung, deren Nenner durch p, aber nicht durch q teilbar ist. Sobald dies der Fall ist, haben wir

$$ggT(\text{Nenner}, n) = p$$

gefunden und sind fertig.
Wäre allerdings der Nenner sowohl durch p, als auch durch q teilbar, hätten wir nichts gewonnen, da

$$ggT(\text{Nenner}, n) = n$$

gälte und dies offenbar nicht den gesuchten Faktor liefert.

4 Vergleich & Zusammenhang der verschiedenen Faktorisierungsmethoden

In diesem Abschnitt soll darauf eingegangen werden, was die ECM mit den klassischen Faktorisierungsmethoden verbindet und was sie voneinander unterscheidet.

4.1 Anwendung von ECM auf singuläre Kurven

Wenn wir die klassischen Faktorisierungsmethoden und die ECM in eine einzige Theorie einbinden könnten, hätte dies eine gewisse Schönheit und Geschlossenheit, die Mathematiker im Allgemeinen sehr anziehend und erstrebenswert finden.
Tatsächlich ist dies für unseren Fall möglich, denn wenn die ECM auf singuläre Kurven angewendet wird, führt dies zu drei anderen Methoden, die wir bereits kennen.

Wir unterscheiden im folgenden 2 allgemeine Gruppen singulärer Kurven und betrachten diese einzeln.

4.1.1 Singuläre Kurven mit dreifacher Nullstelle

Wir betrachten Elliptische Kurven der Form:

$$y^2 = x^3 \mod n$$

Dann ist ein Isomorphismus mit der additiven Gruppe $\mod n$ gegeben durch:

$$\begin{aligned} E_{ns} = E(\mathbb{Z}_n) \setminus (0,0) &\to \mathbb{Z}_n \\ (x,y) &\mapsto \frac{x}{y} \end{aligned}$$

Und daraus ergeben sich sofort folgende Entsprechungen:
(Man beachte, dass ∞ und 0 die jeweiligen Neutralelemente der verschiedenen Gruppen sind.)

	ECM		Division durch Primzahlen	
Wähle zufällig:	Punkt P auf E_{ns}		$a \in \mathbb{Z}_n$	
Berechne:	$(B!)P \mod n$		$(B!)a \mod n$	
	$(B!)P \equiv \infty \mod p$	$\Leftrightarrow$	$(B!)a \equiv 0 \mod p$	$(*)$
Berechne:	$p \mid ggT(\text{Nenner}, n)$		$p \mid ggT((B!)a, n)$	

Das heißt, dass die ECM angewendet auf singuläre Kurven mit einer dreifachen Nullstelle auf die intuitivste Faktorisierungsmethode überhaupt führt: Das Dividieren der Zahl n durch jede Primzahl, die kleiner oder gleich B ist. (Denn (*) tritt genau dann ein, wenn $p \leq B$ ist.)

4.1.2 Singuläre Kurven mit doppelten Nullstellen

1. Wir betrachten als erstes die Elliptische Kurve, die durch folgende Gleichung gegeben ist:

$$y^2 = x^2(x+1) \mod n$$

Nun kann man leicht, durch Nachrechnen der entsprechenden geforderten Eigenschaften, zeigen, dass ein Isomorphismus mit der multiplikativen Gruppe $\mod n$ gegeben ist durch:

$$\begin{aligned} E_{ns} = E(\mathbb{Z}_n) \setminus (0,0) &\to \mathbb{Z}_n^\times \\ (x,y) &\mapsto \frac{x+y}{x-y} \end{aligned}$$

Und daraus ergeben sich offensichtlich sofort folgende Entsprechungen:
(Man beachte dabei, dass ∞ und 1 die Neutralelemente der jeweiligen Gruppen sind.)

	ECM	(p-1)-Methode
Wähle zufällig:	Punkt P auf E_{ns}	$a \in \mathbb{Z}_n^\times$
Berechne:	$(B!)P \mod n$	$a^{B!} \mod n$
	$(B!)P \equiv \infty \mod p \quad \Leftrightarrow$	$a^{B!} \equiv 1 \mod p$
Berechne:	$p \mid ggT(\text{Nenner}, n)$	$p \mid ggT(a^{B!} - 1, n)$

Das bedeutet also, dass wir für Elliptische Kurven der Form $y^2 = x^2(x + 1)$ im Prinzip die (p-1)-Methode durchführen, wenn wir die ECM anwenden.

2. Anmerkung ohne Einzelheiten:
Analog dazu führt die Anwendung von ECM auf Elliptische Kurven der Form

$$y^2 = x^2(x + a) \qquad \mod n,$$

wobei a keine Quadratzahl $\mod p$ sein darf, auf die (p+1)-Methode.

4.2 Vergleich von ECM mit der (p-1)-Methode

Im Abschnitt über die Singulären Kurven mit doppelter Nullstelle wurde deutlich, wie die beiden Verfahren konkret zusammenhängen und was sie für Gemeinsamkeiten haben.

Bereits in der Einleitung wurde aber auch erwähnt, dass die Faktorisierungsmethode mit Elliptischen Kurven eine deutlich Verbesserung der (p-1)-Methode darstellt. Nun kann man diese Aussage schon allein aus dem Grunde für plausibel halten, weil für die ECM beliebige Elliptische Kurven benutzt werden können und die (p-1)-Methode offenbar auf spezielle eingeschränkt ist.

Man kann den Unterschied der beiden Methoden aber noch konkreter fassen:

Bei der (p-1)-Methode besteht eine begründete Chnace, dass die Zahl $(p - 1)$ B-glatt ist.
Ist sie das aber nicht, hat man praktisch keine Rettungsmöglichkeiten des Algorithmus und die Faktorisierung schlägt fehl.

Auch bei der Methode mit Elliptischen Kurven hat man eine gute Chance, dass $\#E(\mathbb{F}_p)$ B-glatt ist. Ist sie das aber in diesem Fall nicht, wählt man einfach neue zufällige Elliptische Kurven und beginnt von vorn.

Zusammengefasst heißt das also, dass die ECM eine deutlich größere Variationsmöglichkeit hat, als (p-1) dies bieten kann.

5 Zusammenfassung

In dieser Arbeit habe ich die Faktorisierungsmethode von Hendrik Lenstra vorgestellt, die sich die speziellen Eigenschaften von Elliptischen Kurven zu Nutze macht und somit deutlich größere Zahlen zuverlässig faktorisieren kann, als dies die klassischen Methoden vermochten.

Trotzdem stehen die verschiedenen Methoden keinesfalls diametral zueinander, sondern haben bestimmte Gemeinsamkeiten und können in eine große einheitliche Theorie eingebaut werden - wie ich in Abschnitt 4 zeigte.

Wie wir gesehen haben, hat aber auch die ECM Grenzen, die besonders durch die Rechenleistung unserer aktuellen Computer gesetzt werden.

Die Algorithmen, die ich verwendet habe, könnte man noch weiter optimieren. Beispielsweise, indem man nicht $(B!)P$ berechnet, sondern nur das Produkt aller Primzahlen $\pi_i \leq B$ bestimmt und dann berechnet:

$$\left(\prod_{i=2}^{B} \pi_i\right) P$$

Weiterhin kann man durch geschicktes Aufteilen von $\left(\prod_{i=2}^{B} \pi_i\right)$ in Zweierpotenzen oder mit Hilfe von Lucas-Ketten einige Rechenzeit für den Computer einsparen. Oder man betrachtet spezielle Darstellungen von Elliptischen Kurven, die dann Edwards-Kurven genannt werden, die im Vergleich zu der Weierstraß-Darstellung zu vereinfachten Additionsformeln führen.[15]

Letztlich bleibt festzuhalten, dass, wie ich bereits in der Einleitung schrieb, es seit der Antike bekannt ist, dass es eine eindeutige Primfaktorzerlegung für jede natürliche Zahl gibt. Das Problem lag und liegt nur darin, diese Primfaktoren für eine beliebige vorgegebene Zahl auch zu finden. Theoretisch hat man verschiedene Faktorisierungsmethoden, die allerdings in der Praxis durch Speicher- und Rechenkapazität von Computern beschränkt sind.

[15] Für genauere Informationen vgl. [BBLP]

6 Literaturverzeichnis

BBLP Daniel J. Bernstein, Peter Birkner, Tanja Lange, Christiane Peters: ECM using Edwards Curves, 2010.
verfügbar unter: http://eecm.cr.yp.to/eecm-20100616.pdf
Zugriff vom Januar 2011

Fo1 Otto Forster: Algorithmische Zahlentheorie; Vieweg Verlag, 1996.

Fo2 Otto Forster: ARIBAS
verfügbar unter: ftp://ftp.mathematik.uni-muenchen.de/pub/forster/aribas/
Zugriff vom Januar 2011

Le Hendrik W. Lenstra (Jr.): Factoring Integers with Elliptic Curves.
In: Annals of Mathematics, Second Series, Volume 126, Issue 3 (Nov., 1987), 649-673

LM Arjen Lenstra, Peter Montgomery u.a., 2010.
Zitiert nach: http://www.loria.fr/∼zimmerma/records/top50.html
Zugriff vom Januar 2011

Wa Lawrence C. Washington: Elliptic Curves. Number Theory and Cryptography.
2nd Edition; Chapman & Hall / CRC, 2008.

7 Anhang: Programm Codes von ARIBAS

In den folgenden Unterabschnitten zitiere ich die von Otto Forster zur Verfügung gestellten Programm Codes für ARIBAS, die ich für das zweite Beispiel aus Abschnitt 3 verwendet habe.
Dabei sind in 7.1 bis 7.3 die konkret verwendeten Programme und in 7.4 von ihnen verwendete Funktionen abgedruckt.

7.1 Division der zu faktorisierenden Zahl durch alle Primzahlen $< 2^{16}$

```
** Trial division by primes p < 2**16
** Constructs an array of factors of x.
** All elements with possible exeption of the last,
** are prime factors < 2**16. If the last element
** in the array is < 2**32, it is a prime.
** The product of all elements in the array equals x.
*)
function trialdiv(x: integer): array;
var
    st: stack;
    q: integer;
begin
    q := 2;
    while q := factor16(x,q) do
        stack_push(st,q);
        x := x div q;
    end;
    stack_push(st,x);
    return stack2array(st);
end;
```

7.2 (p-1)-Methode

```
** Pollard's (p-1)-factoring algorithm
** In general a prime factor p of x is found, if
** p-1 is a product of prime powers q**k <= bound
*)
function p1_factorize(x: integer; bound := 16000): integer;
const
    anz0 = 128;
var
    base, d, n, n0, n1, ex: integer;
begin
    base := 2 + random(64000);
    d := gcd(base,x);
    if d > 1 then
        return d;
    end;
    writeln(); write("working ");
    for n0 := 0 to bound-1 by anz0 do
        n1 := min(n0 + anz0, bound);
        ex := ppexpo(n0,n1);
        base := base ** ex mod x;
        write('.'); flush();
        if base <= 1 then
```

```
            return 0;
        else
            d := gcd(base-1,x);
            if d > 1 then
                writeln();
                writeln("factor found with bound ",n1-1)
                return d;
            end;
        end;
    end;
    return 0;
end;
(*-------------------------------------------------------------------*)
```

7.3 Lenstras ECM

```
** Elliptic curve factorization with big prime variation.
** N is the number to be factored.
** bound and bound2 are bounds for the prime factors
** of the order of the randomly chosen elliptic curve.
** anz is the maximal number of elliptic curves tried
** Returns a factor of N or 0 in the case of failure
*)
function ECfactorize(N: integer; bound := 1000;
                     bound2 := 10000; anz := 200): integer;
var
    k, a, d: integer;
begin
    write("working ");
    for k := 1 to anz do
        a := 3 + random(64000);
        d := gcd(a*a-4,N);
        if d = 1 then
            write('.'); flush();
            d := ECfact0(N,a,bound);
        end;
        if d <= 0 then
            write(':'); flush();
            d := ECbigprimevar(N,a,-d,bound2);
        end;
        if d > 1 and d < N then return d; end;
    end;
    return 0;
end;
```

7.4 Verwendete Hilfsprogramme

```
** Constructs a list of all prime factors of x.
** If a prime p is a multiple factor of x,
** it is listed repeatedly according to its multiplicity.
** In case of failure, the empty list () is returned
**
** Uses trial division, rho_factorize and qs_factorize.
**
** This function writes a progress report to the screen,
** which can be suppressed by setting the last argument verbose = 0.
*)
function factorlist(x: integer; verbose := 1): array;
var
    st, st1: stack;
    q, y, bound: integer;
    vec: array;
    count: integer;
begin
    x := abs(x);
    if x < 2 then
        return ();
    end;
    q := 2;
    while q := factor16(x,q) do
        stack_push(st,q);
        x := x div q;
        if verbose then
            writeln(q);
        end;
    end;
    if x < 2**32 then
        stack_push(st,x);
        if verbose then
            writeln(x);
        end;
    else
        stack_push(st1,x);
    end;
    while not stack_empty(st1) do
        x := stack_pop(st1);
        if rab_primetest(x) then
            stack_push(st,x);
            if verbose then
                writeln(x);
            end;
        else
            bound := bit_length(x);
            bound := 4*bound**2;
            if verbose then
                writeln("trying to factorize ",x," using Pollard rho")
            end;
            y := rho_factorize(x,bound,verbose);
            count := 0;
            while (y <= 1 or y >= x) do
                if inc(count) > 2 then
                    writeln("unable to factorize ",x);
                    return ();
```

```
                end;
                if verbose then
                    writeln("trying to factorize ",x,
                            " using quadratic sieve");
                end;
                y := qs_factorize(x,verbose);
            end;
            if verbose then
                writeln("found factor ",y);
            end;
            stack_push(st1,x div y);
            stack_push(st1,y);
        end;
    end;
    vec := stack2array(st);
    return sort(vec);
end;
 (*-------------------------------------------------------------*)
(*
** Produkt aller Primzahlen B0 < p <= B1
** und aller ganzen Zahlen isqrt(B0) < n <= isqrt(B1)
** Diese Funktion wird gebraucht von den Funktionen
** p1_factorize, pp1_factorize und ECfactorize
*)
function ppexpo(B0,B1: integer): integer;
var
    x, m0, m1, i: integer;
begin
    x := 1;
    m0 := max(2,isqrt(B0)+1); m1 := isqrt(B1);
    for i := m0 to m1 do
        x := x*i;
    end;
    if odd(B0) then inc(B0) end;
    for i := B0+1 to B1 by 2 do
        if prime32test(i) > 0 then x := x*i end;
    end;
    return x;
end;
```

```
(*-----------------------------------------------------------------*)
(*
** Called by function ECfactorize, not to be called directly
**
** Faktorisierungs-Algorithmus mit der elliptischen Kurve
**      y*y = x*x*x + a*x*x + x
** bound ist Schranke fuer die Primfaktoren der Elementezahl
**      der elliptischen Kurve
*)
(*-----------------------------------------------------------------*)
function ECfact0(N,a,bound: integer): integer;
const
    anz0 = 128;
var
    x, B0, B1, s, d: integer;
    xx: array[2];
begin
    x := random(N);
    for B0 := 0 to bound-1 by anz0 do
        B1 := min(B0+anz0,bound);
        s := ppexpo(B0,B1);
        xx := mod_pemult(x,s,a,N);
        if xx[1] = 0 then
            d := xx[0];
            if d > 1 and d < N then
                writeln(); write("factor found with curve ");
                writeln("parameter ",a," and bound ",B1);
            end;
            return d;
        else
            x := xx[0];
        end;
    end;
    return -x;
end;
(*-----------------------------------------------------------------*)
(*
** auxiliary function, called by ECfactorize
*)
function ECbigprimevar(N,a,x,bound: integer): integer;
const
    Maxhdiff = (22, 36, 57);
    Maxbound = (15000, 31000, 1000000);
var
    XX: array of array[2];
    maxhdiff: integer;
    c, i, q, k, d: integer;
    P,Q,R: array[2];
begin
    k := length(Maxhdiff) - 1;
    while k > 0 and bound <= Maxhdiff[k-1] do
    dec(k);
    end;
    bound := min(bound,Maxbound[k]);
    maxhdiff := Maxhdiff[k];
    XX := alloc(array,maxhdiff+1,(0,0));

    c := ((x + a)*x + 1)*x mod N;
    P := (x,1);
```

```
    Q := ecN_dup(N,a,c,P);
    if Q[1] < 0 then return Q[0]; end;
    XX[1] := R := Q;
    for i := 2 to maxhdiff do
    R := ecN_add(N,a,c,R,Q);
        if R[1] < 0 then return R[0]; end;
    XX[i] := R;
    end;
    R := ecN_add(N,a,c,P,Q); (* R = 3*P *)
    if R[1] < 0 then return R[0]; end;
    d := 0;
    q := 3;
    while q < bound do
        k := 1; inc(q,2);
        while prime32test(q) /= 1 do
        inc(q,2); inc(k);
    end;
        R := ecN_add(N,a,c,R,XX[k]);
    if R[1] < 0 then
        d := R[0];
        if d > 1 and d < N then
            writeln();
                writeln("factor found with curve parameter ",a,
                    ", bigprime q = ",q);
            end;
            break;
    end;
    end;
    return d;
end;
(*-----------------------------------------------------------------*)
(*
** auxiliary function, called by ECfactbpv
**
** Addition zweier Punkte P,Q auf der elliptischen Kurve
**      c*y**2 = x**3 + a*x**2 + x (modulo N)
** Falls waehrend der Rechnung durch eine nicht zu N teilerfremde
** Zahl geteilt werden muss, wird ein Paar (d,-1) zurueckgegeben,
** wobei d ein Teiler von N ist.
** Sonst Rueckgabe der Summe P+Q = (x,y) mit 0 <= x,y < N.
*)
function ecN_add(N,a,c: integer; P,Q: array[2]): array[2];
var
    x1,x2,x,y1,y2,y,m: integer;
begin
    if P = Q then
        return ecN_dup(N,a,c,P);
    end;
    x1 := P[0]; x2 := Q[0];
    m := mod_inverse(x2-x1,N);
    if m = 0 then
        return (gcd(x2-x1,N),-1);
    end;
    y1 := P[1]; y2 := Q[1];
    m := (y2 - y1)*m mod N;
    x := (c*m*m - a - x1 - x2) mod N;
    y := (- y1 - m*(x - x1)) mod N;
    return (x,y);
end;
```

```
(*-------------------------------------------------------------*)
(*
** Verdopplung eines Punktes P auf der elliptischen Kurve
**      c*y**2 = x**3 + a*x**2 + x (modulo N)
** Falls waehrend der Rechnung durch eine nicht zu N teilerfremde
** Zahl geteilt werden muss, wird ein Paar (d,-1) zurueckgegeben,
** wobei d ein Teiler von N ist.
** Sonst Rueckgabe von P+P = (x,y) mit 0 <= x,y < N.
*)
function ecN_dup(N,a,c: integer; P: array[2]): array[2];
var
    x1,x,y1,y,z,m,Pprim: integer;
begin
    x1 := P[0]; y1 := P[1];
    z := 2*c*y1;
    m := mod_inverse(z,N);
    if m = 0 then
        return (gcd(z,N),-1);
    end;
    Pprim := (((3*x1 + 2*a)*x1) + 1) mod N;
    m := Pprim*m mod N;
    x := (c*m*m - a - 2*x1) mod N;
    y := (- y1 - m*(x - x1)) mod N;
    return (x,y);
end;
(*-------------------------------------------------------------------*)
(*
** Multiplication of a point P on the elliptic curve
**      c*y**2 = x**3 + a*x**2 + x (modulo N)
** by an integer s >= 1.
** If during the calculation a division by a number which is
** not coprime to N must be performed, the function returns
** immediately a pair (d,-1), where d is a divisor of N.
*)
function ecN_mult(N,a,c: integer; P: array[2]; s: integer): array[2];
var
    k: integer;
    Q: array[2];
begin
    if s = 0 then return (0,-1); end;
    Q := P;
    for k := bit_length(s)-2 to 0 by -1 do
        Q := ecN_dup(N,a,c,Q);
        if Q[1] < 0 then
            return Q;
        end;
        if bit_test(s,k) then
            Q := ecN_add(N,a,c,Q,P);
            if Q[1] < 0 then
                return Q;
            end;
        end;
    end;
    return Q;
end;
(*********************************************************************)
```