Diplomarbeit

Untersuchung von Elliptischen Kurven für die Tauglichkeit zur Hardwareakzeleration von Kryptoverfahren

Frank Bohnsack

Universität Hamburg
Fachbereich Informatik
Juni 1997

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Historie von Kryptosystemen

Schon immer gab es das Bestreben des Menschen, Nachrichten vor unbefugten Personen zu schützen. Die Kryptologie beschäftigt sich mit dieser Problematik. Die Geschichte der Verschlüsselungen reicht weit zurück. Bereits vor ungefähr 2500 Jahren benutzte die Regierung von Sparta Zylinder, sogenannte Skytale, um geheime Nachrichten an die Generäle zu senden. Um den Zylinder wurde ein schmales Band aus Pergament gewickelt, auf das dann die Nachricht in Längsrichtung geschrieben wurde. War das Band abgewickelt, so konnte nur eine Person die Nachricht lesen, die einen Zylinder gleichen Umfangs hatte. Diese Verfahren ist der Prototyp eines Transpositionsalgorithmus: die Buchstaben bleiben, was sie sind, nur ihre Position ändert sich. Es handelt sich hier also um eine Permutation der Stellen der Buchstaben. Der römische Feldherr und Staatsmann C. Julius Caesar (100 - 44 v. Chr.) benutzte eine andere Variante der Verschlüsselung: jeder Buchstabe wurde durch einen anderen ersetzt, die Position bleibt dabei unverändert. Das geheime Alphabet ergab sich daraus, daß unter dem Klartextalphabet noch einmal dasselbe um eine bestimmte Anzahl von Stellen verschoben schreibt:

Klartext	:	a	b	c	d	...	w	x	y	z
Geheimtext	:	d	e	f	g	...	z	a	b	c

Aber schon bei längeren Nachrichten wird eine derartige Verschlüsselung recht mühsam und unsicher. So erfand bereits der Italiener Leon Battista Alberti (1404 - 1472) eine Maschine, die das Verschlüsseln mechanisierte. Sie bestand aus zwei Scheiben, von denen die innere gegenüber der äußeren verdrehbar war. So konnte problemlos jede Verschiebung eingestellt werden. Auch die im zweiten Weltkrieg verwendete ENIGMA basiert auf einem ähnlichen Prinzip, welches allerdings wesentlich komplexer ist als das hier beschriebene von Alberti. Alle diese Verfahren basierten auf einer symmetrischen Verschlüsselung. Der Empfänger mußte genau entgegengesetzt die Schritte des Senders durchführen bei Verwendung desselben Schlüssels. 1976 führten Diffie und Hellmann ein ganz neues Verschlüsselungsverfahren ein: ein asymmetrisches Verfahren. Wie der Name schon vermuten läßt, invertiert der Empfänger hier nicht die Schritte des Senders zum Entschlüsseln der geheimen Nachricht, sondern verwendet einen anderen Schlüssel als der Sender. Somit werden also unterschiedliche Aufgaben beim Ver- und Entschlüsseln durchgeführt, mit anderen Worten: wer entschlüsseln kann, kann noch lange nicht verschlüsseln und umgekehrt.

1.2 Anwendungsgebiete

Die treibende Kraft für die Entwicklung komplexer Verschlüsselungssysteme war und ist das Militär. Jegliche militärische Operationen, sollten sie von Erfolg gekrönt werden, mußten vor den Feinden geheimgehalten werden. Auch heute ist die Geheimhaltung von militärischen Operationen, wie zuletzt erfolgreich in Kuwait praktiziert, von großer Bedeutung.
Neben der militärischen Bedeutung zeichnen sich aber auch zunehmend zivile Anwendungsgebiete ab. Heutzutage finden sowohl die symmetrischen als auch die asymmetrischen Kryptosysteme in vielen Bereichen ihre Anwendung. Entscheidend für die Wahl eines bestimmten Verfahrens sind die Sicherheitsanforderungen. Bei low-security Anwendungen können kleinere Schlüssellängen verwendet werden als bei high-security Anwendungen, wie zum Beispiel Schlüsselaustausch, Telebanking, usw.

1.3 Entwicklungen der Computertechnik

Waren vor einigen Jahren noch Einzelplatzrechner in den Unternehmen die Regel, so ist nun nicht nur eine lokale Vernetzung zu verzeichnen, sondern in zunehmenden Maße wird auch eine globale Vernetzung, siehe zum Beispiel das Internet, angestrebt. Diese Entwicklung birgt natürlich die Gefahr des Datenmißbrauchs, so daß die Frage nach einer gesicherten und geschützten Übertragung der Daten immer mehr in den Vordergrund tritt. Die heutigen Kryptoverfahren sind der wachsenden Datenflut und den schnelleren Prozessoren nicht gewachsen: Zum einen steigt die Anforderung des Benutzers, der eine möglichst transparente Verschlüsselung haben möchte (möglichst on-line), zum anderen erfordern immer schnellere Prozessoren einen höheren Sicherheitsstandard. Das bedeutet eine immer größere Schlüssellänge. So hat aber zum Beispiel eine Verdopplung der Schlüssellänge beim RSA-Verfahren im Prinzip einen vierfachen Hardware Aufwand zur Folge, was dementsprechend trotz schnellerer Prozessoren unweigerlich einen Geschwindigkeitsverlust zur Folge hat. So ist also abzusehen, daß die Diskrepanz zwischen Anforderung des Benutzers und den Sicherheitsanforderungen zunimmt. Folglich müssen entweder die bisherigen Kryptoalgorithmen stark verbessert oder neue Verfahren basierend auf anderen mathematischen Modellen entwickelt werden. Diese Arbeit wird sich mit letzterem befassen und eine Untersuchung über die Tauglichkeit von Elliptischen Kurven, die im folgenden Kapitel näher erläutert werden, zur Hardwareakzeleration von Kryptoverfahren durchführen.

Kapitel 2

Mathematische Grundlagen

Das im Rahmen dieser Arbeit implementierte Verfahren basiert auf einer Elliptischen Kurve, die ihrerseits wieder über einen Endlichen Körper definiert ist. Im folgenden werden deshalb zunächst die mathematischen Strukturen definiert und an einigen kleinen Beispielen veranschaulicht.

2.1 Körper

Dieser Abschnitt gibt einen Überblick über die grundlegenden Definitionen und Eigenschaften eines Körpers.

Definition Körper:
K sei eine nichtleere Menge, auf der zwei Operationen $+$ und $\cdot$ (Addition und Multiplikation) erklärt sind. K heißt ein kommutativer **Körper**, falls folgende Gesetze erfüllt sind:

1. Assoziativgesetze:
   $$\begin{aligned} a + (b + c) &= (a + b) + c \\ a \cdot (b \cdot c) &= (a \cdot b) \cdot c \end{aligned}$$

2. Kommutativgesetze:
   $$\begin{aligned} a + b &= b + a \\ a \cdot b &= b \cdot a \end{aligned}$$

3. Distributivgesetze:
   $$a \cdot (b + c) = a \cdot b + a \cdot c$$

4. Existenz neutraler Elemente:
   Es gibt in K zwei verschiedene Elemente $\mathbf{0}$ und $\mathbf{1}$, welche die Eigenschaft haben, daß für jedes $a \in K$ gilt:
   $$a + \mathbf{0} = a \quad \text{und} \quad a \cdot \mathbf{1} = a$$

5. Existenz inverser Elemente:
   Zu jedem $a \in K$ gibt es ein Element $-a \in K$ mit folgender Eigenschaft:

$a + (-a) \quad = \quad 0$
Ferner gibt es zu jedem $a \in K, a \neq 0$ ein Element $a^{-1} \in K$ mit folgender Eigenschaft:
$a \cdot a^{-1} \quad = \quad 1$

Ein Beispiel für einen Körper sind die reellen Zahlen ($\mathbb{R}$) oder die komplexen Zahlen ($\mathbb{C}$).

Definition Charakteristik eines Körpers:
Es sei K ein Körper. Läßt sich die Zahl $0 = 1 + 1 + \cdots + 1$ durch Additionen des neutralen Elements 1 bezüglich der Multiplikation darstellen, dann wird die Anzahl der notwendigen neutralen Elemente als *Charakteristik* des Körpers K bezeichnet. Gibt es eine derartige Zahl nicht, dann hat der Körper die Charakteristik 0.

Unendliche Körper haben den Nachteil, daß sie nicht exakt auf den Rechner abgebildet werden können. Ihre Repräsentation ist zudem noch maschinenabhängig. So kann es zu Überläufen und Rundungsfehlern kommen, die bei Kryptoverfahren unerwünscht sind. Diese Ungenauigkeiten führen letztendlich zu der Einführung von Endlichen Körpern.

2.2 Endlicher Körper

Es gelten für die Endlichen Körper dieselben Gesetze wie für die im vorigen Abschnitt angesprochenen Körper, allerdings ist die Anzahl der Elemente beschränkt und damit endlich. Prinzipiell gibt es zwei Typen von Endlichen Körpern, nämlich der Form $GF(p)$ und der Form $GF(p^m)$, wobei p eine Primzahl ist und $m \in \mathbb{N}$. Im folgenden sollen nur die für die Kryptosysteme wichtigen Eigenschaften aufgeführt werden, insbesondere sollen hier nur Endliche Körper der Form $GF(p)$ betrachtet werden.

2.2.1 Endliche Körper der Form $GF(p)$

Sei p eine Primzahl. Dann sind die Elemente des Endlichen Körpers $GF(p)$ die Menge $\{0, 1, 2, \ldots, p-1\}$ mit den folgenden arithmetischen Operationen in $\mathbb{Z}$:

- *Addition:* Wenn $a, b \in GF(p)$, dann ist $a + b = r$, wobei r der Rest der Division von $(a + b)$ durch p ist. Das entspricht folgender Rechenoperation:

$$(a + b) \bmod p = r, \quad r \in GF(p)$$

- *Multiplikation:* Wenn $a, b \in GF(p)$, dann ist $ab = s$, wobei s der Rest der Division von ab durch p ist. Das entspricht folgender Rechenoperation in $\mathbb{Z}$:

$$ab \bmod p = s, \quad s \in GF(p)$$

Definition Generator:
Es sei $GF(p)^*$ die Menge aller Elemente aus $GF(p)$, die $\neq 0$ sind. Dann gibt es in $GF(p)$ mindestens ein Element g, durch dessen Potenzen alle Elemente von $GF(p)^*$ dargestellt

werden können. Ein solches Element g bezeichnet man als *Generator* oder *primitives Element*.

Das bedeutet, daß folgendes gilt:

$$GF(p)^* = \{g^i : 0 \leq i \leq p-2\}$$

Generatoren spielen bei der Berechnung von Inversen und der Wurzel eine große Rolle. Sie können aber auch als Basis für einen Endlichen Körper aufgefaßt werden, mit deren Hilfe alle Elemente des Körpers erzeugt werden können.

Inverse in $GF(p)$
Es sei $GF(p)^*$ die Menge aller Elemente aus $GF(p)$, die $\neq 0$ sind. Dann bildet $GF(p)^*$ eine multiplikative Gruppe, weil alle Elemente aus $GF(p)^*$ teilerfremd zu p sind. Man kann $GF(p)^*$ auch folgendermaßen beschreiben:

$$GF(p)^* = \{a \in GF(p) | ggT(a,p) = 1\}$$

Da p eine Primzahl ist, wird damit auch ersichtlich, daß es zu jedem Element $a \in GF(p)^*$ eine **multiplikative Inverse** gibt. Dieses kann mit Hilfe der Vielfachsummendarstellung berechnet werden. Es gilt:
Ist $ggT(a,p) = 1 = (sa + tp) \bmod p$ mit $s, t \in \mathbb{Z}$, so ist $a^{-1} = s \bmod p$ die multiplikative Inverse zu a mit $s \neq 0$.
Daß diese Aussage korrekt ist, läßt sich leicht durch folgende Umformung zeigen, wobei immer $a^{-1} \cdot a = 1$ gilt:

$$\begin{array}{rcll} ((s \bmod p) \cdot a) \bmod p &=& (s \cdot a) \bmod p & \\ &=& (s \cdot \frac{1-tp}{s}) \bmod p & \text{mit } 1 = sa + tp \Leftrightarrow a = \frac{1-tp}{s} \\ &=& (1 - tp) \bmod p & \\ &=& 1 \bmod p - tp \bmod p & \\ &=& 1 & \end{array}$$

Um also das Inverse zu einem Element $a \in GF(p)^*$ zu bestimmen, berechnet man mit dem Erweiterten Euklidischen Algorithmus die Koeffizienten s und t. Die genaue Beschreibung des Algorithmus erfolgt in 5.1.2.

Beispiel $GF(17)$
Anhand von $GF(17) = \{0, 1, \ldots, 16\}$, einem kleinen Endlichen Körper, der sich als überschaubares Beispiel durch diese Arbeit zieht, sollen hier einmal die Rechenregeln für die wichtigsten Operationen aufgezeigt werden.

- **Addition**
  $$\begin{array}{rcrcl} (12+13) \bmod 17 &=& 25 \bmod 17 &=& 8 \\ (5+9) \bmod 17 &=& 14 \bmod 17 &=& 14 \end{array}$$

- **Subtraktion**
  $$\begin{array}{rcrcl} (12-3) \bmod 17 &=& 9 \bmod 17 &=& 9 \\ (5-9) \bmod 17 &=& -4 \bmod 17 &=& 13 \end{array}$$

- **Multiplikation**

$$\begin{array}{lllll} (12 \cdot 3) \bmod 17 & = & 36 \bmod 17 & = & 2 \\ (5 \cdot 9) \bmod 17 & = & 45 \bmod 17 & = & 11 \end{array}$$

- **Inverses**

Die Existenz eines Inversen eines Elementes ist nach der Definition eines Körpers gegeben. Die Berechnung desselben ist allerdings nicht so einfach, denn es muß ja in unserem Fall folgende Gleichung gelöst werden:

$$x \cdot \text{invers}(x) = 1 \bmod 17 \quad \text{(1 ist das neutrale Element der Multiplikation)}$$

Zum Berechnen des Inversen verwendet man den *erweiterten Euklidischen Algorithmus*, welcher detailiert in [COH93] beschrieben ist.
Die folgende Tabelle listet alle inversen Elemente aus $GF(17)$ auf:

x	0	1	2	3	4	5	6	7	8
invers(x)	-	1	9	6	13	7	3	5	15

x	9	10	11	12	13	14	15	16
invers(x)	2	12	14	10	4	11	8	16

- **Division**

Die Division läßt sich leicht auf die Berechnung des Inversen zurückführen, denn es gilt:

$$(a : b) \bmod p = (a \cdot b^{-1}) \bmod p$$

- **Wurzel**

Die Bestimmung der Wurzel ist in einem Endlichen Körper sehr schwierig, weil die Funktion im Gegensatz zu $\mathbb{R}$ nur partiell definiert ist. Mit dem Legendre Symbol (siehe 5.1.3) kann man leicht feststellen, ob es sich bei dem Argument um einen quadratischen Rest handelt. Nur in diesem Fall ist die Funktion $\sqrt{a}$ mit $a \in GF(p)$ überhaupt definiert. Wie dieses im Einzelnen geschieht, wird im Kapitel 5.1.4 detailliert beschrieben. Ist dieses der Fall, dann gibt es eine Lösung zur Bestimmung der Wurzel. Kennt man eine Lösung, dann läßt sich die zweite Lösung wie folgt bestimmen:

$$2. \text{ Lösung} = p - 1. \text{ Lösung}$$

Der Beweis folgt ebenfalls im Kapitel 5.1.4. Die folgende Tabelle listet alle Lösungen der Wurzelberechnung von $GF(17)$ auf:

x	0	1	2	3	4	5	6	7	8
1.$\sqrt{x}$	0	1	6	-	2	-	-	-	12
2.$\sqrt{x}$	0	16	11	-	15	-	-	-	5

x	9	10	11	12	13	14	15	16
1.$\sqrt{x}$	3	-	-	-	8	-	7	4
2.$\sqrt{x}$	14	-	-	-	9	-	10	13

- **Generator**
  Ein Generator ist natürlich bei unserem Beispiel einfach zu finden, weil wir nur maximal $17-2$ Elemente überprüfen müssen. Als Beispiel soll hier die 5 als Generator dienen. Dann lauten die Potenzen von 5 in $GF(p)$:

$$\begin{array}{cccccc} 5^0=1 & 5^1=5 & 5^2=8 & 5^3=6 & 5^4=13 & 5^5=14 \\ 5^6=2 & 5^7=10 & 5^8=16 & 5^9=12 & 5^{10}=9 & 5^{11}=11 \\ 5^{12}=4 & 5^{13}=3 & 5^{14}=15 & 5^{15}=7 & 5^{16}=1 & \end{array}$$

2.3 Elliptische Kurven

Historisch gesehen kommt der Begriff der Elliptischen Kurven aus der Theorie der elliptischen Integrale, wie sie zum Beispiel in der Bogenmaßberechnung bei Ellipsen (daher der Name) oder in der Physik, um die Bewegung eines Pendels zu berechnen ([COH93]), vorkommen.
Im folgenden sollen die notwendige Grundlagen über Elliptische Kurven dargestellt werden, die zum Verständnis für Kryptoverfahren basierend auf Elliptischen Kurven notwendig sind.

2.3.1 Grundlegende Eigenschaften

Elliptische Kurven spielen bei der Faktorisierung großer natürlicher Zahlen und beim Primzahlentest eine wichtige Rolle. Da sie eine kommutative Gruppe bilden, in der, auch wenn sie sehr groß sind, sich noch gut rechnen läßt, finden sie auch Anwendung in Kryptosystemen. Die hierzu notwendigen Rechenregeln sollen in diesem Abschnitt aufgezählt werden. Beweise werden hier nicht explizit durchgeführt. Für interessierte Leser sei hier auf die Bücher von [COH93], [KOB87] und [SIL86] verwiesen.

Definition Elliptische Kurve:
Sei K ein Körper (z.B. $\mathbb{R}$ oder $\mathbb{C}$) der Charakteristik $\neq 2, \neq 3$ und sei x^3+ax+b $(a, b \in K)$ eine kubisches Polynom mit keiner vielfachen Wurzel. Dann bezeichnet man eine Elliptische Kurve über K als Menge der Punkte (x, y) mit $x, y \in K$, die folgende Gleichung erfüllen

$$y^2 = x^3 + ax + b, \tag{2.1}$$

mit einem zusätzlichen Punkt O, dem sogenannten *Unendlichkeitspunkt.*

Wenn der Körper K die Charakteristik $= 2$ hat, dann ist eine Elliptische Kurve die Menge der Punkte, die die Gleichung (2.1) erfüllen, wobei y^2 durch $y^2 + y$ zu ersetzen ist:

$$y^2 + y = x^3 + ax + b \tag{2.2}$$

Hierbei ist es aber unerheblich, ob das kubische Polynom vielfache Wurzel hat oder nicht. Zu dieser Menge kommt ebenfalls noch der Punkt O.

Wenn der Körper die Charakteristik $= 3$ hat, dann ist eine Elliptische Kurve die Menge der Punkte, die der Gleichung

$$y^2 = x^3 + ax^2 + bx + c \tag{2.3}$$

genügen, inklusive des Unendlichkeitspunktes O.

Dabei sind $a, b \in K$ so zu wählen, daß $4a^3 + 27b^2 \neq 0$ in K gilt. Ist letzteres nicht der Fall, handelt es sich um eine degenerierte Elliptische Kurve, d.h. es handelt sich nicht mehr um eine Elliptische Kurve, weil sie nun einen singulären Punkt hat. Ein singulärer Punkt ist ein Punkt, der unterschiedliche Ableitungen hat, z.B. zwei unterschiedliche Steigungen (siehe Abbildung 2.1, 3a). Damit gelten die Gruppengesetze (zum Beispiel die Addition) nur noch auf dem restlichen Teil der Elliptischen Kurve ohne singulärem Punkt.

Definition Addition auf Elliptischer Kurve E:
Für eine Elliptische Kurve E werden folgende Additionsregeln definiert:

1. Für die Addition zweier Unendlichkeitspunkte O gilt:

$$O + O = O$$

2. Für die Addition eines beliebigen Punktes $(x, y) \in E$ mit dem Unendlichkeitspunkt O gilt:

$$(x, y) + O = (x, y) \quad \text{für alle } (x, y) \in E$$

3. Für zwei beliebige Punkte (x_1, y_1) und (x_2, y_2) werden folgende Fallunterscheidungen getroffen:

   (a) Falls $x_1 = x_2$ und $y_1 = -y_2$ sind, gilt:

   $$(x_1, y_1) + (x_2, y_2) = (x_1, y_1) + (x_1, -y_1) = O,$$

   also ist der Punkt (x_1, y_1) der inverse Punkt zu $(x_1, -y_1)$.

   (b) Falls $x_1 = x_2$ und $y_1 = y_2$ sind, sprich der Punkt zu sich selbst addiert wird (Verdoppelung des Punktes), gilt:

   $$(x_1, y_1) + (x_2, y_2) = 2(x_1, y_1) = (x_3, y_3)$$

   wobei sich die Koordinate x_3 und y_3 nach folgender Formel berechnen lassen:

$$x_3 = \lambda^2 - 2x_1, \quad y_3 = \lambda(x_1 - x_3) - y_1, \quad \text{und } \lambda = \tfrac{3x_1^2+a}{2y_1}$$

(c) Sonst gilt:

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

Die Koordinaten errechnen sich dabei wie folgt:

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1, \quad \text{und } \lambda = \tfrac{y_2-y_1}{x_2-x_1}$$

Es scheint so, daß der Fall $x_1 = x_2$ und $y_1 \neq y_2$ nicht berücksichtigt wird. Dies ist aber nicht der Fall, denn wenn die y Koordinaten unterschiedlich sind, dann können die Punkte nur invers zueinander sein. Durch Einsetzen von x_1 oder x_2 (es macht ja keinen Unterschied, weil beide gleich sind) in $y^2 = x^3 + ax + b$ ergibt sich für y^2 ein bestimmter Wert w. Daraus ergibt sich als Lösung für $y = \pm\sqrt{w}$. Da aber $y_1 \neq y_2$ sein soll, folgt sofort, daß $y_1 = -y_2$ sein muß. Somit können die beiden Punkte nur invers zueinander sein.

Nach diesen Additionsregeln bildet die Menge der Punkte der Elliptischen Kurve $E(K)$ eine kommutative Gruppe. Es gilt also dementsprechend auch das Kommutativgesetz:

$$(x_1, y_1) + (x_2, y_2) = (x_2, y_2) + (x_1, y_1) \quad \text{für alle } (x_1, y_1) \text{ und } (x_2, y_2) \in E.$$

Der Unendlichkeitspunkt O ist bezüglich der Addition das neutrale Element.

2.3.2 Elliptische Kurven über $\mathbb{R}$

Betrachtet werden nun Elliptische Kurven über $\mathbb{R}$. Um sich eine Elliptische Kurve zu veranschaulichen, muß man sich ihre Gleichung ansehen:

$$y^2 = x^3 + ax + b \tag{2.4}$$

Aus dieser Gleichung sieht man sofort, daß diese Kurve symmetrisch zur x-Achse ist. Betrachtet wird nun die linke Seite von Gleichung (2.4), also das Polynom $Q(x) = x^3 + ax + b$. Aufgrund von x^3 erhält man eine, zwei (eine doppelte Nullstelle) oder drei Nullstellen. Sobald man eine doppelte Nullstelle erhält, handelt es sich nicht mehr um eine Elliptische Kurve, sondern um eine degenerierte, weil sie nun einen singulären Punkt enthält. Betrachtet werden nun die unterschiedlichen Fälle, die auftreten können. Für eine Fallunterscheidung diene die Bedingung $\Delta = 4a^3 + 27b^2 \neq 0 \in \mathbb{R}$. Dann können folgende Fälle unterscheiden werden (alle Beispielkurven zeigt Abbildung 2.1 auf Seite 14):

1. Fall $\Delta < 0$
   dann hat die Gleichung $Q(x) = 0$ nur eine reelle Nullstelle und die Graph der Kurve besteht nur noch aus einer zusammenhängenden Komponente. Wir können hier aber noch eine zusätzliche Unterscheidung treffen:

- Fall 1a:
  Die Kurve hat horizontale Tangenten, d.h. sie hat reelle Extrema. Das ist genau dann der Fall, wenn $a > 0$ ist. Beispielkurve: $y^2 = x^3 + x + 2$

- Fall 1b:
  Die Kurve hat keine horizontale Tangenten, d.h. sie hat keine reellen Extrema. Das ist genau dann der Fall, wenn $a \leq 0$ ist. Beispielkurve: $y^2 = x^3 - 3x + 3$

2. Fall $\Delta > 0$
   dann hat die Gleichung $Q(x) = 0$ drei verschiedene reelle Nullstellen. Die Kurve besteht aus zwei Teilen, einem Oval und einem nicht kompakten Teil, der den Unendlichkeitspunkt O enthält. Beispielkurve: $y^2 = x^3 - 2x + 1$

3. Fall $\Delta = 0$
   Bei dieser Kurve handelt es sich nicht mehr um eine Elliptische Kurve, weil sie einen singulären Punkt hat. Die Kurve hat nun mindestens eine doppelte Nullstelle, so daß wir das Polynom $Q(x)$ auch folgendermaßen schreiben können:

$$Q(x) = (x-c)^2(x-d) \qquad mit\ \ 2c + d = 0 \tag{2.5}$$

Die Bedingung $2c + d = 0$ ergibt sich aus der Tatsache, daß in der Gleichung (2.4) einer Elliptischen Kurve keine Potenz zweiter Ordnung vorkommt. Durch Ausmultiplizieren der Gleichung (2.5) erhält man:

$$\begin{aligned} Q(x) &= (x^2 - 2cx + c^2)(x-d) \\ &= x^3 - (2c+d)x^2 + (c^2 + 2cd)x - c^2 d \end{aligned}$$

Folglich muß der Koeffizient $2c+d = 0$ sein. Nun kann man diesen dritten Fall wieder in 3 weitere Fälle unterteilen:

- Fall 3a: $c > d$
  dann besteht die Kurve aus einem zusammenhängenden Graphen mit einer doppelten reellen Nullstelle bei $x = c$ mit zwei unterschiedlichen 1. Ableitungen. Beispielkurve: $y^2 = x^3 - 3x + 2$

- Fall 3b: $c > d$
  dann besteht die Kurve aus zwei getrennten Teilen, einem nicht kompakten Teil, der den Unendlichkeitspunkt O enthält, und einem einzelnen Punkt mit den Koordinaten $(c, 0)$, mit allerdings rein imaginären unterschiedlichen ersten Ableitungen. Beispielkurve: $y^2 = x^3 - 3x - 2$

- Fall 3c: $c = d$
  dann müssen nach Bedingung von (2.5) $c = d = 0$ sein. Die Kurve hat dann bei $x = 0$ dieselbe 1. Ableitung. Beispielkurve: $y^2 = x^3$

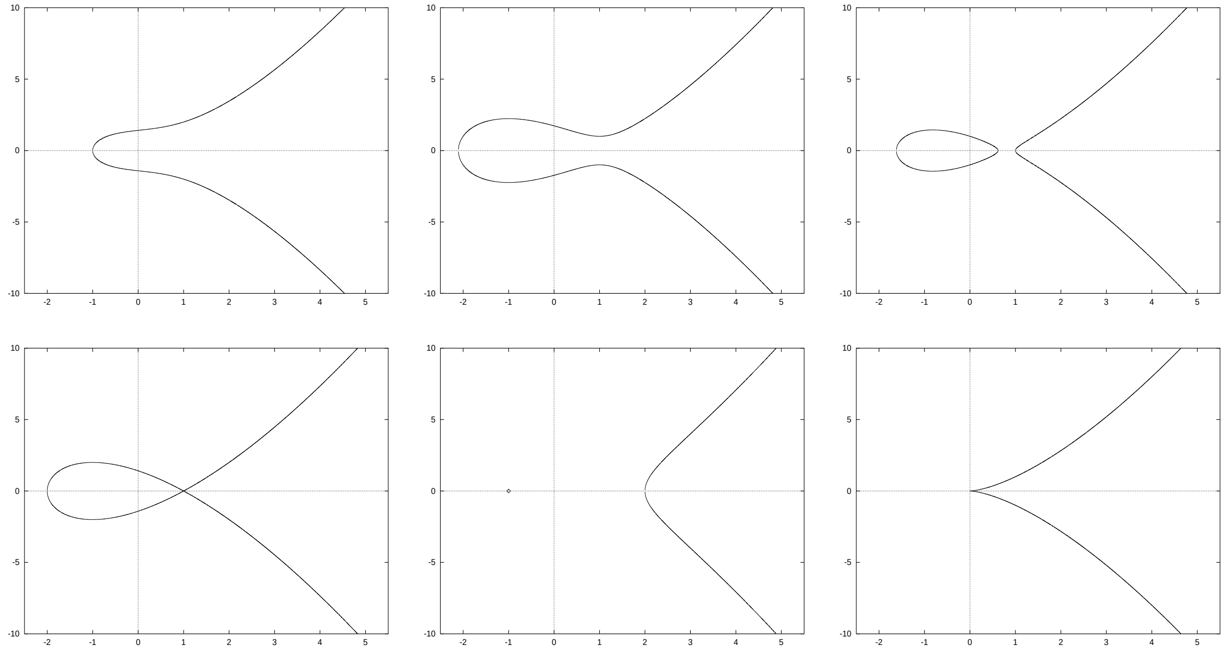

Abbildung 2.1: **Elliptische Kurven:** *Die obere Reihe zeigt von links nach rechts die Fälle 1a, 1b und 2, die untere Reihe ebenfalls von links nach rechts die Fälle 3a, 3b und 3c (degenerierte Kurven)*

Im folgenden soll die geometrische Deutung der Addition von zwei Punkten auf einer Elliptischen Kurve E gezeigt werden und dabei Formeln für die Berechnung des resultierenden Punktes hergeleitet werden. Im folgenden seien $P = (x_1, y_1)$, $Q = (x_2, y_2)$ die zu addierenden Punkte und $R = (x_3, y_3)$ der resultierende Punkt. Um den dritten Punkt zu erhalten, wird eine Gerade durch die Punkte P und Q gelegt, die E in einem dritten Punkt S schneidet. Der resultierende Punkt R ist dann der inverse Punkt zu S. Die Gerade $l = \overline{PQ}$ läßt sich auch durch die Geradengleichung $y = \lambda x + \beta$ beschreiben. Dann ergibt sich für die Steigung λ und den y-Achsenabschnitt β:

$$\lambda = \tfrac{y_2 - y_1}{x_2 - x_1}, \quad \beta = y_1 - \lambda x_1$$

Ein Punkt, der sowohl auf der Geraden l als auch auf E liegt, muß also folgende Bedingung erfüllen:

$$(\lambda x + \beta)^2 = x^3 + ax + b \quad \Leftrightarrow \quad 0 = x^3 - (\lambda x + \beta)^2 + ax + b \tag{2.6}$$

Es werden die drei Lösungen der Gleichung (2.6) gesucht, wobei schon zwei von ihnen bekannt sind, nämlich die Punkte P und Q, die auf E liegen. Um die dritte Lösung zu finden, bedient man sich einfach der Tatsache, daß die Summe der Lösungen gleich dem negativen Koeffizienten der zweithöchsten Potenz ist, wenn der Koeffizient der höchsten Potenz gleich 1 ist, was in unserem Fall gegeben ist. Demnach gilt also folgendes:

$$\begin{aligned}\lambda^2 &= x_1 + x_2 + x_3 \\ x_3 &= \lambda^2 - x_1 - x_2\end{aligned}$$

Durch Einsetzen von λ kommt man dann zur obigen Formel:

$$x_3 = \left(\tfrac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$$

Durch folgende Umformungen bekommt man für y_3:

$$\begin{aligned}y_3 &= -(\lambda x_3 + \beta) && \text{mit } \beta = y_1 - \lambda x_1 \\ &= -(\lambda x_3 + y_1 - \lambda x_1) \\ &= \lambda(x_1 - x_3) - y_1 && \text{mit } \lambda = \tfrac{y_2 - y_1}{x_2 - x_1} \\ &= \left(\tfrac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1\end{aligned}$$

Für die Verdoppelung des Punktes P wird eine Tangente zu diesem Punkt gezogen. Auch hier erhält man einen dritten Punkt, dessen Spiegelbild bezüglich der x-Achse der resultierende Punkt $2P$ ist. Gegenüber der vorherigen Gleichungen für x_3 und y_3 haben sich die Steigung λ und die x_2 Koordinate, die nun $= x_1$ ist, geändert. Die Steigung der Tangente entspricht der ersten Ableitung von E im Punkt P. Dadurch ergibt sich für λ:

$$\lambda = \tfrac{x_1^3 + ax_1 + b}{y_1^2} \tfrac{dx}{dy} = \tfrac{3x_1^2 + a}{2y_1}$$

Somit ergibt sich für den resultierenden Punkt $R = (x_3, y_3)$:

$$\begin{aligned}x_3 &= \lambda^2 - x_1 - x_2 && \text{wegen Tangente gilt } x_1 = x_2 \\ x_3 &= (\tfrac{3x_1^2 + a}{2y_1})^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ y_3 &= \tfrac{3x_1^2 + a}{2y_1}(x_1 - x_3) - y_1\end{aligned}$$

Die Abbildung 2.2 veranschaulicht die Konstruktion von $P + Q$ und die Verdoppelung des Punktes H zu $2H$.

2.3.3 Elliptische Kurven über $GF(p)$

Die Addition bezüglich Elliptischer Kurven über $GF(p)$ unterscheidet sich von der über $\mathbb{R}$ nur dadurch, daß alle Operationen Modulo p gerechnet werden müssen. Interessant ist nur die Addition zweier beliebiger Punkte. Somit ergeben sich für die Addition zweier beliebiger Punkte (x_1, y_1) und (x_2, y_2) mit $y_1 \neq -y_2$ folgende Gleichungen:

1. normale Addition:

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

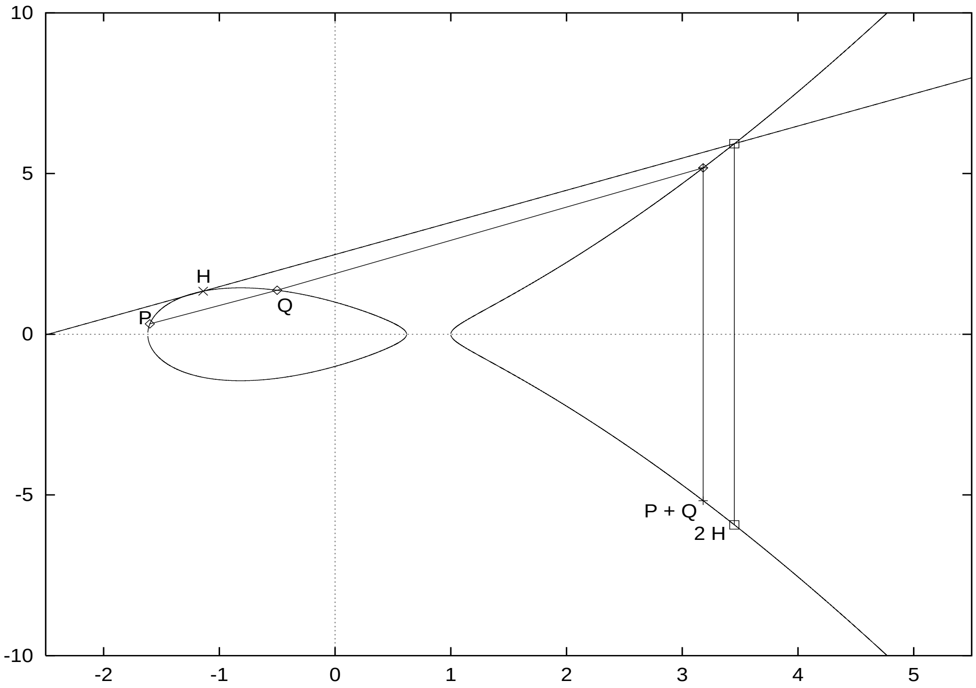

Abbildung 2.2: **Addition auf Elliptischen Kurven:** *Addition zweier unterschiedlicher Punkte P und Q zu $P + Q$ und Verdoppelung des Punktes H zu $2H$*

Die Koordinaten errechnen sich dabei wie folgt:

$$\begin{aligned} \lambda &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \bmod p, \\ x_3 &= (\lambda^2 - x_1 - x_2) \bmod p, \\ y_3 &= (\lambda(x_1 - x_3) - y_1) \bmod p, \end{aligned} \tag{2.7}$$

2. Verdoppelung des Punktes:

$$(x_1, y_1) + (x_2, y_2) = 2(x_1, y_1) = (x_3, y_3)$$

wobei sich die Koordinate x_3 und y_3 nach folgender Formel berechnen lassen:

$$\begin{aligned} \lambda &= \left(\frac{3x_1^2 + a}{2y_1}\right) \bmod p, \\ x_3 &= (\lambda^2 - 2x_1) \bmod p, \\ y_3 &= (\lambda(x_1 - x_3) - y_1) \bmod p \end{aligned} \tag{2.8}$$

Eine wichtige Eigenschaft für Elliptische Kurven in Kryptosystemen ist die Ordnung $\#E$, die wie folgt definiert ist:

Definition Ordnung $\#E$:
Gegeben sei eine Elliptische Kurve E. Dann versteht man unter der *Ordnung* $\#E$ die Anzahl der Punkte, die auf dieser Elliptischen Kurve liegen, inklusive des Unendlichkeitspunktes O.

Die Sicherheit eines Kryptosystems basierend auf Elliptischen Kurven hängt sehr stark von der Ordnung $\#E$ ab. Hat die Ordnung $\#E$ ausschließlich kleine Primfaktoren, so reduziert sich das Problem des Diskreten Logarithmus (siehe Kapitel 2.4) auf kleinere Zahlen. Optimal wären demnach also Elliptische Kurven E, deren Ordnung $\#E$ prim und möglichst groß ist. In welchen Bereichen sich die Ordnung $\#E$ zu einem gegebenen $GF(p)$ bewegt, wird aus dem **Hasses Theorem** deutlich:

Hasse Theorem:
Sei N die Anzahl der Punkte $(x, y), x, y \in GF(p)$ einer Elliptischen Kurve E definiert über $GF(p)$. Dann gilt folgender Zusammenhang zwischen N und p:

$$|N - (p + 1)| \leq 2\sqrt{p} \tag{2.9}$$

Da nach Definition die Anzahl der Punkte der Elliptischen Kurve nichts anderes ist als deren Ordnung, ist folgendes äquivalent:

$$|\#E - (p + 1)| \leq 2\sqrt{p} \tag{2.10}$$

Ein Beweis dieses Theorems befindet sich in [SIL86]. Folgende Betrachtung soll die Gleichung (2.9) verdeutlichen. Maximal können auf einer Elliptischen Kurve $2p + 1$ Punkte, nämlich der Unendlichkeitspunkt O und $2p$ Punkte (x, y) mit $x, y \in GF(p)$, liegen. $2p$ Punkte deshalb, weil es zu jedem $x \in GF(p)$ nach Gleichung 2.3 maximal zwei Lösungen gibt. In $GF(p)$ kann man aber nicht aus jedem Element die Wurzel ziehen, genauer gesagt geht das nur bei der Hälfte der Elemente aus $GF(p) - 0$. Es ist naheliegend, daß nur noch die Hälfte der $2p$ Punkte übrigbleiben. Das ist natürlich nicht korrekt, weil für unterschiedliche Elemente aus $GF(p)$ in der Gleichung $y^2 = x^3 + ax + b$ sehr wohl gleiche Werte für y^2 herauskommen können, die zufällig ein quadratischer Rest sind. Nimmt man das Legendre-Symbol (siehe 5.1.3) zu Hilfe, dann ergibt sich folgende Beziehung für die Anzahl A_l der Lösungen für $y \in GF(p)$ der Gleichung $y^2 = u$:

$$A_l = 1 + (\tfrac{u}{p})$$

Auf unsere Elliptische Kurve angewendet gilt dann:

$$\begin{aligned} \#E &= 1 + \sum_{x \in GF(p)} \left(1 + \left(\tfrac{x^3+ax+b}{p}\right)\right) \\ &= 1 + p + \sum_{x \in GF(p)} \left(\tfrac{x^3+ax+b}{p}\right) \end{aligned} \tag{2.11}$$

Man kann zeigen, daß die Summe $\sum_{x \in GF(p)} (\frac{x^3+ax+b}{p})$ begrenzt ist durch $2\sqrt{p}$, was dann dem Hasse Theorem entspricht.

Anhand der Ordnung $\#E$ kann man die Elliptischen Kurven noch in zwei Kategorien unterteilen, nämlich in die **supersingulären** und die **nicht-supersingulären**. Eine Elliptische Kurve E ist **supersingulär**, wenn ihre Ordnung $\#E = p + 1$ ist, anderenfalls ist sie **nicht-supersingulär**. Für Kryptoverfahren sind nur nicht-supersinguläre Elliptische Kurven einsetzbar, weil der Diskrete Logarithmus supersingulärer Elliptischer Kurven, wie in [MOV91] erläutert wird, auf die Bestimmung eines Diskreten Logarithmus über $GF(p)$ mit kleinem Grad reduziert werden kann.

Definition Ordnung Punkt $\#P$:
Die Ordnung $\#P$ eines Punktes P der Elliptischen Kurve E ist die kleinste natürliche Zahl n, für die folgendes gilt:

$$n \cdot P = \underbrace{P + P + \ldots + P}_{n} = O \tag{2.12}$$

Für die Berechnung der Ordnung $\#P$ ist die Ordnung der Elliptischen Kurve E notwendig. Wenn letztere eine Primzahl ist, dann haben alle Punkte der Elliptischen Kurve E dieselbe Ordnung. Es gilt also $\#P = \#E(GF(p))$. Dann fällt auch die Wahl eines Basispunktes B, der nach Kapitel 4 zu den geheimen Schlüsseln die öffentlichen Schlüssel generiert, nicht schwer. Es kommen alle in Frage, weil sie ja alle die gleiche Ordnung haben. Wenn $\#E \neq$ prim ist, dann wird $\#P$ nach folgendem Algorithmus berechnet:

1. Bestimme Gruppenordnung $\#E(GF(p))$
2. Faktorisiere die Gruppenordnung und bestimme so alle Teiler von $\#E(GF(p))$
3. Teste für alle Teiler d von $\#E(GF(p))$ in aufsteigender Reihenfolge, ob $d \cdot P = O$ ist. Das kleinste d, das diese Bedingung erfüllt, ist die Ordnung des Punktes Q.

Für große Zahlen kann die Berechnung sehr lange dauern, weil die Ordnung der Elliptischen Kurve zunächst faktorisiert werden muß.

2.3.4 Beispiel Elliptische Kurve über $GF(17)$

Anhand einer Elliptischen Kurve $E : y^2 = x^3 + 1x + 2$, die auch später Verwendung in einer Beispielverschlüsselung findet, sollen hier die obigen mathematischen Operationen vorgeführt werden.

- **Ordnung $\#E$**
  Die Punkte der Elliptischen Kurve $E : y^2 = x^3 + 1x + 2$ lauten:

							$\sum$
$(0,11)$	$(0,6)$	$(1,2)$	$(1,15)$	$(2,-)$	$(3,7)$	$(3,10)$	6
$(4,6)$	$(4,11)$	$(5,8)$	$(5,9)$	$(6,-)$	$(7,-)$	$(8,-)$	4
$(9,3)$	$(9,14)$	$(10,3)$	$(10,14)$	$(11,1)$	$(11,16)$	$(12,5)$	7
$(12,12)$	$(13,6)$	$(13,11)$	$(14,-)$	$(15,3)$	$(15,14)$	$(16,0)$	6
							23

Zu diesen 23 Punkten kommt dann noch der Unendlichkeitspunkt O. Damit ergibt sich für die Ordnung von E:

$$\#E = 23 + 1 = 24.$$

- **Addition zweier Punkte P und Q**
Gegeben seien der Punkt $P = (5, 8) = (x_1, y_1)$ und $Q = (10, 3) = (x_2, y_2)$. Nach (2.7) erhalten wir für λ, x_3 und y_3:

$$\begin{aligned}
\lambda &= \left(\frac{3-8}{10-5}\right) \bmod 17 = \left(\frac{-5}{5}\right) \bmod 17 = \left(\frac{12}{5}\right) \bmod 17 = (12 \cdot 7) \bmod 17 \\
&= 84 \bmod 17 = 16 \\
x_3 &= (16^2 - 5 - 10) \bmod 17 = (256 - 15) \bmod 17 = 241 \bmod 17 = 3 \\
y_3 &= (16(5-3) - 8) \bmod 17 = (16 \cdot 2 - 8) \bmod 17 = 24 \bmod 17 = 7
\end{aligned}$$

Somit erhalten wir als Ergebnis $P + Q = (5, 8) + (10, 3) = (3, 7) \in E$.

- **Verdoppelung eines Punktes P**
Gegeben sei der Punkt $P = (5, 8) = (x_1, y_1)$. Gesucht wird der Punkt $R = 2P = 2(5, 8) = (x_3, y_3)$. Nach (2.8) ergeben sich für λ, x_3 und y_3:

$$\begin{aligned}
\lambda &= \left(\frac{3 \cdot 5^2 + 1}{2 \cdot 8}\right) \bmod 17 = \left(\frac{76}{16}\right) \bmod 17 = \left(\frac{8}{16}\right) \bmod 17 = \left(\frac{1}{2}\right) \bmod 17 \\
&= (1 \cdot 9) \bmod 17 = 9 \\
x_3 &= (9^2 - 2 \cdot 5) \bmod 17 = (81 - 10) \bmod 17 = 71 \bmod 17 = 3 \\
y_3 &= (9(5-3) - 8) \bmod 17 = (9 \cdot 2 - 8) \bmod 17 = 10
\end{aligned}$$

Somit erhalten wir als Ergebnis $R = 2P = 2(5, 8) = (3, 10) \in E$. Der Punkt R ist zudem der inverse Punkt zu dem Punkt $P + Q = (3, 7)$ aus der vorherigen Addition.

- **Ordnung $\#P$**
um die Ordnung eines Punktes P zu bestimmen, benötigt man die Primzahlzerlegung von $\#E$. Diese lautet in unserem Beispiel:

$$\#E = 24 = 2^3 \cdot 3$$

Jeder Punkt P bildet eine Untergruppe von E, deren Ordnung Teiler von der Gruppenordnung $\#E$ oder die Gruppenordnung selbst ist. Damit kann $\#P$ die Werte 2, 3, 4, 6, 8, 12 oder 24 haben. Exemplarisch soll hier die Ordnung des Punktes $P = (15, 3)$ bestimmt werden. Berechnen wir zunächst $2P$ mit Hilfe von (2.8). Dann erhalten wir:

$$\begin{aligned}
\lambda &= \left(\frac{3 \cdot 15^2 + 1}{2 \cdot 3}\right) \bmod 17 = \left(\frac{13}{6}\right) \bmod 17 = (13 \cdot 3) \bmod 17 = 39 \bmod 17 = 5 \\
x_3 &= (5^2 - 2 \cdot 15) \bmod 17 = (-5) \bmod 17 = 12 \\
y_3 &= (5(15-12) - 3) \bmod 17 = 12
\end{aligned}$$

Wenden wir erneut (2.8) auf das Ergebnis $2P = (12, 12)$ an, dann erhalten wir $4P$:

$$\begin{aligned}
\lambda &= \left(\frac{3 \cdot 12^2 + 1}{2 \cdot 12}\right) \bmod 17 = \left(\frac{25}{24}\right) \bmod 17 = \left(\frac{8}{24}\right) \bmod 17 = \left(\frac{1}{3}\right) \bmod 17 = 6 \\
x_3 &= (6^2 - 2 \cdot 12) \bmod 17 = (36 - 24) \bmod 17 = 12 \\
y_3 &= (6(12 - 12) - 12) \bmod 17 = (-12) \bmod 17 = 5
\end{aligned}$$

Wir erhalten dann $4P = (12, 5)$. Dieses ist aber der inverse Punkt zum Punkt $2P$, also ist $4P = -2P$. Daraus folgt $6P = O$. Daher kann der Punkt P nur die Ordnung $2, 3$ oder 6 haben. Aber da $2P = (12, 12) \neq O$ nicht der Unendlichkeitspunkt O und $4P \neq P$, was der Fall sein müßte, wenn P die Ordnung 3 hätte, ist, ist $\#P = 6$.

2.4 Diskreter Logarithmus

Wichtige Grundbausteine der Kryptographie sind Funktionen, die leicht berechenbar sind, aber möglichst schwer umkehrbar sind, beispielsweise $a \cdot b$ oder a^2 gegenüber $a : b$ bzw. $\sqrt{a}$. Eine wichtige Klasse von schwer umkehrbaren Funktionen sind die diskreten Exponentialfunktionen.

Definition diskrete Exponentialfunktion:
Sei p eine Primzahl und sei n eine natürliche Zahl mit $n \leq p - 1$. Dann ist die diskrete Exponentialfunktion zur Basis n definiert durch:

$$k \to n^k \bmod p \quad \text{mit } 1 \leq k \leq p - 1.$$

Definition diskrete Logarithmusfunktion:
Sei p eine Primzahl und n eine natürliche Zahl mit $n \leq p - 1$. Dann ist die diskrete Logarithmusfunktion dl_n die Umkehrfunktion der diskreten Exponentialfunktion. Es gilt:

$$dl_n(n^k) = k$$

Definition diskreter Logarithmus:
Das Problem des diskreten Logarithmus bezeichnet dann die Aufgabe, zu einem gegebenen p, n und y ein k zu bestimmen, so daß folgende Gleichung gilt:

$$y = n^k \bmod p.$$

Die diskrete Exponentialfunktion zu berechnen, ist einfach, z.B. nach dem Square-and-multiply Verfahren (siehe Kapitel 5.1.1). Die Berechnung des diskreten Logarithmus ist dagegen schwer, weil die diskrete Exponentialfunktion nicht stetig ist. Ein Verfahren für die Berechnung des diskreten Logarithmus ist der Baby-Step-Giant-Step Algorithmus. Zunächst wird eine natürliche Zahl $w \geq \sqrt{N}$, wobei N die Anzahl der Elemente der

Gruppe ist, gewählt. Jeder diskrete Logarithmus k läßt sich dann in der Form $k = aw + b$ mit $0 \leq b < w$ darstellen. Somit können wir die Gleichung

$$y = n^k = n^{aw+b} \tag{2.13}$$

umformen in die Gleichung

$$yn^{-b} = n^{aw} \,. \tag{2.14}$$

Um also eine Lösung für k der Gleichung (2.13) zu finden, müssen wir Zahlen a und b bestimmen, die der Gleichung (2.14) genügen. Hierzu werden zwei Listen angelegt, die Baby-Step-Liste, die alle Elemente der Form yn^{-1} mit $b \in \{0, 1, \ldots, w\}$ enthält, und die Giant-Step-Liste, die alle Elemente der Form n^{aw} mit $a \in \{0, 1, \ldots, w\}$ enthält. Beide Listen werden geordnet. In beiden Listen wird nun ein gemeinsames Element gesucht. Ist dieses gefunden, erhalten wir sofort a und b und können damit den diskreten Logarithmus $k = aw + b$ berechnen. Der Baby-Step-Giant-Step Algorithmus hat eine Komplexität von $\mathrm{O}(\sqrt{p})$.

Definition diskreter Logarithmus über E:
Gegeben sei eine Elliptische Kurve E und ein Punkt B auf dieser Kurve. Als Problem des diskreten Logarithmus bezeichnet man die Aufgabe, zu einem gegebenen Punkt $P \in E$ eine Zahl $x \in \mathbb{Z}$ so zu finden, daß $xB = P$ gilt, wenn eine Zahl x überhaupt existiert.

Man vermutet, daß dieses Problem noch schwerer zu lösen ist als der diskrete Logarithmus bei Endlichen Körpern der Form $GF(p)$. Die am besten geeignete Technik, die für Endliche Körper der Form $GF(p)$ entwickelt worden ist, scheint bei Elliptischen Kurven nicht zu funktionieren.

2.5 Eulersche φ-Funktion

Eulersche φ-Funktion
Sei n eine natürliche Zahl. Dann wird die Funktion $\varphi(n)$ wie folgt definiert:

$$\varphi(n) = \sum_{t=1, ggt(t,m)=1}^{n} 1 = |\{k | k \in [1:n], ggt(n,k) = 1\}|,$$

Die Eulersche Funktion $\varphi(n)$ gibt also die Anzahl der zu n teilerfremden natürlichen Zahlen, die kleiner gleich n sind, an. Wenn die Faktorisierung der Zahl n bekannt ist, dann läßt sich $\varphi(n)$ mittels einer Formel berechnen. Dabei gilt zum Beispiel nach [BEU95]:

$$\begin{array}{lcll} \varphi(p) & = & p-1 & \text{für jede Primzahl } p \\ \varphi(pq) & = & (p-1)(q-1) & \text{für je zwei verschiedene Primzahlen } p \text{ und } q \end{array}$$

Interessant ist dabei der Aspekt, daß die Berechnung von $\varphi(n)$ mit $n = p \cdot q$ (p und q sind zwei unterschiedliche Primzahlen) genau so schwer ist wie die Faktorisierung von n. Kennt man die Faktoren p und q, dann läßt sich sofort $\varphi(n) = \varphi(pq) = (p-1)(q-1)$ berechnen. Kennt man umgekehrt $\varphi(n)$, dann kann man die Faktoren wie folgt berechnen:

$$\begin{aligned}
\varphi(n) &= (p-1)(q-1) \\
&= pq - p - q + 1 \qquad \text{mit } pq = n \\
&= n - p - q + 1 \\
p + q &= n - \varphi(n) + 1 \\
p &= n - \varphi(n) + 1 - q
\end{aligned}$$

Durch Einsetzen für p in der Gleichung $n = pq$ ergibt sich:

$$\begin{aligned}
n &= (n - \varphi(n) + 1 - q)q \\
0 &= (n - \varphi(n) + 1 - q)q - n \\
0 &= q^2 - (n - \varphi(n) + 1)q + n
\end{aligned}$$

Diese quadratische Gleichung läßt sich leicht lösen.
Für die Anwendung in der Kryptographie, insbesondere dem RSA-Verfahren (siehe 3.2.1), ist der Satz von Euler-Fermat wichtig:

Satz von Euler-Fermat:
Sei $Z_n = \{0, 1, \ldots, n-1\}$ und $Z_n^* = \{a \in Z_n | ggT(a, n) = 1\}$. Dann gilt für alle Zahlen $a \in Z_n^*$:

$$a^{\varphi(n)} \bmod n \equiv 1.$$

Dieser Satz läßt sich mit Überlegungen aus der Gruppentheorie begründen [WAE71]. Diese besagen, daß, wenn man ein Gruppenelement (in unserem Fall a) mit der Ordnung der Gruppe (in unserem Fall $\varphi(n)$) potenziert, sich immer das neutrale Element (in unserem Fall 1) ergibt.
Wendet man den Satz von Euler-Fermat auf $GF(p)$ an, so ergibt sich aus für alle $a \in GF(p)$, weil $\varphi(p) = p - 1$ ist:

$$a^{\varphi(n)} = a^{p-1} \equiv 1 \bmod n.$$

Daraus ergibt sich ferner, daß man, wenn $a \in GF(p)$ mit p potenziert wird, wieder $a \bmod p$ erhält.

Kapitel 3

Allgemein eingesetzte Kryptoverfahren

Prinzipiell können die verwendeten Kryptoverfahren in zwei Kategorien eingeteilt werden, nämlich in die **symmetrischen** und die **asymmetrischen** Verfahren, die auch als **Public-Key-**Verfahren bezeichnet werden. Diese beiden unterschiedlichen Klassen werden in den nächsten Abschnitten kurz erläutert und einige ihrer Vertreter vorgestellt.

3.1 Symmetrische Verfahren

Bei symmetrischen Verfahren wird die Verschlüsselung so realisiert, daß sowohl der Sender als auch der Empfänger den gleichen geheimen Schlüssel benutzen, um sich gegen die Außenwelt zu schützen. Dadurch läßt sich ein symmetrisches Verfahren auch folgendermaßen kennzeichnen:

1. Verschlüsselungsalgorithmus
   besteht aus einer Funktion f mit zwei Eingabewerten, dem Schlüssel k und dem Klartext m, und einer Ausgabe, dem Geheimtext c, der sich aus k und m ergibt:

   $$c := f(k, m)$$

2. Verschlüsselungsfunktion f
   muß umkehrbar sein, d. h. es muß eine Funktion f^* geben, welche die Wirkung von der Verschlüsselungsfunktion f rückgängig macht. Somit rekonstruiert f^* mit dem gleichen Schlüssel k aus dem Geheimtext c wieder den Klartext m:

   $$f^*(k, c) := m$$

Der Vorgang einer symmetrischen Verschlüsselung soll folgendes Schema verdeutlichen: Grundsätzlich kann man davon ausgehen, daß sowohl Ver- als auch Entschlüsselungsfunktion bekannt sind (**Kerckhoffsches Prinzip**). Also muß die Verschlüsselungsfunktion f sicher sein. Dieses ist sie, wenn sie folgende Angriffe übersteht:

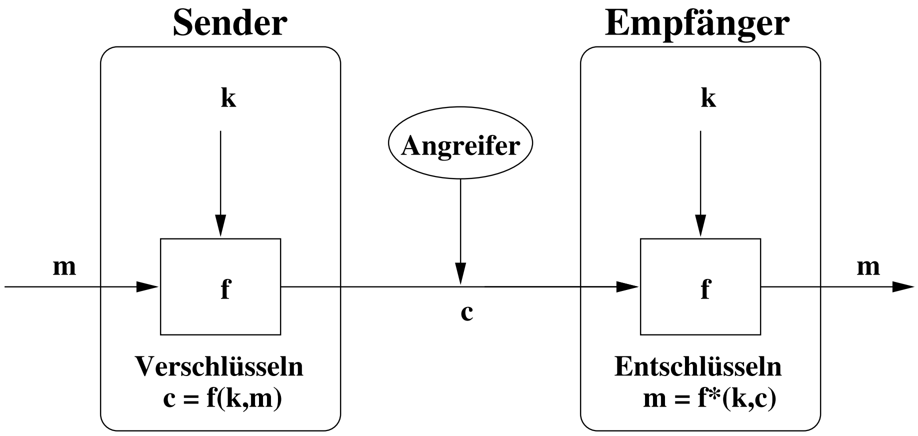

Abbildung 3.1: Ein Funktionsschema der symmetrischen Verschlüsselung. Ver- und Entschlüsselung müssen in einer sicheren Umgebung stattfinden, was im Bild durch die Kästen dargestellt ist.

- **Ciphertext-Only-Attack:** Der Angreifer kennt eine begrenzte Anzahl von Geheimtexten und möchte daraus die zugehörigen Klartexte bzw. den verwendeten Schlüssel berechnen. Dieser Angriff ist praktisch nicht zu vermeiden.

- **Known-Plaintext-Attack:** Der Angreifer kennt eine begrenzte Anzahl von Geheimtexten mit den zugehörigen Klartexten und möchte daraus den verwendeten Schlüssel berechnen. Dieses kann praktisch vorkommen.

- **Chosen-Plaintext-Attack:** Der Angreifer hat sich Zugang zu der mit dem Schlüssel k parametrisierten Verschlüsselungsfunktion f verschafft. Dabei kann er den Schlüssel k zwar nicht auslesen, wohl aber bestimmte von ihm ausgewählte Klartexte mit Hilfe von $f(k, \cdot)$ verschlüsseln. Mit Hilfe dieser Information versucht er dann andere Geheimtexte zu entschlüsseln oder den Schlüssel k zu berechnen. Diese Angriffsmöglichkeit gilt es unbedingt zu vermeiden.

- **Chosen-Ciphertext-Attack:** Der Angreifer hat sich Zugang zu der mit dem Schlüssel k parametrisierten Entschlüsselungsfunktion $f^*(k, \cdot)$ verschafft. Dabei kann er den Schlüssel k zwar nicht direkt auslesen, wohl aber bestimmte von ihm ausgewählte Geheimtexte entschlüsseln und so andere nicht von ihm gewählte Geheimtexte entschlüsseln oder den Schlüssel k zu berechnen. Auch diese Angriffsmöglichkeit ist unbedingt zu verhindern.

Dadurch wird aber auch deutlich, daß eine Geheiminformation vorher zwischen den Parteien ausgetauscht werden muß, ein Schwachpunkt, der für eine erfolgreiche Attacke auf dieses Verfahren auszunutzen ist. Folglich wäre ein Kryptosystem wünschenswert, das ohne vorherigen Austausch von Geheiminformationen auskommt. Dieser Ansatz führte dann zu den sogenannten asymmetrischen Kryptoverfahren (siehe Kapitel 3.2).

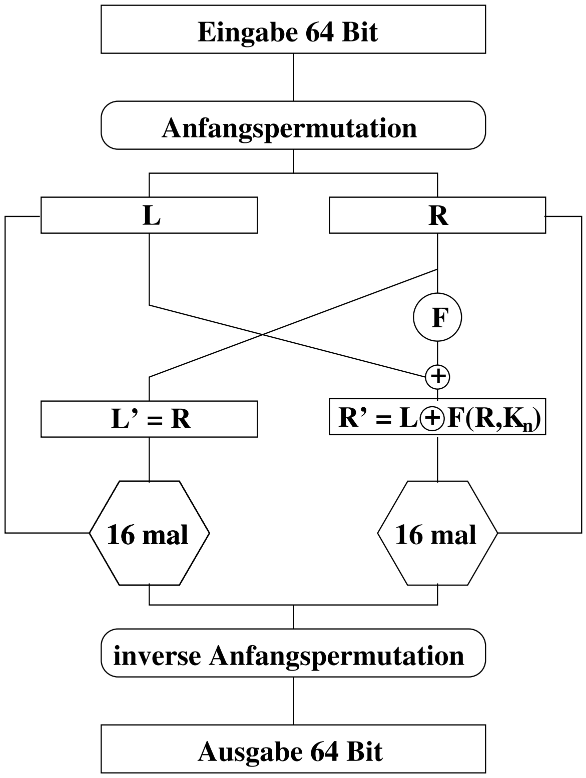

Abbildung 3.2: Funktionsschema des **D**ata **E**ncryption **S**tandard.

3.1.1 DES-Verfahren

Der Hauptvertreter dieser Kategorie ist der DES (**D**ata **E**ncryption **S**tandard). Er ist der im kommerziellen Bereich am häufigsten eingesetzte Algorithmus. So wird er zum Beispiel von den Banken auf Eurocheck-Karten eingesetzt. Beim DES-Verfahren wird die Nachricht in Blöcke mit fester Länge von 64 Bit eingeteilt, die dann nacheinander verarbeitet werden. Der zugehörige Schlüssel besteht aus 56 Bit, die durch 8 Paritätsbit ergänzt werden. Die 64 Datenbits werden nach einer Permutation in zwei Teile L und R zerlegt. Erst dann wird der Verschlüsselungsschritt, der folgendermaßen definiert ist:

For i = 1 to 16 do
$L[i] = R[i-1];$
$R[i] = L[i-1] \oplus F(R[i-1], K[i]);$ ($\oplus$: XOR)

durchlaufen. Die Funktion F besteht im wesentlichen aus der modulo-2 Addition von $R[i-1]$ und $K[i]$, nicht linearen Substitutionen und einer Permutation. Die $K[i]$ sind speziell ausgewählte Teilschlüssel des Schlüssels K. Abschließend wird die Anfangspermutation der Datenbits wieder rückgängig gemacht. Wie bei einem symmetrischen Kryptoverfahren üblich muß der Empfänger, um die Nachricht zu entschlüsseln, den Prozess umkehren, wobei die Teilschlüssel $K[i]$ in umgekehrter Reihenfolge zu verwenden sind. Dieses Verfahren ist schon auf einigen Integrierten Schaltungen realisiert worden. Man sollte aber beachten, daß die Sicherheit des DES-Algorithmus bis jetzt noch nicht zweifelsfrei bewiesen worden ist. In [HOR85] befindet sich eine ausführliche Beschreibung des DES-Verfahrens.

3.2 Asymmetrische Verfahren

Bei asymmetrischen Verfahren, die auch als **Public-Key**-Verfahren bezeichnet werden, kann jeder Sender einem Empfänger eine Nachricht senden, ohne vorher irgendeine Geheiminformation ausgetauscht zu haben. Aber nur der Empfänger ist in der Lage, die Verschlüsselung rückgängig zu machen, d. h. den Geheimtext zu entschlüsseln. Das Grundprinzip ist folgendes: Jeder Teilnehmer T des Systems besitzt einen privaten Schlüssel T_d, der geheim gehalten werden muß, und einen öffentlichen Schlüssel T_e, der möglichst allen Teilnehmern bekannt sein sollte.
Der Algorithmus f ordnet unter einem öffentlichen Schlüssel e jedem Klartext m eine Geheimtext

$$c = f_e(m)$$

zu. Umgekehrt ordnet f unter jedem privaten Schlüssel d jedem Geheimtext c einen Klartext

$$m' = f_d(c)$$

zu. Also erhält man durch die Parametrisierung der Funktion f entweder die Verschlüsselungsfunktion f_e oder die Entschlüsselungsfunktion f_d. Dabei müssen die folgenden Eigenschaften erfüllt sein:

- *Korrekte Entschlüsselung:* Bei der Entschlüsselung muß der korrekte Klartext reproduziert werden, d. h. es muß folgendes für alle Klartexte gelten:

  $$m' = f_d(c) = f_d(f_e(m)) = m$$

  Demzufolge muß die Funktion f so gewählt werden, daß obige Bedingung erfüllt wird. Die Funktion f ist dabei ein Systemparameter des verwendeten Kryptoverfahrens.

- *public-Key-Eigenschaft:* Aus der Kenntnis des öffentlichen Schlüssels e ist es praktisch unmöglich, auf den privaten Schlüssel d zu schließen. Gleiches gilt für die entsprechenden Funktionen f_e und f_d. Die Schlüssel können entweder von den Teilnehmern selbst oder von einem Systemanbieter generiert werden. Praktisch wird aber letzteres eingesetzt.

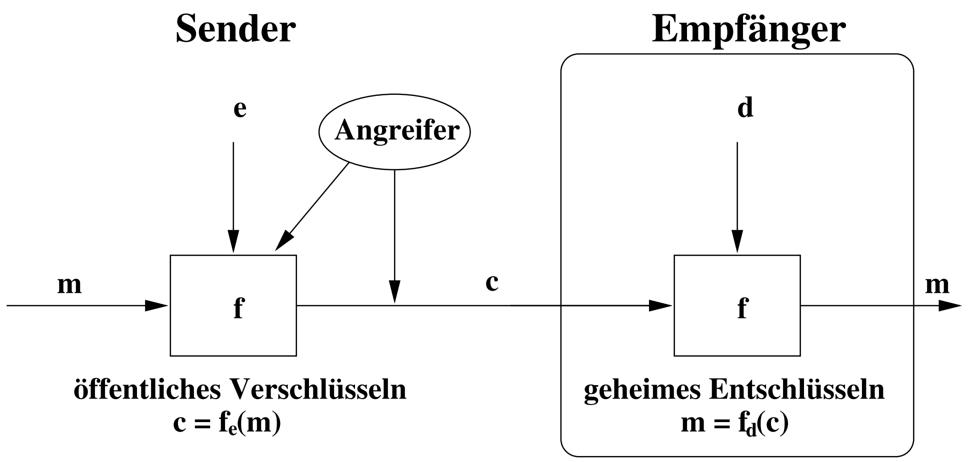

Abbildung 3.3: Funktionsschema der asymmetrischen Verschlüsselung. Nur die Entschlüsselung muß in einer sicheren Umgebung stattfinden.

Um einem Teilnehmer T eine Nachricht zu senden, nimmt man zunächst dessen öffentlichen Schlüssel $e = e_T$. Daraus ergibt sich die Funktion f_e, die dann angewendet auf die zu verschlüsselnde Nachricht m den Geheimtext c ergibt. Dieser wird dann an T geschickt, der (und nur dieser) dann mit Hilfe seines privaten Schlüssels die Funktion f_d bilden und damit letztendlich den Geheimtext entschlüsseln kann:

$$m = f_d(c) = f_d(f_e(m)).$$

Aus der Abbildung 3.3 werden folgende Vorteile von asymmetrischen Verschlüsselungssystemen deutlich:

- Kein Schlüsselaustausch notwendig
  Im Gegensatz zum symmetrischen Verfahren muß vor einer Kommunikation kein Schlüsselaustausch stattfinden. Es kann also eine spontane Kommunikation erfolgen, wenn der Anwender sich selbst seine Schlüssel generieren kann.

- Keine hohe Schlüsselanzahl notwendig
  Bei einem symmetrischen Verfahren müssen bekanntlich je zwei Teilnehmer einen Schlüssel austauschen, bei n Teilnehmern sind das bereits $\frac{n(n-1)}{2}$ Schlüssel. Somit steigt die Anzahl der Schlüssel quadratisch zu der Zahl der Teilnehmer. Demgegenüber braucht beim asymmetrischen Verfahren jeder Teilnehmer nur zwei Schlüssel, einen privaten und einen öffentlichen. Die Anzahl der Schlüssel ist also gerade doppelt so groß wie die Zahl der Teilnehmer.

- Erweiterung um neue Teilnehmer unproblematisch
  Im Gegensatz zum symmetrischen Verfahren, wo alle alten Teilnehmer mit einem neuen einen Schlüssel austauschen müssen, brauchen die Teilnehmer eines asymmetrischen Verfahrens ihre Daten nicht zu aktualisieren, d. h. sie müssen keine weiteren

Schlüssel hinzufügen. Wenn dem neuen Teilnehmer eine Nachricht geschickt werden soll, kann der öffentliche Schlüssel beim Systemverwalter wie in einer Art Telefonbuch angefragt werden.

3.2.1 RSA-Algorithmus

Das RSA-Verfahren wurde 1978 von R. **R**ivest, A. **S**hamir und L. **A**dleman am MIT Laboratory for Computer Science in Cambridge vorgestellt ([RIV78]). Es ist das meist eingesetzte und am weitesten analysierte Kryptoverfahren. Der RSA-Algorithmus basiert auf der Tatsache, eine große natürliche Zahl zu faktorisieren, also eine Primzahlzerlegung durchzuführen. Folgende Voraussetzungen werden für das RSA-Verfahren getroffen:

1. Zwei Primzahlen p und q mit $p \neq q$. Als Modul wird $n = p \cdot q$ festgelegt.
2. Zwei natürliche Zahlen $e, d < n$, für die gilt $ed \bmod \varphi(n) = 1$.
3. Sei $m \in [0 : n-1]$ eine Nachricht, ist die Nachricht m zu groß, also $m \geq n$, dann wird m in Blöcke geeigneter Länge zerlegt.

Die Schlüssel des RSA-Verfahrens lauten:

1. zum Verschlüsseln: e, n beide öffentlich
2. zum Entschlüsseln: d, n, wobei d geheim ist

Die Chiffrierfunktionen lauten:

1. Verschlüsselungsfunktion:
   $E : [0 : n-1] \longrightarrow [0 : n-1], E(x) = x^e \bmod n$
2. Entschlüsselungsfunktion:
   $D : [0 : n-1] \longrightarrow [0 : n-1], D(x) = x^d \bmod n$

Schlüsselerzeugung
Für jeden Teilnehmer werden zwei verschiedene große Primzahlen p und q gewählt, deren Produkt $n = pq$ und die Zahl $\varphi(n) = \varphi(pq) = (p-1)(q-1)$ berechnet, wobei $\varphi(n)$ die Anzahl der zu n teilerfremden natürlichen Zahlen $\leq n$ angibt (siehe Kapitel 2.5). Schließlich werden zwei Zahlen e und d mit

$$ed \bmod \varphi(n) = 1 \tag{3.1}$$

berechnet, also läßt sich die Zahl ed auch folgendermaßen darstellen:

$$ed = 1 + k\varphi(n) = 1 + k\varphi(pq) = 1 + k(p-1)(q-1) \tag{3.2}$$

für eine natürliche Zahl k.
Dann bilden n und e den öffentlichen Schlüssel, während d der geheime Schlüssel ist. Die

Zahlen p, q und $\varphi(n)$ sind für das weitere Verfahren nicht mehr notwendig und sollten aus Sicherheitsgründen auch nicht aufbewahrt werden. In der Praxis werden diese gelöscht.

Verschlüsselung
Nehmen wir an, daß die zu verschlüsselnde Nachricht eine natürliche Zahl $m \leq n$ ist. Dann ergibt sich der Geheimtext c wie folgt:

$$c = m^e \bmod n$$

Entschlüsselung
Zum Entschlüsseln muß der Empfänger seinen geheimen Schlüssel d auf den Geheimtext in folgender Weise anwenden:

$$m' = c^d \bmod n.$$

Es ist jetzt nur noch zu zeigen, daß der entschlüsselte Text $m' = m$ ist. Es gilt:

$$m' = c^d \bmod n = m^{ed} \bmod n$$

also arbeitet der Entschlüsselungsalgorithmus genau dann korrekt, wenn

$$m^{ed} \bmod n = m \tag{3.3}$$

für alle natürlichen Zahlen m gilt. Betrachtet werden nun folgende Fälle:

1. $m = 0$:
   Die Aussage ist trivial.

2. $m \in [1 : n - 1]$:
   Wegen (3.1) und (3.2) ergibt sich für $1 \leq m \leq pq$:

   $$\begin{aligned} \Rightarrow \quad & (ggt(p,m) = 1 \wedge ggt(q,m) = 1) \vee \\ & (ggt(p,m) = 1 \wedge ggt(q,m) = q) \vee \\ & (ggt(p,m) = m \wedge ggt(q,m) = 1) \end{aligned}$$

   Es bleiben also zwei Fälle zu untersuchen, nämlich der Fall, daß $ggt(p,m) = 1$ ist, und der, daß $ggt(q,m) = 1$ ist. Der dritte Fall, daß beides gilt, wird durch die beiden anderen Fälle bereits abgedeckt. Beide Fälle sind in der Argumentation identisch, so daß hier exemplarisch der Fall $ggt(p,m) = 1$ gezeigt werden soll:
   Nach dem Satz von Euler-Fermat (siehe 2.5) gilt, da p und m nach Voraussetzung teilerfremd sind:

   $$m^{\varphi(p)} = m^{(p-1)} \equiv 1 \pmod p \tag{3.4}$$

   Damit ergibt sich für m^{ed} folgende Umformungen:

$$\begin{array}{rcll} m^{ed} \bmod p & = & m^{k\cdot\varphi(n)+1} \bmod p & \text{wegen (3.1)} \\ & = & m \cdot m^{k\cdot\varphi(n)} \bmod p & \\ & = & m \cdot m^{k\cdot\varphi(pq)} \bmod p & \\ & = & m \cdot m^{k\cdot(p-1)(q-1)} \bmod p & \text{wegen } \varphi(n) = \varphi(pq) = (p-1)(q-1) \\ & = & m \cdot m^{(p-1)^{k\cdot(q-1)}} \bmod p & \\ & = & m \cdot 1^{k\cdot(q-1)} \bmod p & \text{(gilt nur, wenn } m \text{ und } p \text{ teilerfremd)} \\ & = & m \bmod p & \end{array}$$

Für den Fall, daß m und p nicht teilerfremd sind, gilt:

$$m \bmod p = 0,$$

weil p eine Primzahl ist und damit notwendigerweise ein Teiler von m sein muß. Wenn p die Zahl m teilt, dann teilt p erst recht die Zahl m^{ed}. Daraus folgt:

$$m^{ed} \bmod p = 0,$$

so daß die Kongruenz auch für $m \bmod p = 0$ trivialerweise gilt. Bei gleicher Argumentation für q ergibt sich:

$$m^{ed} \bmod p \quad = \quad m \bmod p$$

Damit ist (3.3) für $\bmod p$ und $\bmod q$ nachgewiesen. Diese beiden Teilergebnisse müssen nun für die Gültigkeit von mod n kombiniert werden. Aufgrund der obigen Umformungen kann m^{ed} auch wie folgt ausgedrückt werden:

$$(\exists j \in \mathbb{N})\, m^{ed} = jp + m \;\text{ bzw. }\; (\exists j' \in \mathbb{N})\, m^{ed} = j'q + m \tag{3.5}$$

Aus (3.5) folgt, daß $jp = j'q$ ist. Da sowohl p als auch q Primzahlen sind, gilt auch:

$$(\exists r \in \mathbb{N})(j = rq \wedge j' = rp) \tag{3.6}$$

Durch Einsetzen von (3.6) in (3.5) erhält man:

$$\begin{array}{rcll} m^{ed} \bmod n & = & (rpq + m) \bmod n & \\ & = & (rn + m) \bmod n & \text{da } pq = n \\ & = & m \bmod n & \end{array}$$

Damit ist die Gültigkeit von (3.3) gezeigt. Die Abbildung 3.4 verdeutlicht den Ablauf des RSA-Verfahrens. Der Sender verschlüsselt mit den öffentlichen Schlüsseln n und e des Empfängers die Nachricht m zu c. Dadurch, daß der Empfänger das Inverse zu e, nämlich d, kennt, kann der Empfänger leicht die Verschlüsselungsfunktion f umkehren.

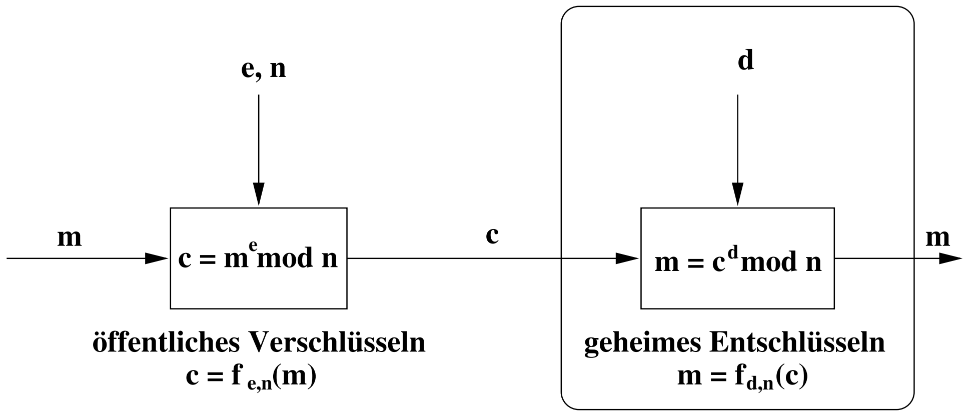

Abbildung 3.4: Funktionsschema des RSA-Verfahrens.

3.2.2 Das ElGamal-Verschlüsselungsverfahren

Dieses Verfahren geht auf Taher ElGamal zurück [ElG85]. Für dieses Verfahren benötigt man als Systemparameter eine Primzahl p und eine natürliche Zahl g, die öffentlich bekannt sind. Für das ElGamal-Verfahren gelten folgende Voraussetzungen:

1. Eine Primzahl p

2. Eine zufällige natürliche Zahl $g < p$

Die Schlüssel des ElGamal-Verfahrens lauten:

1. zum Verschlüsseln: τ, der öffentlich ist, und a, der geheim ist. Aus beiden Schlüsseln wird eine natürliche Zahl $k := \tau^a \bmod p$ berechnet, die dann in die Verschlüsselungsfunktion einfließt.

2. zum Entschlüsseln: t, der eigene geheime Schlüssel des Empfängers, und α, der zusammen mit der Nachricht übermittelt wird.

Die Chiffrierfunktionen lauten:

1. Verschlüsselungsfunktion:
   $E : f(k, m)$ mit $k := \tau^a \bmod p$ und der Nachricht m.

2. Entschlüsselungsfunktion:
   $D : f^{-1}(k, c)$ mit $k := \alpha^t \bmod p$ und der verschlüsselten Nachricht c.

Das ElGamal-Verfahren verdeutlicht Abbildung 3.5. Im ursprünglichen ElGamal-Verfahren war die Verschlüsselungsfunktion einfach die Multiplikation von k und m modulo der Primzahl p, aber dieses ist eine unnötige starke Einschränkung.

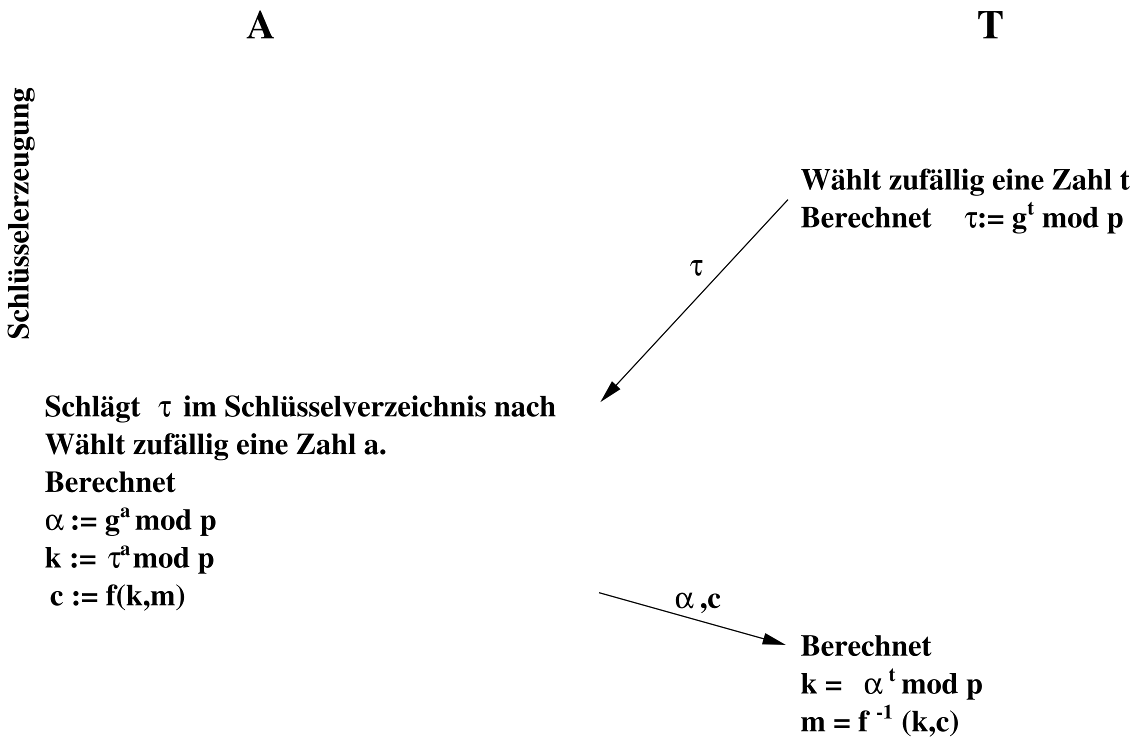

Abbildung 3.5: Das ElGamal-Verschlüsselungsverfahren

Schlüsselerzeugung
Jeder Teilnehmer wählt zufällig eine Zahl $t \in GF(p)$, die gleichzeitig der eigene geheime Schlüssel ist. Aus t wird der öffentliche Schlüssel $\alpha := g^t \bmod p$ berechnet.

Nachrichtenaustausch
Um an den Teilnehmer T eine verschlüsselte Nachricht zu schicken, geht der Sender A wie folgt vor: Er wählt eine natürliche Zahl a und berechnet

$$\alpha = g^a \bmod p \quad \text{und} \quad k = \tau^a \bmod p.$$

Die Nachricht m wird dann unter dem Schlüssel k mit Hilfe eines beiden Teilnehmern bekannten symmetrischen Algorithmus f verschlüsselt und als Geheimtext $c = f(k, m)$ zusammen mit α an T gesendet. Dieser entschlüsselt den Geheimtext c, indem er α mit seinem geheimen Schlüssel t potenziert, wodurch er k erhält, denn es gilt:

$$\begin{aligned} k &= \tau^a \bmod p \\ &= (g^t)^a \bmod p \quad \text{mit } \tau = g^t \bmod p \\ &= g^{at} \bmod p \\ &= (g^a)^t \bmod p \\ &= \alpha^t \bmod p \qquad \text{mit } \alpha = g^a \bmod p \end{aligned}$$

Mit Hilfe von k, daß nach obiger Ausführung für Sender und Empfänger identisch ist, kann dann der Empfänger $m = f^{-1}(k, c)$ berechnen. Bleibt nur noch zu zeigen, daß $m = f^{-1}(c, k)$

gilt. Um dieses zu zeigen, soll hier die ursprüngliche Funktion des ElGamal-Verfahrens $f(m,k) = (m \cdot k) \bmod p$ als Verschlüsselungsalgorithmus dienen. Somit gilt für die Entschlüsselungsfunktion $f^{-1}(c,k) = \left(\frac{c}{k}\right)$:

$$\begin{aligned} f^{-1}(c,k) &= \left(\frac{m \cdot k}{k}\right) \bmod p, \quad \text{da } c = (m \cdot k) \bmod p \\ &= m \bmod p \end{aligned}$$

Wenn ein Angreifer von dem öffentlichen Schlüssel τ auf den privaten Schlüssel t schließen wollte, dann müßte dieser den diskreten Logarithmus von τ zur Basis g berechnen. Also ist die Sicherheit des ElGamal-Verfahrens eng mit der Schwierigkeit der Berechnung des diskreten Logarithmus modulo p verbunden.

Das ElGamal-Verfahren soll nun mit Elliptischen Kurven realisiert werden. Dazu wird als Verschlüsselungsfunktion die Addition der Elliptischen Kurve genommen. Die Umsetzung des Verfahrens wird im folgenden Kapitel 4 beschrieben.

Kapitel 4

Analysiertes Kryptoverfahren

Das RSA-Verfahren ist ein relativ sicheres Kryptosystem, das gegen eine Vielzahl von Attacken analysiert wurde bzw. sich als resistent erwiesen hat. Praktisch benötigt die Verschlüsselung mit diesem Verfahren und geeignetem Schlüssel (mindestens 512 Bit, als neuer Standard wird 1024 Bit angesetzt) zuviel Zeit, um on-line Daten übertragen zu können. Außerdem werden die Prozessoren immer leistungsfähiger, so daß auch die Angreifer bessere Möglichkeiten bekommen, Kryptosysteme zu brechen. Um aber weiterhin eine gleiche Sicherheit gewähren zu können, wird man also in der Zukunft nicht um eine Verlängerung der Schlüssellänge herumkommen. Aber so führt zum Beispiel eine Verdoppelung der Schlüssellänge beim RSA-Verfahren (derzeit werden 512 Bit Schlüssellänge als sicher angenommen, aber es gibt schon Untersuchungen, wie in [ROB95], die eine Schlüssellänge von 1024 Bit propagieren) im Prinzip zu einem achtfachen Aufwand an Zeit und Hardware. Zahlen doppelter Länge erfordern bei $m^e \bmod n$ zweimal soviele Multiplikationen, die wiederum zweifachen Zeitaufwand benötigen. Insgesamt gibt das einen vierfachen Aufwand an Zeit. Außerdem sind doppelt so große Register für die doppelt so großen Zahlen notwendig, was einen doppelten Hardwareaufwand zur Folge hat.
Mit der steigenden Prozessorleistung des Angreifers geht auch eine Steigerung der Sicherheitsanforderung einher, so daß auf diesem Wege keine Geschwindigkeitssteigerung zu erzielen ist. Daher sollten Alternativen zum RSA-Verfahren gefunden werden, die die gleiche Sicherheit bieten bei geringerer Schlüssellänge oder geringerem Hardwareaufwand. Auch sollte die Steigerung der Sicherheit eine geringere Steigerung der Schlüssellänge zur Folge haben, als es bei den derzeitigen Kryptosystemen der Fall ist.
Ähnlich wie beim RSA-Verfahren muß die Primzahl des ElGamal-Verfahrens für eine sichere Verschlüsselung in derselben Größenordnung liegen wie für das RSA-Modul (entspricht n aus der Formel $m^e \bmod n$). Aber im Gegensatz zum RSA-Verfahren fließt die eigentliche Struktur des Endlichen Körpers $GF(p)$ beim ElGamal-Verfahren in den Verschlüsselungsprozeß nicht ein. Entscheidend ist lediglich, daß die diskrete Exponentiation eine Einweg-Eigenschaft aufweist, d.h. daß es nur mit sehr hohem Rechenaufwand möglich ist, die diskreten Logarithmen zu berechnen. Eine solche Exponentiation ist demnach nicht von $GF(p)$ abhängig, sondern läßt sich auch in einer mathematisch einfacher zu behandelnden Gruppe einführen. Diesen Umstand nutzt man mit der Einführung der Ellipti-

schen Kurven. Bisherige Untersuchungen von Kryptoverfahren basierend auf Elliptischen Kurven haben gezeigt ([SOP95]), daß eine Schlüssellänge von 155 Bit der Sicherheit eines RSA-Verfahrens mit 512 Bit entspricht. Außerdem wächst die Schlüssellänge nicht so stark an bei gleicher Sicherheit wie beim RSA-Verfahren. Somit scheint der Hardwareaufwand bei Kryptosystemen basierend auf Elliptischen Kurven bedeutend geringer zu sein als beim RSA-Verfahren. Diese Aussage ist dahingehend zu diskutieren, denn sonst hätten solche Systeme sich schon lange durchgesetzt und das RSA-Verfahren verdrängt. Warum der Hardwareaufwand nicht viel geringer ist, liegt an den komplexen Operatoren der Elliptischen Kurven. Für eine Addition von zwei unterschiedlichen Punkten einer Elliptischen Kurve sind immerhin 6 Subtraktionen, 2 Multiplikationen und eine Division notwendig. Ob diese komplexen Operationen Vorteile bringen, sollen die Untersuchungen des hier implementierten Kryptosystems basierend auf Elliptischen Kurven zeigen. Das hier implementierte Verfahren verwendet die gleiche Technik, die auch dem ElGamal-Verfahren zugrunde liegt (siehe hierzu 3.2.2). und wurde bei IEEE in [IEE95] vorgestellt.
Die Systemlogistik besteht bei diesem Verfahren aus der Verwaltung der Schlüssel wie in einem Telefonbuch. Zu jedem Teilnehmer des Kryptosystems kann man hier den öffentlichen Schlüssel finden, der nötig ist, um eine geheime Nachricht an einen bestimmten Adressaten zu verschicken. Damit dieses System auch funktioniert, müssen bestimmte Parameter, die *Systemparameter*, allen Teilnehmern bekannt sein. Bei dem hier zu analysierenden Verfahren sind das folgende:

1. Primzahl p für Endlichen Körper $GF(p)$
2. Parameter $a \in GF(p)$ und $b \in GF(p)$ der Elliptischen Kurve $E : y^2 = x^3 + ax + b$ über $GF(p)$
3. Basispunkt $B(x_b, y_b) \in E$ mit $x_b, y_b \in GF(p)$
4. Ordnung des Basispunktes $\#B(E(GF(p))) = n$
5. Personenspezifische Schlüssel:
   (a) Öffentlicher Schlüssel jedes Teilnehmers (x_t, y_t) mit $x_t, y_t \in GF(p)$
   (b) Geheimer Schlüssel jedes Teilnehmers $d_s \in GF(p)$

4.1 Schlüsselgenerierung

Jeder Teilnehmer bekommt zwei Schlüssel, einen öffentlichen Schlüssel T_p und einen geheimen Schlüssel d. Diese werden nach folgendem Schema generiert:

1. Wahl einer zufälligen natürlichen Zahl d aus dem Bereich $[2, n-2]$ (Die Zahlen 1 und $n-1$ sind zwar auch denkbar, doch entsprechen deren öffentliche Schlüssel $1 \cdot B$ bzw. $(n-1) \cdot B$ dem Basispunkt selbst bzw. dessen Spiegelpunkt, so daß die natürliche Zahl d leicht zu erkennen wäre.)

2. Berechnen des Punktes $T_p := d \cdot B$
3. Der öffentliche Schlüssel ist dann der Punkt $T_p = (x_t, y_t)$
4. Der geheime Schlüssel ist dann die zufällig gewählte natürliche Zahl $d_s = d$

4.2 Verschlüsselung

Wenn ein Sender S einem Empfänger R eine Nachricht senden möchte, dann muß er folgende Schritte durchführen:

1. Den öffentlichen Schlüssel $T_e = d_{s_e} \cdot B = (x_e, y_e)$ von R heraussuchen
2. Die Nachricht M in Feldelementen $m_1, m_2 \in GF(p)$ darstellen (siehe 4.2.1)
3. Wahl einer zufälligen natürlichen Zahl $k \in [2, n-2]$
4. Berechnen des Punktes $(x_1, y_1) := kB$ auf der Elliptischen Kurve
5. Berechnen des Punktes $(x_2, y_2) := kT_e$ auf der Elliptischen Kurve
6. m_1, m_2 und x_2 in einer vorgeschriebenen komplexen Weise miteinander zu den Feldelementen c_1 und c_2 verknüpfen (siehe 4.2.1)
7. Das Datentupel (x_1, y_1, c_1, c_2) an den Empfänger R übertragen (siehe 4.2.1)

Diese Vorgehensweise wird in Kapitel 4.5 an einem Beispiel veranschaulicht. Im zweiten Schritt wird die Nachricht so zerlegt, daß sie eindeutig als Element des vorgegebenen Endlichen Körpers dargestellt werden kann. Ist die Nachricht länger, so wird die Nachricht solange in Blöcke zerlegt, bis die gesamte Nachricht als Elemente von $GF(p)$ darstellbar ist. Die Verschlüsselung muß dann dementsprechend häufig durchgeführt werden. Der dritte und vierte Schritt sind analog zur Schlüsselgenerierung. Daher ist es also nicht notwendig (und manchmal nach Kapitel 4.4 auch nicht möglich), den zum öffentlichen passenden geheimen Schlüssel zu verwenden. Im fünften Schritt werden dann der zur Verschlüsselung der Nachricht M verwendete Schlüssel und der öffentliche des Empfängers zu einem neuen Punkt (x_2, y_2) auf der Elliptischen Kurve kombiniert, dessen Koordinanten dann zur Verschlüsselung der in Blöcke aufgeteilten Nachricht M dienen. Hier geht also der öffentliche Schlüssel des Empfängers R direkt in die Verschlüsselung ein. Da der Empfänger den verwendeten geheimen Schlüssel k nicht kennt, muß der Empfänger eine Möglichkeit bekommen, ohne Kenntnis von k die Nachricht M zu entschlüsseln. Diese bekommt er durch die Übermittlung von $(x_1, y_1) := kB$ (Schritt 6), woraus er den für die Entschlüsselung notwendig Punkt $(x_2, y_2) \in E$ berechnen kann, denn es gilt:

$$(x_2, y_2) = kT_e = kd_{s_e}B = d_{s_e}kB = d_{s_e}(x_1, y_1) \tag{4.1}$$

wobei d_{s_e} der geheime Schlüssel des Empfängers R ist, der zu seinem öffentlichen Schlüssel T_e gehört. Im siebten Schritt werden durch vordefinierte Verknüpfungen einige mögliche Angriffe erschwert. Wie diese Verknüpfung im Detail aussieht, wird im folgenden skizziert.

4.2.1 Einbinden der Nachricht M

Die Länge der zu übertragenden Nachricht ist durch die Wahl des Endlichen Körpers $GF(p)$ begrenzt, d.h. die zu übertragende Nachricht muß, sofern sie länger als die maximal zu übertragende Nachricht ist, in entsprechende Blöcke zerlegt werden. Dabei ergibt sich die maximal zu übertragende Nachricht wie folgt:

$$|M_{max}| = 2 * l - 3 \text{ mit } l = \lceil \tfrac{log_2 p}{8} \rceil \text{ (in Bytes)}$$
$$\text{oder}$$
$$|M_{max}| = 2 * l - 3 \text{ mit } l = \lceil log_2 p \rceil \text{ (in Bits)}$$

Vor diese Nachricht wird noch ein Trennbyte bzw. Trennbit gesetzt, um die für die Entschlüsselung wichtigen Bits kenntlich zu machen. Dadurch ist auch immer gewährleistet, daß es sich um eine gerade Anzahl zu verschlüsselnder Bytes bzw. Bits handelt. Damit der Nachrichtenblock die einheitliche Länge von $2 * l - 2$ bekommt, wird noch ein Padding String, im folgenden als PS bezeichnet, angefügt. Dieser besteht aus 1'en. Daraus ergibt sich für die Länge von PS:

$$|PS| = 2l - 3 - |M| \qquad \text{mit } |M_{max.}| = 2l - 3,$$

wobei $|M_{max.}|$ die maximale Größe des zu übertragenden Nachrichtenblocks pro Verschlüsselung ist. Wenn die Nachricht die maximale Nachrichtenlänge erreicht hat, ist $|PS| = 0$, d.h., es wird kein PS angefügt. Da aber PS nur aus Einsen besteht und die Nachricht aber durch eine 0 oder ein Nullbyte abgetrennt ist, kann die Ver- und Entschlüsselung eindeutig erfolgen.
So erhält man als zu verschlüsselnde Nachricht durch Konkatenation:

$$M' = PS \| 00 \| M \qquad \text{mit } |M_{max.}| = 2l - 3.$$

Danach wird die Nachricht M' in zwei Teilnachrichten M_1 und M_2 aufgeteilt:

$$\begin{aligned} M_1 &: \text{ ersten } l - 1 \text{ Bytes bzw. Bits} \\ M_2 &: \text{ zweiten } l - 1 \text{ Bytes bzw. Bits} \end{aligned}$$

Sowohl M_1 als auch M_2 sind kleiner als die Primzahl p, so daß keine Daten aufgrund einer Modulo-Reduktion verloren gehen können. Diese Prämisse und die eindeutige Abgrenzung der Nachricht machte eine Beschränkung auf $|M_{max}| = 2 * l - 3$ der zu übertragenden Nachricht notwendig.

1. Bilden der Feldelemente m_1 und m_2:
   - Konvertierung von M_1 und M_2 in Binärstrings
   - Anhängen jeweils eines Null-Binärstrings der Länge:

   $$|8 - 8l + t| \qquad (t = \lceil log_2 p \rceil)$$

2. Bilden von x_3:
   x_2 mit 0 im MSB (**M**ost **S**ignificant **B**it)

3. Bilden von y_3:
   $x_2^3 \in GF(p)$ mit 0 im MSB

4. Bilden von x_4:
   $\lceil \frac{t}{2} \rceil$ MSB von x_3 $\|\lfloor \frac{t}{2} \rfloor$ LSB (**L**east **S**ignificant **B**it) von y_3

5. Bilden von y_4:
   $\lceil \frac{t}{2} \rceil$ MSB von y_3 $\|\lfloor \frac{t}{2} \rfloor$ LSB von x_3

6. Berechnen von $z_1 = m_1 \oplus y_3$ und $z_2 = m_2 \oplus x_3$, wobei $\oplus$ eine bitweise XOR-Verknüpfung ist.

7. Berechnen von $c_1 = x_4 * z_1$ und $c_2 = y_4 * z_2$, wobei $*$ die Multiplikation in $GF(p)$ ist.

4.3 Entschlüsselung

Der Empfänger R der Nachricht M benutzt folgende Schritte, um aus dem ihm übersendeten Datentupel den Klartext zu decodieren:

1. Berechnen des Punktes $(x_2, y_2) = d_{s_e}(x_1, y_1)$, es gilt entsprechend der Berechnung von $d_{s_e}(x_1, y_1)$:

$$\begin{aligned} d_{s_e}(x_1, y_1) &= d_{s_e} kB \quad \text{mit } k \text{ geheimer Schlüssel des Senders} \\ &= k d_{s_e} B \quad \text{mit } T_e = d_{s_e} B \\ &= kT_e \end{aligned}$$

2. Die Feldelemente m_1 und m_2 aus c_1, c_2 und x_2 bilden

3. Aus den Feldelementen m_1 und m_2 die Nachricht M generieren

In Schritt 1 wird der Punkt berechnet, den der Sender S zum Verschlüsseln der Nachricht M verwendet hat. R kann also diesen Punkt berechnen, ohne S's geheimen Schlüssel k zu kennen. Das Dekodieren der Nachricht M aus c_1, c_2 und x_2 ergibt sich analog zu der Verschlüsselung wie folgt:

1. Bilden von x_3:
   x_2 mit 0 im MSB

2. Bilden von y_3:
   $x_2^3 \in GF(p)$ mit 0 im MSB

3. Bilden von x_4:
   $\lceil \frac{t}{2} \rceil$ MSB von x_3 $\|\lfloor \frac{t}{2} \rfloor$ LSB von y_3

4. Bilden von y_4:
   $\lceil \frac{t}{2} \rceil$ MSB von y_3 $\|\lfloor \frac{t}{2} \rfloor$ LSB von x_3

5. Berechnen von $z_1 = c_1 * x_4^{-1}$ und $z_2 = c_2 * y_4^{-1}$, wobei $*$ die Multiplikation in $GF(p)$ ist.

6. Berechnen von $m_1 = z_1 \oplus y_3$ und $m_2 = z_2 \oplus x_3$, wobei $\oplus$ eine bitweise XOR-Verknüpfung ist.

7. Entfernen der $|8 - 8l + t|$ MSB von m_1 und m_2

8. Konvertierung von m_1 und m_2 in die Teilnachrichten M_1 und M_2

Die Entschlüsselung der Nachricht aus m_1 und m_2 geschieht in umgekehrter Reihenfolge als es bei der Einbindung der Nachricht erfolgt ist.

4.4 Sonderfälle des Verfahrens

Das hier analysierte Verfahren hat einen bemerkenswerten Sonderfall. Es muß nämlich garantiert werden, daß es zu einem Zwischenergebnis x_4 und y_4 ein Inverses gibt, anderenfalls kann eine Entschlüsselung nicht erfolgen. Solange sowohl x_4 als auch y_4 ungleich Null sind, gibt es ein Inverses in $GF(p)$. Nur zu dem Feldelement 0 existiert kein Inverses. Also muß schon bei der Verschlüsselung verhindert werden, daß der Fall $x_4 = 0$ oder $y_4 = 0$ eintreten kann. Aus der Bildung von x_4 und y_4:

$$\begin{array}{rcl} x_4 & = & \lceil \frac{t}{2} \rceil \text{ MSB von } x_3 || \lfloor \frac{t}{2} \rfloor \text{ LSB von } y_3 \\ y_4 & = & \lceil \frac{t}{2} \rceil \text{ MSB von } y_3 || \lfloor \frac{t}{2} \rfloor \text{ LSB von } x_3 \end{array} \tag{4.2}$$

ergibt sich zum einen, daß bereits x_3 und y_3 ungleich Null sein müssen, zum anderen, daß die Teilkonkatenation der Bits von x_3 und y_3 ungleich Null sein müssen. Beide Fälle sollen nun näher betrachtet werden:

1. $x_3 \neq 0$ und $y_3 \neq 0$:
   Beide berechnen sich aus x_2, welches die x-Koordinate des Punktes $(x_2, y_2) = kT_e$ ist. Also kann x_2 nur dann Null sein, wenn die Muliplikation des geheimen Schlüssels k des Senders mit dem öffentlichen Schlüssel T_e des Empfängers für den x-Wert eine Null ergibt. Bei der Schlüsselwahl muß also überprüft werden, ob $kT_e \neq (0, y_2)$ ist, ansonsten muß ein anderer geheimer Schlüssel k gewählt werden. Für dieses Verfahren ist es ja nicht notwendig, daß ein Teilnehmer zum Verschlüsseln einer Nachricht unbedingt seinen geheimen Schlüssel k, der zu seinem öffentlichen Schlüssel $T_e = kB$ gehört, verwendet. Der Empfänger kann in jedem Fall die Nachricht nach Gleichung (4.1) ohne Kenntnis des geheimen und öffentlichen Schlüssels des Senders entschlüsseln.

2. Teilkonkatenation von x_3 und y_3:
   Nach Gleichung (4.2) müssen folgende Bedingungen verhindert werden, damit eine korrekte Entschlüsselung stattfinden kann:

- $\lceil \frac{t}{2} \rceil$ MSB von $x_3 = 0$ und $\lfloor \frac{t}{2} \rfloor$ LSB von $y_3 = 0$ (damit wäre $x_4 = 0$)
- $\lceil \frac{t}{2} \rceil$ MSB von $y_3 = 0$ und $\lfloor \frac{t}{2} \rfloor$ LSB von $x_3 = 0$ (damit wäre $y_4 = 0$)

Beide Punkte können nicht durch geeignete Maßnahmen im Voraus ausgeschlossen werden. Es bleibt also nichts anderes übrig, wenn sich bei der Berechnung von x_4 oder y_4 herausstellt, daß entweder $x_4 = 0$ oder $y_4 = 0$ ist, die Werte dann auf einen anderen Wert, z.B. auf 1 (wie es im Simulationsprogramm - Teil dieser Arbeit - mit einer entsprechenden Warnung realisiert ist) zu ändern. Diese Wertmanipulation muß natürlich jedem Teilnehmer bekannt sein, denn sonst wird er kaum in der Lage sein, die Nachricht richtig zu entschlüsseln.

Zusätzlich gibt es noch eine weitere Einschränkung bei der Festlegung des Basispunktes: Die y-Koordinate muß $\neq 0$ sein, anderenfalls ist eine Schlüsselgenerierung des öffentlichen Schlüssels T_e jedes Teilnehmers nicht möglich. Um den Punkt $T_e = kB$ zu berechnen, muß immer zunächst $2B$, also eine Punktverdoppelung, durchgeführt werden (siehe Kapitel 5.2.2). Dieses ist nach der Gleichung (2.8) aber nur möglich, wenn die y-Koordinate des Punktes, in diesem Fall des Basispunktes B, $\neq 0$ ist. Diese Einschränkung hat allerdings keine Auswirkungen auf die Sicherheit des Kryptoverfahrens, weil der Basispunkt B als Systemparameter allgemein öffentlich bekannt ist.
Die gleiche Einschränkung gilt auch bei der Generierung des öffentlichen Schlüssel eines Teilnehmers. Auch hier muß die y-Koordinate ungleich 0 sein, ansonsten wären keine Skalarmultiplikationen möglich.

4.5 Beispiel

Im folgenden soll exemplarisch eine Ver- und Entschlüsselung dargestellt werden. Um das Beispiel möglichst übersichtlich zu gestalten, wird hier als endlicher Körper $GF(17)$ verwendet. Dies hat eine bitweise Betrachtung zur Folge. Die Abgrenzung zwischen Nachricht und Padding String besteht hier also nur aus einem Bit, d.h. in Binärdarstellung $PS = 0$. Zur Initialisierung des Kryptosystems werden die folgenden Systemparameter festgelegt:

Endlicher Körper	:	$GF(17)$	
Elliptische Kurve E	:	$y^2 = x^3 + 1x + 2$	mit $a = 1$, $b = 2$, $\#E = 24$
Basispunkt	:	$B(13, 6)$	mit $\#B(13, 6) = 24$
Nachricht	:	$M = 4 = 100_2$	

4.5.1 Verschlüsselung

Schlüssel	:	Empfänger	:	geheim: $e_s = 11$	öffentlich: $e_p = 11 \cdot (13, 6) = (9, 14)$
		Sender	:	geheim: $s_s = 6$	öffentlich: $s_p = 6 \cdot (13, 6) = (1, 15)$

Maximale Nachrichtenzahl:

$$|M_{max}| \quad = \quad 2\lceil log_2 p\rceil - 3 \quad = \quad 2 \cdot 5 - 3 \quad = \quad 7 \text{ Bit, d.h. } M_{max} \in [0..127]$$

Zu verschlüsselnde Nachricht: $M = 4 = 100_2$
Für die Verschlüsselung ergibt sich folgendes:

1. Blockformatierung:

   - PS (Padding String):
     $|PS| \;=\; 2\lceil log_2 p\rceil - 3 - |M| \;=\; 2 \cdot 5 - 3 - 3 \;=\; 4$ Bit
   - Bilden von $M^{'}$:
     $M^{'} \;=\; PS\|0\|M \;=\; 11110100_2 \;=\; 244$
   - Bilden von $M^{'}_1$ und $M^{'}_2$ (Länge in Bits):

     $$\begin{array}{lllllllll} |M^{'}_1| & = & \lceil log_2 p\rceil - 1 \text{ MSB} & = & 5-1=4 \text{ MSB} & \Rightarrow & M^{'}_1 = 1111_2 & = & 15 \\ |M^{'}_2| & = & \lceil log_2 p\rceil - 1 \text{ LSB} & = & 5-1=4 \text{ LSB} & \Rightarrow & M^{'}_2 = 0100_2 & = & 4 \end{array}$$

2. Einbinden der Nachricht:

   - Bilden des Nullbinärstrings $0B$:
     $$\begin{array}{lll} |0B| & = & 1 - \lceil log_2 p\rceil + \lceil log_2 p\rceil \\ & = & 1 - 1 \cdot 5 + 5 = 1 \text{ Bit} \end{array}$$
   - Bilden von $m^{'}_1$ und $m^{'}_2$:
     $$\begin{array}{lllll} m^{'}_1 & = 0B\|M^{'}_1 & = & 01111_2 & = & 15 \\ m^{'}_2 & = 0B\|M^{'}_2 & = & 00100_2 & = & 4 \end{array}$$
   - Bilden des Punktes (x_2, y_2):
     $(x_2, y_2) = s_s \cdot e_p = 6 \cdot (9, 14) = (1, 15)$
   - Bilden von x_3 (x_2 mit 0 im MSB):
     $x_3 = 00001_2$
   - Bilden von y_3 (x_2^3 in $GF(p)$ und 0 im MSB):
     $y_3 = x_2^3 = 1^3 = 1 = 00001_2$
   - Bilden von x_4 und y_4:
     $$\begin{array}{lll} x_4 & = & \lceil \frac{log_2 p}{2}\rceil \text{ MSB } x_3 \| \lfloor \frac{log_2 p}{2}\rfloor \text{ LSB } y_3 \\ & = & 3 \text{ MSB } x_3 \| 2 \text{ LSB } y_3 = 000_2\|01_2 = 00001_2 = 1 \\ y_4 & = & \lceil \frac{log_2 p}{2}\rceil \text{ MSB } y_3 \| \lfloor \frac{log_2 p}{2}\rfloor \text{ LSB } x_3 \\ & = & 3 \text{ MSB } y_3 \| 2 \text{ LSB } x_3 = 000_2\|01_2 = 00001_2 = 1 \end{array}$$
   - Bilden von z_1 und z_2:
     $$\begin{array}{lll} z_1 & = & m^{'}_1 \oplus y_3 = (01111_2) \oplus (00001_2) = (01110_2) = 14 \\ z_2 & = & m^{'}_2 \oplus x_3 = (00100_2) \oplus (00001_2) = (00101_2) = 5 \end{array}$$
   - Bilden von c_1 und c_2:
     $$\begin{array}{lll} c_1 & = & x_4 \cdot z_1 = 1 \cdot 14 = 14 = (01110_2) \\ c_2 & = & y_4 \cdot z_2 = 1 \cdot 5 = 5 = (00101_2) \end{array}$$

3. Die verschlüsselte Nachricht lautet:
   $c = (x_1, y_1, c_1, c_2) = (1, 15, 14, 5) = (00001_2, 01111_2, 01110_2, 00101_2)$

4.5.2 Entschlüsselung

Der Empfänger hat vom Sender die folgende Nachricht erhalten:
$c = (x_1, y_1, c_1, c_2) = (1, 15, 14, 5) = (00001_2, 01111_2, 01110_2, 00101_2)$
Zum Entschlüsseln verfährt der Empfänger wie folgt:

1. Berechnung des Punktes (x_2, y_2):
$$\begin{aligned}(x_2, y_2) &= e_s \cdot (x_1, y_1) = 11 \cdot (1, 15)\\ &= 3 \cdot (1, 15) \quad \text{da die Ordnung } \#(1, 15) = 4\\ &= (1, 2) = (00001_2, 00010_2)\end{aligned}$$

2. Bilden von x_3 (x_2 mit 0 im MSB):
$x_3 = 00001_2$

3. Bilden von y_3 (x_2^3 in $GF(p)$ und 0 im MSB):
$y_3 = x_2^3 = 1^3 = 1 = 00001_2$

4. Bilden von x_4 und y_4:
$$\begin{aligned}x_4 &= \lceil \tfrac{log_2 p}{2} \rceil \text{ MSB } x_3 \| \lfloor \tfrac{log_2 p}{2} \rfloor \text{ LSB } y_3\\ &= 3 \text{ MSB } x_3 \| 2 \text{ LSB } y_3 = 000_2 \| 01_2 = 00001_2 = 1\\ y_4 &= \lceil \tfrac{log_2 p}{2} \rceil \text{ MSB } y_3 \| \lfloor \tfrac{log_2 p}{2} \rfloor \text{ LSB } x_3\\ &= 3 \text{ MSB } y_3 \| 2 \text{ LSB } x_3 = 000_2 \| 01_2 = 00001_2 = 1\end{aligned}$$

5. Berechnen von x_4^{-1} und y_4^{-1}:
$$\begin{aligned}x_4^{-1} &= 1^{-1} = 1 = 00001_2\\ y_4^{-1} &= 1^{-1} = 1 = 00001_2\end{aligned}$$

6. Bilden von z_1 und z_2:
$$\begin{aligned}z_1 &= c_1 \cdot x_4^{-1} = 14 \cdot 1 = 14 = 01110_2\\ z_2 &= c_2 \cdot y_4^{-1} = 5 \cdot 1 = 5 = 00101_2\end{aligned}$$

7. Berechnen von m_1' und m_2':
$$\begin{aligned}m_1' &= z_1 \oplus y_3 = (01110_2) \oplus (00001_2) = (01111_2) = 15\\ m_2' &= z_2 \oplus x_3 = (00101_2) \oplus (00001_2) = (00100_2) = 4\end{aligned}$$

8. Bilden von m_1 und m_2 durch Löschen von $0B$ (Nullbinärstring):
$$\begin{aligned}m_1 &= 1111_2 = 15\\ m_2 &= 0100_2 = 4\end{aligned}$$

9. Bilden von M':
$M' = m_1 \| m_2 = 11110100_2 = PS \| 0 \| M$

10. Entfernen von PS und 0 ergibt:
$M = 100_2 = 4$

Damit lautet die entschlüsselte Nachricht $M = 4$.

Kapitel 5

Implementation

Das hier implementierte Verfahren basierend auf Elliptischen Kurven über $GF(p)$ läßt sich in folgende drei Phasen unterteilen (siehe Abbildung 5.1 auf Seite 44):

1. Bereitstellung der Systemparameter:
   Auswahl des endlichen Körpers $GF(p)$ und Berechnung der Parameter a und b der Elliptischen Kurve $E : y^2 = x^3 + ax + b$, wobei die Ordnung $\#E$ aus Sicherheitsgründen möglichst prim bzw. zumindest einen sehr großen Primfaktor haben sollte.

2. Schlüsselgenerierung:
   Wahl der geheimen und Berechnung der zugehörigen öffentlichen Schlüssel.

3. Ver- und Entschlüsselung:
   Eigentliche Ver- und Entschlüsselung einer Nachricht.

Beim praktischen Einsatz eines solchen Systems werden die einzelnen Phasen von verschiedenen Spezialisten mit unterschiedlichen Hardwareanforderungen ausgeführt. Das hier implementierte Verfahren beinhaltet Punkt 3 vollständig und aus Punkt 2 die Generierung der öffentlichen Schlüssel, wobei die geheimen Schlüssel selbst eingegeben werden müssen. Das Simulationsprogramm führt mit den aus einer Liste gewählten Systemparametern ($GF(p)$, $E : y^2 = x^3 + ax + b$, $\#E$ und Basispunkt B), den generierten öffentlichen und den eingegebenen geheimen Schlüsseln für Sender und Empfänger, eine Ver- und Entschlüsselung einer Datei durch. Der genaue Ablauf kann der Beispielsitzung in Anhang B entnommen werden.
Bevor die wichtigsten Algorithmen näher erläutert werden, noch einige Bemerkungen zu den nicht implementierten Teilen des Kryptoverfahrens.
Die Wahl eines endlichen Körpers bereitet in der Praxis Schwierigkeiten, wenn die Primzahl möglichst groß sein soll. Bei der Bestimmung von großen Primzahlen geht man so vor, daß man zunächst eine Zahl in der gewünschten Größenordnung generiert und dann die Zahl Primzahltests unterzieht. Dabei handelt es sich meistens um einen Pseudoprimzahltest ([RIE85], d.h. man weiß nicht genau, ob es sich wirklich um eine Primzahl handelt, es ist nur sehr wahrscheinlich, d.h. die Fehlerwahrscheinlichkeit ist sehr gering.
Ist p bestimmt, muß eine geeignete Elliptische Kurve $E : y^2 = x^3 + ax + b$ berechnet werden,

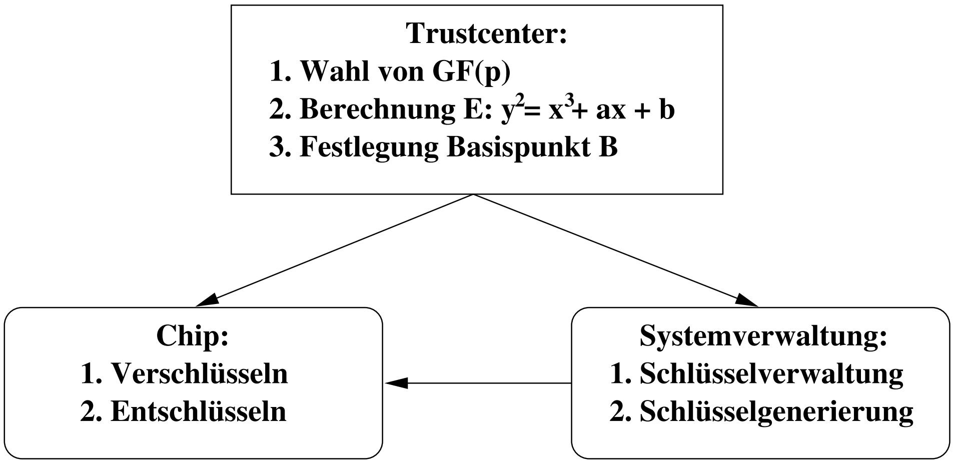

Abbildung 5.1: **Dreiteilung:** *Aufteilung der verschiedenen Komponenten des Kryptoverfahrens*

d.h. es müssen Parameter a und b so bestimmt werden, daß die Ordnung $\#E$ möglichst eine Primzahl ist oder zumindest einen sehr großen Primfaktor hat. Diese Berechnungen nehmen viel Zeit in Anspruch, weil einige Elliptische Kurven über $GF(p)$ gebildet werden, deren Ordnung $\#E$ bestimmt wird und zum Schluß überprüft werden muß, ob die berechnete Ordnung die geforderten Anforderungen erfüllt. Heuristisch betrachtet benötigt man, um eine Elliptische Krve E über $GF(p)$ mit $p > 2^{70}$ und Ordnung $\#E = prim$ zu bestimmen, auf einer Sparc ELC etwa 25 Minuten. Um die Gruppenordnung von E von Primzahlen p dieser Größenordnung zu bestimmen, benötigt das Programm aus [MUL95] im Durchschnitt 20 Sekunden. Nimmt man an, daß ein Primzahlentest unter 10 Sekunden liegt, dann kann in einer halben Minute E mit der zugehörigen Ordnung $\#E$, die keine Primzahl sein muß, bestimmt werden. Nimmt man ferner an, daß eine zufällige Wahl einer Elliptischen Kurve E zu einer zufälligen Gruppenordnung in dem Intervall $p \pm 2\sqrt{p}$ (siehe Hasse Theorem 2.9) führt (dieses ist nicht ganz richtig in Theorie und Praxis, vereinfacht die Rechnung aber etwas), so ist die Wahrscheinlichkeit laut Herrn Dr. Müller, eine Elliptische Kurve E mit einer primen Gruppenordnung $\#E$ zu erhalten, so ungefähr 2%. Damit benötigt man also 50 Versuche, was auf einer Sparc ELC 25 Minuten entspricht. Bleibt noch die Frage zu klären, ob man zu einem gegebenen $GF(p)$ mehrere verschiedene Elliptische Kurven mit geforderten Eigenschaften konstruieren kann. Dieses ist theoretisch möglich, weil in dem Intervall $p \pm 2\sqrt{p}$ nicht nur eine Primzahl existiert. Leider bereitet es laut Herrn Dr. Müller in der Praxis große Schwierigkeiten, zwei Elliptische Kurven mit der Ordnung prim zu einem $GF(p)$ zu bestimmen. Auf jeden Fall gibt es die Möglichkeit, Elliptische Kurven mit Ordnung prim zu berechnen.

5.1 Funktionen von $GF(p)$

Das hier implementierte Verfahren basiert auf einer Elliptischen Kurve E, die ihrerseits wieder über einen Endlichen Körper $GF(p)$ definiert ist. Alle Operationen der Elliptischen Kurve könnenüber Operationen in $GF(p)$ definiert werden. Im folgenden sind hier nur die komplizierteren Operationen von $GF(p)$ beschrieben. Genauere Analysen können in [COH93] nachgelesen werden.

5.1.1 Potenzieren

Das Potenzieren in $GF(p)$ unterscheidet sich von dem Methode in $\mathbb{R}$ nur dadurch, daß das Ergebnis mod p gerechnet werden muß. Der einfachste Ansatz wäre

$$a^n \bmod p = (\underbrace{a \cdot a \cdot \ldots \cdot a}_{n}) \bmod p$$

zu berechnen, wobei die $n-1$ Multiplikationen immer mod p zu reduzieren sind. Dieser Algorithmus ist für große Zahlen zu zeitintensiv und damit nicht praktikabel.
Eine viel effizientere Methode ist das **Square-and-Multiply**-Verfahren [KNU81]. Bei diesem Verfahren wird der Exponent so zerlegt, daß möglichst viele Quadrierungen ausgeführt werden können. Um zum Beispiel a^{17} mit $a \in \mathbb{N}$ zu berechnen, müssen nicht siebzehn Multiplikationen durchgeführt werden, sondern man kommt mit wesentlich weniger Gruppenoperationen aus, wenn man a^{17} wie folgt berechnet:

$$a^{17} = a^{16} \cdot a^1 = (((a^2)^2)^2)^2 \cdot a \tag{5.1}$$

Im obigen Beispiel kommt man also auch mit fünf Multiplikationen aus. Im günstigsten Fall, wenn der Exponent eine Zweierpotenz ist, werden nach diesem Verfahren $\log_2 n$, wobei hier $n \in \mathbb{Z}$ der Exponent ist, Multiplikationen benötigt. Im ungünstigsten Fall kommen noch die Anzahl von Einsen in der Binärdarstellung von n minus 1 Multiplikationen hinzu. Also müssen insgesamt $2\lfloor \log_2 |n| \rfloor$ Multiplikationen durchgeführt werden. Da die Zeitkomplexität der Multiplikation in $GF(p)$ $O(\ln^2 p)$ ist, kann das Potenzieren in $O(\ln^2 p \ln |n|)$ erfolgen. Da $\ln |n| \leq \ln p$ ist, ist die Zeitkomplexität $O(\ln^3 p)$. Diese Methode wird im folgenden Algorithmus beschrieben:

Algorithmus zum Potenzieren: Gegeben $g \in GF(p)$ und $n \in \mathbb{Z}$. Dieser Algorithmus berechnet $y = g^n \in GF(p)$, dabei bezeichnet $\mathbf{1}$ das neutrale Element von $GF(p)$.

1. Initialisierung
   (a) Setze $y \leftarrow 1$.
   (b) Wenn $n = 0$, dann gebe y aus und beende den Algorithmus.
   (c) Wenn $n < 0$, dann sei $N \leftarrow -n$ und $z \leftarrow g^{-1}$, sonst sei $N \leftarrow n$ und $z \leftarrow g$
2. Multiplizieren?
   Wenn N ungerade ist, dann ist $y \leftarrow z \cdot y$

3. N halbieren
   (a) Setze $N \leftarrow \lfloor N/2 \rfloor$
   (b) Falls $N \neq 0$, setze $z \leftarrow z \cdot z$ und gehe zu Schritt 2. Sonst gebe y als Ergebnis aus.

Die Richtigkeit des Algorithmus läßt sich am einfachsten in Schritt 2 zeigen: man hat am Anfang des 2. Schrittes immer $g^n = y \cdot z^N$, d.h die Binärstellen des Exponenten werden von rechts nach links abgearbeitet. Ist $N = 0$, dann ist der Exponent vollständig abgearbeitet und das Ergebnis ist $g^n = y \cdot z^0 = y$. Der Algorithmus terminiert immer, weil N solange halbiert wird, bis die Abbruchbedingung $\lfloor N/2 \rfloor$ erfüllt ist.

5.1.2 Inverses Element

Es gibt zwei Methoden, das Inverse b^{-1} eines Elementes $b \in GF(p)$ zu bestimmen. Die erste Methode nutzt die Tatsache aus, daß die Faktorisierung von p bekannt ist, was man bei einer Primzahl p als gegeben voraussetzen kann, die zweite verwendet den Erweiterten Euklidischen Algorithmus. Betrachte man zunächst die erste Methode:
Nach dem Satz von Euler-Fermat (siehe 2.5) gilt für eine Primzahl immer:

$$b^{\varphi(p)} \equiv 1 \pmod n$$

Damit läßt sich das Inverse zu b wie folgt berechnen:

$$b^{-1} = b^{\varphi(p)-1} \pmod n$$

Beweis:

$$b \cdot b^{-1} = b \cdot b^{\varphi(p)-1} = b^{\varphi(p)} \equiv 1 \pmod p$$

Diese Gleichung kann mit dem Algorithmus 5.1.1 zum Potenzieren gelöst werden, allerdings beträgt die Zeitkomplexität $O(\ln^3 p)$, was, wie in der zweiten Methode gezeigt wird, schlechter ist als die des Erweiterten Euklidischen Algorithmus.
Um mit dem Erweiterten Euklidischen Algorithmus das Inverse b^{-1} zu $b \in GF(p)$ zu bestimmen, wird folgende Tatsache ausgenutzt:

$$au + vc = d = ggt(a, c) \qquad u, v \in \mathbb{N}, \tag{5.2}$$

d.h. der größte gemeinsame Teiler d von a und c läßt sich als Linearkombination mit Koeffizienten u und v ausdrücken. Bezogen auf $GF(p)$ und b ergibt Gleichung (5.2):

$$bu + vp = d = ggt(b, p) \pmod p, \tag{5.3}$$

d.h., daß $u = b^{-1}$ ist, wenn $d = 1$ ist, was für jedes $b \in GF(p)$ gilt. Um also das Inverse in $GF(p)$ mit dem Erweiterten Euklidischen Algorithmus zu berechnen, muß nur der Koeffizient u bestimmt werden. Die Berechnung des Koeffizienten v ist nicht notwendig. Folgender

Algorithmus bestimmt b^{-1} basierend auf dem Erweiterten Euklidischen Algorithmus (Methode 2):

Algorithmus zur Bestimmung des Inversen: Gegeben $b \in GF(p)$, dann berechnet dieser Algorithmus $b^{-1} = ub \pmod p$ mit den Hilfsvariablen v_1, v_3, t_1 und t_3.

1. Initialisierung:
   Setze $u \leftarrow 1, d \leftarrow b, v_1 \leftarrow 0, v_3 \leftarrow p$

2. Abbruchbedingung:
   Falls $v_3 = 0$, dann ist $b^{-1} = \begin{cases} \text{u} & \text{wenn } d = 1 \\ \text{keine Lösung} & \text{wenn } d \neq 1 \end{cases}$

3. Euklidischer Schritt:
   Sei $q = \lfloor d/v_3 \rfloor$ und $t_3 = d \bmod v_3$. Setze $t_1 \leftarrow u - qv_1, u \leftarrow v_1, d \leftarrow v_3, v_1 \leftarrow t_1, v_3 \leftarrow t_3$ und gehe zu Schritt 2.

Die Korrektheit des Algorithmus läßt sich durch Einführung dreier weiterer Variablen v_2, t_2 und $v = (d - au)/p$ an folgenden Relationen zeigen, die immer zu Beginn von Schritt 2 gültig sind:

$$bt_1 + pt_2 = t_3, \quad bu + pv = d, \quad bv_1 + pv_2 = v_3 \tag{5.4}$$

Damit die Relationen (5.4) nach der Initialisierung gültig sind, genügt es $v = 0$ und $v_2 = 1$ zu setzen. Die Gültigkeit der Relationen wird auch nach Schritt 3 bewahrt, wenn die drei Hilfsvariablen durch folgende Zuweisungen angepaßt werden:

$$v_2 = t_2 \quad t_2 = v - qv_2 \quad v = v_2 \tag{5.5}$$

Durch die Zuweisungen in Schritt 3 ergeben sich für die Relationen aus Gleichung (5.4):

- $au + pv = d$:
  Durch Einsetzen von $u = v_1$ und $d = v_3$ ergibt sich: $bv_1 + pv = v_3$. Diese Gleichung erfüllt die Relation $bv_1 + pv_2 = v_3$ nur dann, wenn $v = v_2$ ist.

- $bv_1 + pv_2 = v_3$:
  Durch Einsetzen von $v_1 = t_1$ und $v_3 = t_3$ ergibt sich: $bt_1 + pv_2 = t_3$. Diese Gleichung erfüllt die Relation $bt_1 + pt_2 = t_3$ nur dann, wenn $v_2 = t_2$ ist.

- $bt_1 + pt_2 = t_3$:
  Durch Einstzen von $t_1 = u - qv_1$ und $t_3 = d \bmod v_3$ ergibt sich: $b(u - qv_1) + pt_2 = d \pmod{v_3}$. Durch folgende Umformungen ergibt sich für t_2:

$$\begin{aligned}
pt_2 &= d - b(u - qv_1) \pmod{v_3} \\
&= d - (bu - bqv_1 + pv - pv) \pmod{v_3} \\
&= d - (d - bqv_1 - pv) \pmod{v_3} && \text{mit } bu + pv = d \\
&= pv + q(v_3 - pv_2) \pmod{v_3} && \text{mit } bv_1 + pv_2 = v_3 \Leftrightarrow bv_1 = v_3 - pv_2 \\
&= pv - pqv_2 \pmod{v_3} \\
t_2 &= v - qv_2
\end{aligned}$$

Am Ende des Algorithmus ist d der größte gemeinsame Teiler und man erhält das Inverse $b^{-1} = u$. Die Zeitkomplexität des Euklidischen Algorithmus ist $O(\ln^2 p)$. Maximal sind zur Berechnung von b^{-1} $O(\ln p)$ euklidische Schritte notwendig. Die benötigte Zeit für die Berechnung eines euklischen Schrittes $a = bq + r$ ist $O((\ln a)(\ln q + 1))$. Somit ergibt sich eine Zeitkomplexität von $O(\ln p)\left(\sum \ln q + O(\ln p)\right)$ und mit $\sum \ln q = \ln \prod q \leq \ln a \leq \ln p$ ergibt sich die Zeitkoplexität $O(\ln^2 p)$.

5.1.3 Quadratischer Rest (Legendre Symbol)

Sei p eine Primzahl, dann hat die Gleichung

$$x^2 = a \bmod p$$

in $GF(p)$ entweder keine Lösung (kein quadratischer Rest mod p), genau eine Lösung $a = 0 \bmod p$ oder zwei Lösungen (a wird als quadratischer Rest mod p bezeichnet, sofern $\sqrt{a}$ existiert). Dann wird das Legendre Symbol wie folgt definiert:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{wenn } a \bmod p = 0 \\ 1 & \text{wenn } a \text{ ein quadratischer Rest ist} \\ -1 & \text{wenn } a \text{ kein quadratischer Rest ist} \end{cases}$$

Das Legendre Symbol hat folgende Eigenschaften ([COH93]):

1. Das Legendre Symbol ist multiplikativ, d.h. es gilt:

$$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right). \tag{5.6}$$

   Also ergibt die Multiplikation zweier nicht quadratischer Reste einen quadratischen Rest.

2. Es gilt folgende Kongruenz:

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \pmod{p}. \tag{5.7}$$

   Es gibt gleichviele quadratische wie nicht quadratische Reste, nämlich $(p-1)/2$.

Die einfachste Methode wäre das Ausnutzen der Kongruenz (5.7). Benutzt man den Algorithmus aus 5.1.1, dann kann das Legendre Symbol in $O(\ln^3 p)$ berechnet werden. Diese Zeitkomplexität läßt sich aber noch mit Hilfe der Reziprok-Gesetze von Legendre und Gauss vermindern:

Reziprok-Gesetze von Legendre und Gauss:
Sei p eine Primzahl. Dann gelten folgende Gesetze:

1.
$$\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}, \quad \left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8} \tag{5.8}$$

2. Sei q ebenfalls eine Primzahl mit $q \neq p$, dann lautet das Reziprok Gesetz:
$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{(p-1)(q-1)/4} \ . \tag{5.9}$$

Algorithmus Bestimmung Legendre Symbol: Gegeben $a, p \in \mathbb{Z}$ und $p \geq 2$, dann berechnet dieser Algorithmus das Legendre Symbol $\left(\frac{a}{p}\right)$.

1. Initialisierung
   Setze $v \leftarrow 0$, $k \leftarrow 1$. Falls $a < 0$, setze $a = -a$. Falls $p \bmod 4 \equiv 3$, setze $k = -k$

2. Ende?
   Falls $a = 1$, gebe k als Lösung aus. Anderenfalls setze $v \leftarrow 0$ und solange a gerade ist, $v \leftarrow v + 1$. Wenn v ungerade ist, setze $k \leftarrow (-1)^{(p^2-1)/8}k$

3. Reziprok verwenden
   Setze $k \leftarrow (-1)^{(a-1)(b-1)/4} \cdot k$ und $r \leftarrow |a|$, $a \leftarrow b \bmod r$, $b \leftarrow r$ und gehe zu Schritt 2

Nach [COH93] hat dieser Algorithmus die gleiche Zeitkomplexität wie der Euklidische Algorithmus, nämlich $O(\ln^2 p)$.

5.1.4 Wurzel in $GF(p)$

Die Berechnung der Wurzel in einem Endlichen Körper $GF(p)$ ist nicht einfach und daher auch sehr zeitintensiv. Deshalb ist es sinnvoll, zunächst mit dem gerade zuvor vorgestelltem Verfahren zu überprüfen, ob es sich bei dem Element um einen quadratischen Rest handelt. Erst wenn das Vorliegen dieses Falls nachgewiesen worden ist, gibt es garantiert eine Lösung. Sobald man eine Lösung bestimmen konnte, ist das Ausrechnen der zweiten Lösung nicht mehr schwierig. Es gilt folgender Zusammenhang zwischen der ersten Lösung l_1 und der zweiten l_2:

$$l_2 = p - l_1$$

Sei x das Element, aus dem die Wurzel zu ziehen ist. Dann muß gelten:

$$\begin{array}{lcll}
x &=& l_2^2 \bmod p & \\
x &=& (p - l_1)^2 \bmod p & \text{einsetzen von } l_2 = p - l_1 \\
x &=& (p^2 - 2l_1p + l_1^2) \bmod p & \\
x &=& (p^2 \bmod p - 2l_1p \bmod p + l_1^2 \bmod p) \bmod p & \\
x &=& (0 - 0 + l_1^2) \bmod p = l_1^2, & \text{da } l_1 < p \text{ ist}
\end{array}$$

Daher genügt es also, eine Lösung zu bestimmen. Für die Berechnung der Wurzel werden die Primzahlen in drei Kategorien eingeteilt, für die es jeweils ein Algorithmus gibt. Die Einteilung in drei Kategorien erfolgt aus folgender Überlegung:
Die größte Primzahl $p < 10$ ist 7. Um also alle Primzahlen in möglichst wenig Kategorien einteilen zu können, müssen die Primzahlen modulo 8 gerechnet werden. Es ergeben sich somit acht Kategorien, nämlich die Gruppen, die modulo 8 die Reste $0, 1, 2, 3, 4, 5, 6$ und 7 ergeben. Da aber alle betrachteten Primzahlen > 2 und damit sind, scheiden die Ergebnisse $0, 2, 4$ und 6 aus. Von den restlichen 4 Kategorien können die Reste 3 und 7 zusammengefaßt werden, denn der Fall $p \bmod 8 = 7$ entspricht dem Fall $p \bmod 4 = 3$. Somit bleiben die folgenden drei Kategorien übrig:

1. **Erste Kategorie:** $p \bmod 4 = 3$
   In diesem Fall ist eine Lösung für x^2 mod p $= a$:

   $$x = a^{(p+1)/4} \bmod p$$

   Wenn a ein quadratischer Rest ist, dann gilt $a^{(p-1)/2} \equiv 1 \bmod p$, also erhält man durch Umformen:

   $$x^2 = (a^{(p+1)/4})^2 = a^{(p+1)/2} = a \cdot a^{(p-1)/2} = a \bmod p$$

2. **Zweite Kategorie:** $p \bmod 8 = 5$
   Da $a^{(p-1)/2} \equiv 1 \bmod p$ gilt und man sich in einem Endlichen Körper $GF(p)$ befindet, wird aufgrund von $\sqrt{1} = \pm 1$ auch folgendes existieren:

   $$a^{(p-1)/4} \bmod p \equiv \pm 1$$

   Nun hängt es vom Vorzeichen ab, welche Lösung existiert. Betrachten wir zunächst den Fall $+$. Dann lautet die Lösung:

   $$x = a^{(p+3)/8} \bmod p$$

   Durch folgende Umformungen kann man die Korrektheit nachweisen:

   $$x^2 = (a^{(p+3)/8})^2 = a^{(p+3)/4} = a \cdot a^{(p-1)/2} = a \bmod p$$

   Bleibt noch das Vorzeichen $-$ zu betrachten. Hierfür lautet dann die Lösung:

   $$x = 2a \cdot (4a)^{(p-5)/8} \bmod p$$

   Auch diese läßt sich durch einfaches Umformen nachweisen. Wichtig ist dabei, daß $2^{(p-1)/2} \equiv -1 \bmod p$ gilt:

$$\begin{aligned}
x^2 &= (2a \cdot (4a)^{(p-5)/8})^2 &&= (2a)^2 \cdot (4a)^{(p-5)/4} \\
&= 4a^2 \cdot 4^{(p-5)/4} a^{(p-5)/4} &&= 2^2 \cdot 2^{2(p-5)/4} \cdot a^2 \cdot a^{(p-5)/4} \\
&= 2^{2+(p-5)/2} \cdot a^{2+(p-5)/4} &&= 2^{(p-1)/2} \cdot a^{(p+3)/4} \\
&= -1 \cdot a \cdot -1 &&= a \bmod p
\end{aligned}$$

3. **Dritte Kategorie:** $p \bmod 8 = 1$
Dieses ist der schwerste Fall. Der im Simulationsprogramm implementierte Algorithmus ist ein probabilistischer von Tonelli und Shanks (siehe [COH93]).
Jede Primzahl p läßt sich in der folgenden Weise ausdrücken:

$$p - 1 = 2^e \cdot q \quad \text{mit } e, q \in GF(p) \text{ und } q \text{ ungerade}$$

Die multiplikative Gruppe $(\mathbb{Z}/p\mathbb{Z})$ ist isomorph zu der additiven Gruppe $\mathbb{Z}/(p-1)\mathbb{Z}$. Man erhält daher eine zyklische Untergruppe G der Ordnung 2^e, zu der man einen Generator z finden kann. Alle Quadrate dieser Gruppe G haben eine Ordnung, die durch 2^{e-1} teilbar ist. Wenn a ein quadratischer Rest mod p ist, da

$$a^{(p-1)/2} = a^{(2^e \cdot q)/2} = a^{(2^e 2^{-1} q)} = (a^q)^{(2^{e-1})} \equiv 1 \pmod{p} \tag{5.10}$$

gilt, muß $b = a^q \bmod p$ ein Quadrat in G sein. Mit anderen Worten es existiert eine natürliche Zahl $0 \leq k < 2^e$ mit folgender Eigenschaft:

$$a^q z^k = 1 \text{ in } G \text{ und } z \text{ Generator in } G$$

Die Lösung für x aus unserer Ausgangsgleichung $x^2 = a \bmod p$ lautet dann:

$$x = a^{(q+1)/2} z^{k/2}$$

Um zu zeigen, daß dieses wirklich eine Lösung von x ist, berechnet man einfach x^2:

$$x^2 = (a^{(q+1)/2} z^{k/2})^2 = a^{q+1} z^k = a \cdot a^q z^k = a$$

Es bleiben demnach also zwei Probleme zu lösen: Finden eines Generators in G und Berechnen des Exponenten k. Der probabilistische Teil des Algorithmus ist das Finden eines Generators z in G. Dazu berechnet man durch zufällige Wahl von n $z = n^q \bmod p$, wobei z genau dann ein Generator in G ist, wenn n kein quadratischer Rest in $GF(p)$ ist. Ist n kein quadratischer Rest $\bmod p$, dann ist $z^{2^{e-1}} = n^{q^{2^{e-1}}} = -1$ in G und somit werden alle Elemente von n^q in G generiert. Die Wahrscheinlichkeit in $GF(p)$ ein n zu finden, daß kein quadratischer Rest $\bmod p$ ist, beträgt nach Gleichung (5.7) $\frac{p-1}{2p} \approx \frac{1}{2}$. Demnach beträgt die Wahrscheinlichkeit, nach 20 Versuchen kein solches n zu finden, bereits weniger als 10^{-6}. Den Exponenten k zu finden ist zwar schwieriger als einen Generator z zu finden, wird aber nicht explizit zur Berechnung der Wurzel benötigt, sondern nur $a^{(q+1)/2} z^{k/2}$

Algorithmus $\sqrt{x} \bmod p$ **für** $p \bmod 8 = 1$**:** Sei p eine Primzahl, $p - 1 = 2^e \cdot q$ und $a \in \mathbb{Z}$. Dann berechnet der folgende Algorithmus $\sqrt{x}$ aus der Gleichung $x^2 = a \bmod p$, sofern eine Lösung überhaupt existiert.

1. Finde Generator
   Wähle solange zufällig eine Zahl n, bis $(\frac{n}{p}) = -1$ gilt (n ist kein quadratischer Rest). Setze dann $z \leftarrow n^q \bmod p$.

2. Initialisierung
   Setze $y \leftarrow z$, $r \leftarrow e$, $x \leftarrow a^{(q-1)/2} \bmod p$, $b \leftarrow ax^2 \bmod p$ und $x \leftarrow ax \bmod p$.

3. Finde Exponenten
   Wenn $b \equiv 1 \bmod p$, gebe x als Ergebnis aus und beende den Algorithmus. Sonst finde den kleinsten Exponenten $m \geq 1$ für den $b^{2^m} \equiv 1 \bmod p$ gilt. Wenn $m = r$ ist, dann ist a kein quadratischer Rest modulo p.

4. Reduziere Exponenten
   Setze $t \leftarrow y^{2^{r-m-1}}$, $y \leftarrow t^2$, $r \leftarrow m$, $x \leftarrow xt$ und $b \leftarrow by$. Alle Operationen erfolgen mod p.

Am Anfang von Schritt 3 gelten immer folgende Kongruenzen modulo p:

$$ab \equiv x^2 \qquad y^{2^{r-1}} \equiv -1 \qquad b^{2^{r-1}} \equiv 1 \tag{5.11}$$

Wenn G_r eine Untergruppe von G ist, deren Elemente eine durch 2^r teilbare Ordnung haben, dann besagen die Kongruenzen, daß y und b Generatoren von G_r bzw. b von G_{r-1} sind, womit b gleichzeitig ein Quadrat in G_r ist. Die maximale Anzahl der Schleifen ist, da r in jeder Schleife reduziert wird, e. Wenn $r \leq 1$ ist, ist $b = 1$. Also ist x einer der Wurzeln von $a \bmod p$. Die Zeitkomplexität beträgt $O(\ln^4 p)$, weil die Schritte 3 und 4 maximal e^2 Multiplikationen mod p benötigen.

5.2 Funktionen zu Elliptischen Kurven

Um überhaupt irgendwelche Operationen zu implementieren, muß die Repräsentation von Punkten auf Elliptischen Kurven festgelegt werden. Jeder Punkt besteht aus einer x und einer y Koordinate. Das reicht aber noch nicht aus, denn mit dieser Vorgabe kann man nicht den Unendlichkeitspunkt O, also das neutrale Element bezüglich der Addition bei Elliptischen Kurven, eindeutig darstellen. Dieses Problem wird dadurch umgangen, daß zusätzlich zu den Koordinaten ein Flag eingefügt wird, welches angibt, ob es sich um den Unendlichkeitspunkt O handelt (Flag u_punkt = 1) oder nicht (Flag u_punkt = 0). Somit ergibt sich also folgende Datenstruktur:

```
struct punkt {
  Integer x;       /* X-Koordinate des Punktes */
  Integer y;       /* Y-Koordinate des Punktes */
```

```
  int u_punkt;      /* 0: kein Unendlichkeitspunkt, */
                    /* 1: Unendlichkeitspunkt */
};
```

Sobald das Flag *u_punkt* gesetzt ist, interessieren die Einträge für die Koordinaten nicht mehr. Dieses wird auch in den nun folgenden Prozeduren so umgesetzt.

5.2.1 Addition zweier Punkte

In 2.3.1 wurde die Addition von beliebigen Punkten auf der Elliptischen Kurve E definiert, deren Implementierung analog zu der Definition erfolgt. Dabei sei der Punkt $R \in E$ immer das Ergebnis der einzelnen Fälle.

1. $O + O = O$:
   Dieses ist mit der obigen Datenstruktur leicht zu realisieren. Es muß ja nur das Flag *u_punkt* der Punkte jeweils auf 1 überprüft werden. Das Ergebnis ist dann ebenfalls der Unendlichkeitspunkt O, also ist im resultierenden Punkt R ebenfalls das Flag *u_punkt* zu setzen. Der x und der y Koordinate werden einfach die Werte 0 zugeordnet, so daß das Ergebnis folgendermaßen aussieht:

   ```
   struct punkt r {
     r.x = 0;
     r.y = 0;
     r.u_punkt = 1;
   };
   ```

2. $(x, y)+ O = (x, y)$
   Auch hier müssen zuerst die Flags *u_punkt* der Punkte auf 1 überprüft werden. Sobald nur genau ein Flag *u_punkt* auf 1 gesetzt ist, wird der andere Punkt als Lösung ausgegeben. Anderenfalls erhält man entweder den Fall 1 oder den folgenden Fall 3. Das Ergebnis Punkt R sieht dann so aus:

   ```
   struct punkt r {
     r.x = x;
     r.y = y;
     r.u_punkt = 0;
   };
   ```

3. $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$

   (a) $x_1 = x_2$ und $y_1 = -y_2$
   Dieser Fall ist natürlich trivial, denn das Ergebnis ist wieder der Unendlichkeitspunkt O. Also wird das Flag u_punkt des resultierenden Punktes R auf 1 gesetzt und die Koordinaten auf 0. Damit ergibt sich:

```
struct punkt r {
  r.x = 0;
  r.y = 0;
  r.u_punkt = 1;
};
```

(b) $x_1 = x_2$ und $y_1 = y_2$
Hier handelt es sich um die Verdoppelung eines Punktes. Analog zu der Definition der Addition auf Elliptischen Kurven müssen nun die neuen Koordinaten des Ergebnispunktes R berechnet werden. Dann ergibt sich für den Punkt R folgende Struktur:

```
struct punkt r {
  r.x = ((3x_1^2+a)/(2y_1))^2 - 2x_1;
  r.y = (3x_1^2+a)/(2y_1) (x_1 - r.x) - y_1;
  r.u_punkt = 0;
};
```

Eine Verdoppelung eines Punktes umfaßt demnach insgesamt 4 Additionen (Subtraktionen und Additionen zusammengefaßt), 6 Multiplikationen und eine Division in $GF(p)$.

(c) $x_1 \neq x_2$
Auch hier wird analog zur Definition der Addition der Punkt R berechnet. So kommt man direkt zu dem Punkt R, der dann folgende Struktur hat:

```
struct punkt r {
  r.x = ((y_2-y_1)/(x_2-x_1))^2 - x_1 - x_2;
  r.y = (y_2-y_1)/(x_2-x_1) (x_1 - r.x) - y_1;
  r.u_punkt = 0;
};
```

Die Addition zweier beliebiger Punkte auf E umfassen demnach insgesamt 6 Subtraktionen, 2 Multiplikationen und eine Division in $GF(p)$.

5.2.2 Multiplikation eines Punktes mit einer natürlichen Zahl

Die Multiplikation eines Punktes $P \in E$ mit einem Skalar wird wie folgt definiert:

$$k \cdot P = \underbrace{P + P + \ldots + P}_{k} \qquad k > 2, k \in \mathbb{N}, P \in E$$

Ein trivialer Ansatz, dieses Problem zu lösen, wäre den Punkt zunächst zu Verdoppeln und dann $k-2$ mal P aufzuaddieren. Wie aber schon zuvor gezeigt, ist die Addition von Punkten auf Elliptischen Kurven sehr komplex. Und wenn man dazu noch bedenkt, daß bei dem hier implementierten Kryptosystem eine große natürliche Zahl mit einem Basispunkt $B \in E$ multipliziert werden muß, dann ist hier unbedingt eine Verbesserung erforderlich. Also muß ein Schema gefunden werden, daß einige Additionen bereits zusammenfaßt. Ähnlich dem Square-and-Multiply-Verfahren beim Potenzieren in $GF(p)$, um a^k zu finden, kann man die gleiche Vorgehensweise verwenden, um $k \cdot P$ in E zu berechnen. Wie der Name schon sagt, wird k so zerlegt, daß möglichst viele Quadrierungen, bei Elliptischen Kurven Punktverdoppelungen, auftreten. Wie dieses aussieht, soll folgendes Beispiel demonstrieren:

$$\begin{aligned}
100 \cdot P &= 2(50 \cdot P) & &= 2(2(25 \cdot P)) \\
&= 2(2(P + 24 \cdot P)) & &= 2(2(P + 2(12 \cdot P))) \\
&= 2(2(P + 2(2(6 \cdot P)))) & &= 2(2(P + 2(2(2(3 \cdot P))))) \\
&= 2(2(P + 2(2(2(P + 2 \cdot P)))))
\end{aligned}$$

Letztendlich werden also nur noch 6 Verdoppelungen und 2 Additionen benötigt, im Gegensatz zu unserem ersten Ansatz, der eine Verdoppelung und 98 Additionen benötigt hatte, was in diesem Beispiel etwa einer Geschwindigkeitssteigerung um Faktor 12 entspricht. Natürlich ist der Erfolg dieser Methode nicht immer so gravierend wie in diesem Beispiel. Im ungünstigsten Fall ist k eine Zahl, die gleichviele Verdoppelungen wie Additionen erfordert, wie folgendes Beispiel eindrucksvoll bestätigt:

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

Aber auch hier zeigt sich eine extreme Einsparung an Additionen. Demnach müssen im ungünstigsten Fall $2\log_2 k - 2$ Operationen ausgeführt werden, von denen jeweils $\log_2 k - 1$ Verdoppelungen bzw. Additionen sind. Im günstigsten Fall ist k eine Potenz von zwei. Dann sind nur noch $log_2 k$ Verdoppelungen notwendig.
Um das obige Schema in einer Prozedur umzusetzen, muß die Zerlegung von k in einem Array der maximalen Länge $2\log_2 k - 2$ abgespeichert werden. Dieses geschieht am einfachsten, indem, wenn k gerade ist, eine 1 in das Array eingetragen wird und eine 0, wenn k ungerade ist. danach muß k entsprechend koorigiert werden, d.h. durch 2 dividiert werden, wenn k gerade war, oder 1 subtrahieren, wenn k ungerade war. Für $15 \cdot P$ ergibt sich dann folgende Arraystruktur:

$$\{0, 1, 0, 1, 0, 1\}$$

Dieses Array wird dann in umgekehrter Reihenfolge abgearbeitet, d.h. bei einer 1 wird eine Punktverdoppelung und bei einer 0 eine Addition durchgeführt.

Kapitel 6

Eigenschaften des implementierten Verfahrens

Bevor eine Analyse des in Kapitel 4 vorgestellten Verfahrens vorgenommen wird, noch einige Vorbemerkungen zu den folgenden Ergebnissen. Allen Untersuchungen liegen immer nur Elliptische Kurven zugrunde, deren Ordnung $\#E =$ prim ist. Daher ist die Wahl eines Basispunktes unproblematisch, weil alle Punkte die gleiche Ordnung $\#P = \#E$ haben. Zu jedem $GF(p)$ wurde jeweils nur eine Elliptische Kurve untersucht. Es ist aber prinzipiell möglich, zu jedem $GF(p)$ mehrere Elliptische Kurven zu bestimmen, deren Ordnung $\#E =$ prim ist. Die hier verwendeten Elliptischen Kurven sind von einem Programm, welches in [MUL95] beschrieben worden ist, berechnet worden. Das Programm berechnet bei Eingabe von a, b (Parameter a und b der Elliptischen Kurve $y^2 = x^3 + ax + b$) und Primzahl p die Ordnung $\#E$. Diese Ordnung muß nicht zwangsläufig prim sein. Um eine solche zu bestimmen, wird folgendermaßen vorgegangen:

1. Wähle zufällig eine Elliptische Kurve E.
2. Bestimme $\#E$
3. Falls $\#E =$ prim, gebe die Elliptische Kurve als Lösung aus. Falls $\#E \neq$ prim, gehe zu 1

Alle Elliptischen Kurven, die im Rahmendieser Arbeit analysiert werden, haben die Ordnung prim (siehe Anhang B).
Alle Zeiten, die in den folgenden Untersuchungen angegeben werden, beziehen sich ausschließlich auf reine CPU-Rechenzeit, d.h. es wird nur die Zeit angegeben, in der die CPU des Rechners mit der Berechnung beschäftigt war. Alle Zeiten sind auf einem Pentium 133 MHz unter dem Betriebssystem LINUX berechnet worden. Die Zeiten werden anhand von regelmäßigen Impulsen der CPU, auch als Ticks bezeichnet, gemessen und entsprechend in Millisekunden umgerechnet.
Für die Analyse des implementierten Kryptoverfahrens sind folgende Fragestellungen interessant:

1. Komplexität des Verfahrens:
   Die für dieses Verfahren aufwendigen Algorithmen sind bereits in Kapitel 5 untersucht worden. Komplexitätsbetrachtungen sind außerdem kein Bestandteil der analytischen Betrachtungen und werden hier daher auch nicht weiter vertieft. Trotzdem wird bei der Analyse der Elliptischen Kurven auf die Komplexitätsbetrachtungen zurückgegriffen.

2. Rechenzeiten des Systems:
   Wichtig für die praktische Einsetzbarkeit eines Kryptoverfahrens ist die Laufzeit. Ziel ist eine möglichst schnelle Ver- und Entschlüsselung bei höchst möglichem Sicherheitsstandard. Daher ist eine Analyse verschiedener Elliptischer Kurven über unterschiedlichen Endlichen Körpern der Form $GF(p)$ notwendig (siehe Kapitel 6.1).

3. Abhängigkeiten der Rechenzeit:
   Die Rechenzeiten sind nach der Struktur des hier analysierten Kryptoverfahrens aber nicht nur von der Wahl der Primzahl p abhängig. Daher sind auch Abhängigkeiten der Rechenzeit von der Wahl der Schlüssel (siehe Kapitel 6.2) und der Nachricht M (siehe Kapitel 6.3) zu untersuchen. Die Analyse erfolgt hier unter Beibehaltung aller Parameter bis auf genau den zu untersuchenden.

4. Hardwareabschätzung:
   Die Hardwareabschätzung befaßt sich mit der Umsetzung der Operationen in eine Hardwarestruktur. Eine wichtige Größe ist dabei die Anzahl der benötigten Speichereinheiten zur Umsetzung der Ver- und Entschlüsselung. Ausgegangen wird bei der Abschätzung von einer ALU (**A**rithmetic **L**ogical **U**nit), die einen kleinen Operationssatz zur Verfügung hat (siehe Kapitel 6.4).

6.1 Rechenzeiten in Abhängigkeit von $GF(p)$

Eine der grundlegenden Untersuchung des in Kapitel 4 vorgestellten Verfahrens ist die benötigte Zeit für eine Ver- und Entschlüsselung in Abhängigkeit des gewählten Endlichen Körpers $GF(p)$.
Hierzu wurden mit zufällig gewählten Schlüsseln zufällige Zahlen, die Nachrichten repräsentieren, ver- und entschlüsselt. Für jede Elliptische Kurve aus Anhang B wurden mehrere Ver- und Entschlüsselungen durchgeführt und dann über alle diese ein Mittelwert der benötigten Rechenzeit berechnet. Dabei wurde die Generierung der geheimen und öffentlichen Schlüssel der Teilnehmer nicht berücksichtigt. Lediglich die Berechnung des Schlüssels zur Verschlüsselung der Nachricht, die ja nicht unbedingt dem geheimen Schlüssel entsprechen muß und in manchen Fällen (siehe Kapitel 4.4) auch nicht entsprechen darf, wird mitberücksichtigt. Ferner wird bei jeder Elliptischen Kurve nur ein Nachrichtenblock ver- und entschlüsselt, d.h. bei einer Kurve E definiert über einen kleinen Endlichen Körper $GF(p)$ ist ein Nachrichtenblock erheblich kleiner als bei einer Kurve E definiert über einen großen Endlichen Körper $GF(p)$. Anhand der Abbildung 6.1 kann man sehen, in welchen

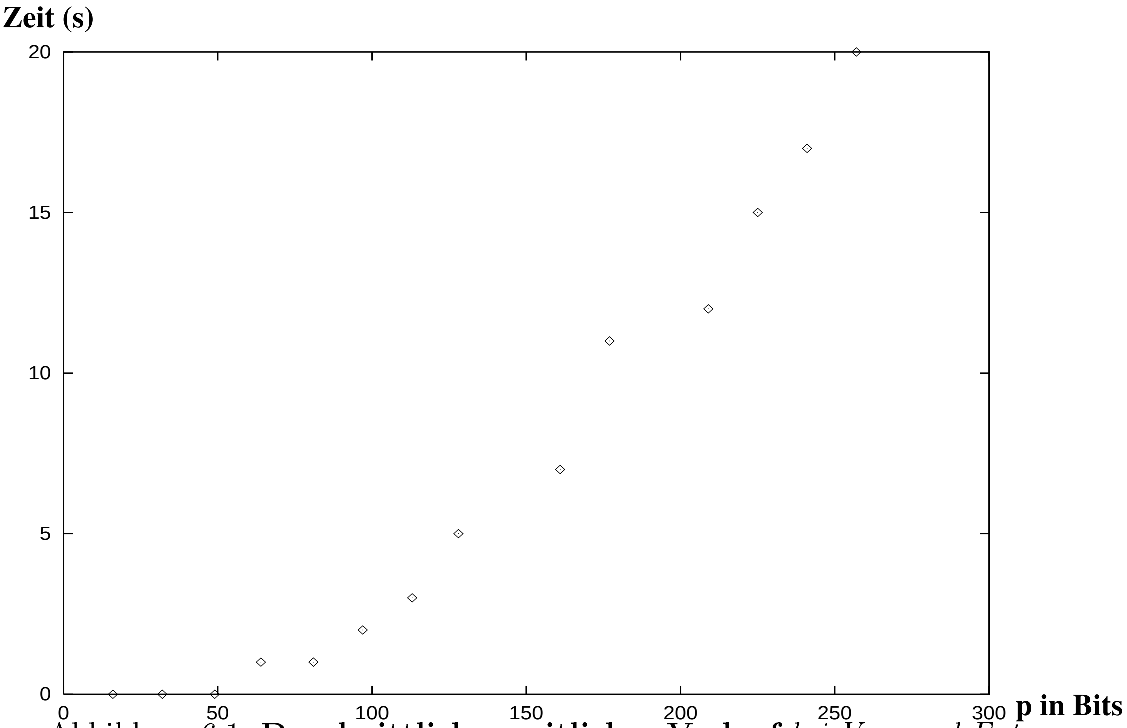

Abbildung 6.1: **Durchnittlicher zeitlicher Verlauf** *bei Ver- und Entschlüsselung eines Nachrichtenblocks bei zufälliger Wahl der Schlüssel und der Nachricht*

zeitlichen Dimensionen man sich bei diesem Kryptoverfahren befindet. Der Verlauf entspricht in etwa einer Parabel, so daß eine Verdoppelung der zugrundeliegenden Primzahl $p = 2p + x$ (p muß immer eine Primzahl sein) eine quadratische Zeitzunahme zur Folge hat.
Die zu dieser Analyse gewählten Voraussetzungen ermöglicht aber nur einen sehr ungenauen zeitlichen Vergleich, weil die Länge der Nachrichtenblöcke, die maximal übertragen werden können, in der vorherigen Betrachtung nicht berücksichtigt werden. Deshalb wird für die nächste Betrachtung folgende Nachricht der Länge 235 Byte (inklusive aller Sonderzeichen wie carriage return) mit den unterschiedlichen Kurven ver- und entschlüsselt (Umlaute sind in der TeX-Schreibweise dargestellt, werden aber nur als ein Byte angesehen):

`Dieses ist ein beliebiger Text zum Verschl"usseln`	49 Byte
`eines kleinen Textes f"ur Elliptische Kurven:`	45 Byte
`1234567890!"$%&/()="s\?`	24 Byte
`qwertzuiop"u+#QWERTZUIOP"U*'`	27 Byte
`asdfghjkl"o"aASDFGHJKL"O"A`	23 Byte
`yxcvbnm,.-YXCVBNM;:_<>`	23 Byte
`Damit sollten viele Zeichen abgedeckt sein.`	44 Byte
	235 Byte

	E_1	E_2	E_3	E_4	E_5	E_5	E_7	E_8
Schlüssel e_p	0,02 s	0,03 s	0,05 s	0,06 s	0,08 s	0,09 s	0,11 s	0,12 s
Schlüssel s_p	0,03 s	0,06 s	0,09 s	0,11 s	0,14 s	0,17 s	0,21 s	0,22 s
Verschlüsselung	0,31 s	0,25 s	0,27 s	0,26 s	0,26 s	0,30 s	0,29 s	0,31 s
Entschlüsselung	1,94 s	1,47 s	1,39 s	1,27 s	1,34 s	1,34 s	1,25 s	1,31 s
Gesamt	2,25 s	1,72 s	1,68 s	1,53 s	1,60 s	1,64 s	1,54 s	1,62 s

	E_9	E_{10}	E_{11}	E_{12}	E_{13}	E_{14}	E_{15}	E_{16}
Schlüssel e_p	0,15 s	0,15 s	0,18 s	0,21 s	0,23 s	0,23 s	0,27 s	0,29 s
Schlüssel s_p	0,26 s	0,30 s	0,36 s	0,37 s	0,41 s	0,45 s	0,50 s	0,52 s
Verschlüsselung	0,31 s	0,36 s	0,37 s	0,38 s	0,42 s	0,41 s	0,45 s	0,45 s
Entschlüsselung	1,34 s	1,47 s	1,42 s	1,27 s	1,45 s	1,56 s	1,35 s	1,48 s
Gesamt	1,65 s	1,83 s	1,79 s	1,65 s	1,87 s	1,97 s	1,80 s	1,93 s

Tabelle 6.1: **Zeitliche Veränderungen** *bei der Wahl der Elliptischen Kurve E bei Verwendung der gleichen Schlüssel und der gleichen Nachricht M, die 235 Byte lang ist.*

Um eine möglichst vergleichbare Ver- und Entschlüsselungszeit zu erhalten, werden für alle Elliptischen Kurven die gleichen geheimen Schlüssel von Sender und Empfänger verwendet. Die Auswirkungen der Schlüsselwahl auf die Ver- und Entschlüsselungszeit wird in Kapitel 6.2 untersucht. Für die hier durchgeführte Untersuchung werden der geheime Schlüssel des Empfängers e_s und der geheime Schlüssel des Senders s_s folgendermaßen festgelegt:

$$e_s = 23451 \qquad s_s = 30983$$

Die willkürliche Wahl dieser Schlüssel ist durch die Ordnung $\#E = 33247$ der kleinsten untersuchten Elliptischen Kurve beschränkt. Entscheidend für den dazugehörigen öffentlichen Schlüssel ist der Basispunkt. Auch dieser sollte daher bei allen hier betrachteten Elliptischen Kurven gleich sein. Dieses ist aber aufgrund der verschiedenen Parameter a und b der Elliptischen Kurven und der unterschiedlichen Endlichen Körper $GF(p)$, auf denen sie definiert sind, nicht realisierbar. Als Startwert der x-Koordinate soll die 1 dienen. Das Programm berechnet dann den Wert $w = x^3 + ax + b$ und überprüft dann, ob es sich dabei um einen quadratischen Rest handelt. Ist dieses der Fall, dann wird aus dem Wert w die Wurzel gezogen und der Basispunkt B lautet $B = (x, \sqrt{w})$. Handelt es sich nicht um einen quadratischen Rest, wird x solange erhöht, bis $w = x^3 + ax + b$ ein quadratischer Rest ist und der Basispunkt B lautet dann $B = (x + z \cdot 1, \sqrt{w})$, wobei z die Anzahl der notwendigen Erhöhungen ist. Die folgende Tabelle enthält die benötigten Zeiten in Sekunden aller 16 Elliptischen Kurven E_1 bis E_{16} aus dem Simulationsprogramm (siehe Anhang B). Die Gesamtzeit ergibt sich aus der CPU-Rechenzeit für die Ver- und Entschlüsselung. Alle Rechenzeiten sind in Tabelle 6.1 zusammengefaßt. Auffällig in Tabelle 6.1 ist zum einen, daß die Berechnung des öffentlichen Schlüssels des Senders s_p ungefähr doppelt so viel Zeit in Anspruch nimmt wie die Berechnung des öffentlichen Schlüssels e_p des Empfängers. Das liegt hauptsächlich daran, daß vor der Verwendung von s_s zu überprüfen ist, ob:

$$s_s \cdot e_p = s_p = (x_{s_p}, y_{s_p}) \qquad \text{mit } x_{s_p} \neq 0 \tag{6.1}$$

gilt. Nur wenn genau dieses der Fall ist, kann überhaupt eine Verschlüsselung erfolgen. Nach einer erfolgreichen Überprüfung wird dann der Punkt:

$$s_p = (x_{s_p}, y_{s_p}) = s_s \cdot e_p$$

berechnet. Insgesamt werden also zwei Multiplikationen mit einem Punkt der Elliptischen Kurve mit derselben näturlichen Zahl s_s ausgeführt. Da die Differenz zwischen den festgelegten geheimen Schlüsseln nicht sehr groß sind, für die Bitlängen der geheimen Schlüssel s_s und e_s gilt:

$$\lceil log_2 s_s \rceil = \lceil log_2 30983 \rceil = 15$$
$$\lceil log_2 e_s \rceil = \lceil log_2 23451 \rceil = 15$$

umfaßt die Multiplikation mit dem Basispunkt B annähernd die gleiche Anzahl von Operationen.
Ferner läßt sich aus der Tabelle 6.1 feststellen, daß die Gesamtzeit der Ver- und Entschlüsselung bei den unterschiedlichen Elliptischen Kurven kaum variiert. Das kommt daher, daß die Berechnung des Punktes $(x_2, y_2) = s_s e_p$, der zur Verschlüsselung der Nachricht M notwendig ist, bei jeder Kurve fast den gleichen Rechenaufwand hat. Die einzigen Unterschiede liegen bei dem Parameter a der Elliptischen Kurve, der nach der Gleichung (2.8) direkt in die Multiplikation eines Punktes der Elliptischen Kurve mit einer natürlichen Zahl eingeht. Alle anderen Operationen der Ver- und Entschlüsselung umfassen nur einfache Bitoperationen, die zwar bei großen Zahlen rechenintensiver sind als bei kleinen, aber nicht so sehr ins Gewicht fallen, weil dieser Zeitverlust durch die Einsparung von Verschlüsselungsschritten kompensiert wird.
Um aber noch genauere Aussagen treffen zu können, sind noch weitere Untersuchungen notwendig, insbesondere interessieren zeitliche Abhängigkeiten durch Wahl der Schlüssel und der Nachricht, die in den folgenden Abschnitten näher analysiert werden sollen.

6.2 Rechenzeiten in Abhängigkeit der Schlüssellänge

Interessant für potentielle Angreifer ist der Zusammenhang zwischen der Schlüssellänge und der daraus resultierenden benötigten Zeit der Ver- und Entschlüsselung. Um die Auswirkungen der einzelnen Schlüssel auf die Gesamtrechenzeit der Ver- und Entschlüsselung zu analysieren, werden hier unter Beibehaltung aller Parameter zwei unterschiedliche Vorgehensweisen untersucht:

1. fester geheimer Schlüssel e_s des Empfängers bei variablen geheimen Schlüssel s_s des Senders (siehe Tabelle 6.2)

2. fester geheimer Schlüssel s_s des Senders bei variablen geheimen Schlüssel e_s des Empfängers (siehe Tabelle 6.3)

Beide Fälle werden mit den Elliptischen Kurven E_1, der kleinsten, E_8, einer mittleren, und E_{16}, der größten, aus dem Anhang B durchgeführt. Dieses führt im ersten Fall, bei

Schlüssel s_s	3	$3 \cdot 2^2$	$3 \cdot 2^4$	$3 \cdot 2^6$	$3 \cdot 2^8$	$3 \cdot 2^{10}$	$3 \cdot 2^{12}$	$3 \cdot 2^{13}$
Schlüssel e_p	0,02 s	0.02 s	0,02 s	0,02 s	0,02 s	0.02 s	0.02 s	0,02 s
Schlüssel s_p	0,00 s	0.01 s	0,01 s	0,02 s	0,02 s	0,03 s	0,02 s	0,03 s
Verschlüsselung	0,27 s	0.31 s	0,27 s	0,28 s	0,28 s	0,30 s	0,30 s	0,32 s
Entschlüsselung	1,93 s	1.94 s	2,04 s	2,02 s	2,03 s	1,97 s	2,05 s	2,02 s
Gesamt	2,20 s	2,26 s	2,32 s	2,32 s	2,33 s	2,30 s	2,37 s	2,37 s

Schlüssel s_s	3	$3 \cdot 2^{16}$	$3 \cdot 2^{32}$	$3 \cdot 2^{48}$	$3 \cdot 2^{64}$	$3 \cdot 2^{80}$	$3 \cdot 2^{96}$	$3 \cdot 2^{112}$	$3 \cdot 2^{124}$
Schlüssel e_p	0,12 s	0,12 s	0,12 s	0,12 s	0,12 s	0,12 s	0,12 s	0,12 s	0,12 s
Schlüssel s_p	0,02 s	0,21 s	0,40 s	0,59 s	0,80 s	1,03 s	1,24 s	1,39 s	1,49 s
Verschlüsselung	0,19 s	0,28 s	0,38 s	0,50 s	0,55 s	0,72 s	0,80 s	0,88 s	0,93 s
Entschlüsselung	1,34 s	1,35 s	1,35 s	1,35 s	1,30 s	1,36 s	1,40 s	1,33 s	1,36 s
Gesamt	1,55 s	1,84 s	1,77 s	2,44 s	2,65 s	3,11 s	3,44 s	3,60 s	3,78 s

Schlüssel s_s	3	$3 \cdot 2^{32}$	$3 \cdot 2^{64}$	$3 \cdot 2^{96}$	$3 \cdot 2^{128}$	$3 \cdot 2^{160}$	$3 \cdot 2^{192}$	$3 \cdot 2^{224}$	$3 \cdot 2^{254}$
Schlüssel e_p	0,30 s	0,30 s	0,30 s	0,30 s	0,30 s	0,30 s	0,30 s	0,30 s	0,30 s
Schlüssel s_p	0,05 s	0,90 s	1,82 s	2,80 s	3,47 s	4,28 s	5,10 s	5,94 s	6,74 s
Verschlüsselung	0,22 s	0,61 s	1,08 s	1,58 s	1,90 s	2,29 s	2,73 s	3,13 s	3,56 s
Entschlüsselung	1,53 s	1,55 s	1,54 s	1,52 s	1,49 s	1,51 s	1,58 s	1,57 s	1,55 s
Gesamt	1,80 s	3,06 s	4,44 s	5,90 s	6,86 s	8,08 s	9,41 s	10,64 s	11,85 s

Tabelle 6.2: **Rechenzeiten** *bei Änderung des geheimen Schlüssels s_s des Senders über E_1 mit $p > 2^{15}$ (oben), E_8 mit $p > 2^{127}$ (mitte) und E_{16} mit $p > 2^{256}$ (unten).*

dem der Schlüssel des Empfängers für alle drei Elliptischen Kurven $e_s = 23451$ und damit $e_p = e_s \cdot B = 23451 \cdot B$ ist, zu den Ergebnissen in Tabelle 6.2. Wie erwartet, ist die Berechnungszeit in der Tabelle 6.2 für den öffentlichen Schlüssel e_p bei jeder Kurve konstant. Die Berechnung des öffentlichen Schlüssels ist ja $e_p = e_s \cdot B$ und damit unabhängig von den Schlüsseln des Senders. Die Berechnungszeit ist aber natürlich abhängig von der Wahl der Elliptischen Kurve, denn die Operationen von $GF(p)$ hängen von der Primzahl p ab. Anders verhält es sich bei der Berechnung des öffentlichen Schlüssels s_p des Senders. Dieser ist von dem geänderten geheimen Schlüssel direkt abhängig. Nach Kapitel 5.2.2 ergibt sich ein logarithmischer Aufwand, was sich in der gemessenen Rechenzeit der Tabelle 6.2 widerspiegelt. Außerdem geht mit der Zunahme der Schlüssellänge ebenfalls eine zeitliche Zunahme der Verschlüsselung einher. Das kommt daher, daß die Punkte $(x_1, y_1) = s_s \cdot B$ und $(x_2, y_2) = s_s \cdot e_p$ berechnet werden müssen. Wird s_s größer, so nehmen auch diese Berechnungen ungefähr logarithmisch zu, was die hauptsächliche Zunahme der Zeit bei der Verschlüsselung ausmacht. Alle anderen Operationen der Verschlüsselung sind vernachlässigbar. Im Gegensatz zu der Verschlüsselung bleibt die Entschlüsselung bei der jeweiligen Elliptischen Kurve nach Tabelle 6.2 fast konstant. Das ist dadurch zu begründen, daß sich der geheime Schlüssel des Empfängers nicht geändert hat. Der Empfänger muß zur Entschlüsselung den Punkt $(x_2, y_2) = e_s \cdot (x_1, y_1)$ berechnen, somit ergeben sich für die Mutiplikation nach Kapitel 5.2.2 keine Änderungen, weil sich der Faktor e_s nicht (und damit auch nicht das abzuarbeitende Array aus Kapitel 5.2.2) geändert hat. Das einzige, was sich mit Sicherheit bei der Berechnung verändert hat, ist der Punkt (x_1, y_1). Damit kann sich

Schlüssel e_s	3	$3 \cdot 2^2$	$3 \cdot 2^4$	$3 \cdot 2^6$	$3 \cdot 2^8$	$3 \cdot 2^{10}$	$3 \cdot 2^{12}$	$3 \cdot 2^{13}$
Schlüssel s_p	0,04 s	0,05 s	0,04 s	0,05 s	0,04 s	0,04 s	0,06 s	0,04 s
Schlüssel e_p	0,00 s	0,00 s	0,00 s	0,00 s	0,01 s	0,01 s	0,01 s	0,01 s
Verschlüsselung	0,28 s	0,28 s	0,30 s	0,28 s	0,32 s	0,29 s	0,30 s	0,32 s
Entschlüsselung	0,48 s	0,62 s	0,75 s	0,89 s	1,09 s	1,24 s	1,38 s	1,40 s
Gesamt	0,76 s	0,90 s	1,05 s	1,17 s	1,42 s	1,54 s	1,69 s	1,73 s

Schlüssel e_s	3	$3 \cdot 2^{16}$	$3 \cdot 2^{32}$	$3 \cdot 2^{48}$	$3 \cdot 2^{64}$	$3 \cdot 2^{80}$	$3 \cdot 2^{96}$	$3 \cdot 2^{112}$	$3 \cdot 2^{124}$
Schlüssel s_p	0,26 s	0,26 s	0,26 s	0,27 s	0,26 s	0,26 s	0,27 s	0,26 s	0,27 s
Schlüssel e_p	0,01 s	0,11 s	0,20 s	0,29 s	0,40 s	0,50 s	0,59 s	0,68 s	0,72 s
Verschlüsselung	0,32 s	0,29 s	0,33 s	0,32 s	0,32 s	0,31 s	0,35 s	0,32 s	0,30 s
Entschlüsselung	0,39 s	1,14 s	1,86 s	2,65 s	3,41 s	4,26 s	5,07 s	5,52 s	6,07 s
Gesamt	0,72 s	1,54 s	2,39 s	3,26 s	4,13 s	5,07 s	6,01 s	6,52 s	7,09 s

Schlüssel e_s	3	$3 \cdot 2^{32}$	$3 \cdot 2^{64}$	$3 \cdot 2^{96}$	$3 \cdot 2^{128}$	$3 \cdot 2^{160}$	$3 \cdot 2^{192}$	$3 \cdot 2^{224}$	$3 \cdot 2^{254}$
Schlüssel s_p	0,60 s	0,59 s	0,60 s	0,59 s	0,59 s	0,59 s	0,59 s	0,60 s	0,59 s
Schlüssel e_p	0,02 s	0,46 s	0,87 s	1,30 s	1,68 s	2,09 s	2,49 s	2,89 s	3,28 s
Verschlüsselung	0,46 s	0,47 s	0,48 s	0,48 s	0,48 s	0,48 s	0,49 s	0,49 s	0,51 s
Entschlüsselung	0,40 s	2,02 s	3,70 s	5,47 s	6,92 s	8,54 s	10,16 s	11,78 s	13,34 s
Gesamt	0,88 s	2,95 s	5,05 s	7,25 s	9,08 s	11,11 s	13,14 s	15,16 s	17,13 s

Tabelle 6.3: **Rechenzeiten** *bei Änderung des geheimen Schlüssels e_s des Empfängers über E_1 mit $p > 2^{15}$ (oben), E_8 mit $p > 2^{127}$ (mitte) und E_{16} mit $p > 2^{256}$ (unten).*

aber höchstens die Anzahl der notwendigen Reduktionen erhöhen. Bei einer Punktverdoppelung sind das im Simulationsprogramm zehn, bei einer Addition zweier verschiedener Punkte acht Reduktionen. Für die gesamte Multiplikation sind das dann im ungünstigsten Fall $9 \cdot (2 \log_2 s_s)$ Reduktionen. Praktisch wird man aber niemals eine Multiplikation auf einer Elliptischen Kurve haben, bei der keine Reduktionen notwendig sind. Letztendlich resultieren die Schwankungen um einen zeitlichen Mittelwert also von der Anzahl der durchgeführten Reduktionen.
Variiert man den geheimen Schlüssel e_s des Empfängers und behält den geheimen Schlüssel s_s des Senders bei (Fall 2), führt das für dieselben Elliptischen Kurven zu den Ergebnissen in Tabelle 6.3. Wie zu erwarten, ist jetzt für die jeweilige Elliptische Kurve die Berechnung des öffentlichen Schlüssels $s_p = s_s \cdot B$ des Senders nach der Tabelle 6.3 konstant, es werden ja auch keine Veränderungen am geheimen Schlüssel s_s des Senders und am Basispunkt B durchgeführt. Nach den Ausführung von Kapitel 6.1 darf man nun aufgrund der notwendigen Kontrolle nach Gleichung (6.1) die doppelte Zeit erwarten als es im Fall 1 für e_p der Fall war. Annähernd wird diese Vermutung durch die gemessene Zeit bestätigt. Anderenfalls kann man vermuten, daß nun die Berechnung des öffentlichen Schlüssels des Empfängers nur halb so lang in Anspruch nimmt wie zuvor im Fall 1, weil hier die gleichen Werte verwendet worden sind und keine Überprüfung durchgeführt werden muß. Wie man den Ergebnistabellen 6.3 entnehmen kann, wird diese Vermutung auch durch die Messergebnisse bestätigt. Die Verschlüsselung ist nur abhängig vom geheimen Schlüssel s_s des Senders, der bei dieser Untersuchung fest ist. Daher ist die Zeit annähernd für die jewei-

Schlüssel e_s	3	$3 \cdot 2^{2}$	$3 \cdot 2^{4}$	$3 \cdot 2^{6}$	$3 \cdot 2^{8}$	$3 \cdot 2^{10}$	$3 \cdot 2^{12}$	$3 \cdot 2^{13}$
Schlüssel s_s	$3 \cdot 2^{13}$	$3 \cdot 2^{11}$	$3 \cdot 2^{9}$	$3 \cdot 2^{7}$	$3 \cdot 2^{5}$	$3 \cdot 2^{3}$	$3 \cdot 2^{1}$	3
Schlüssel e_p	0,00 s	0,00 s	0,01 s	0,01 s	0,01 s	0,01 s	0,02 s	0,01 s
Schlüssel s_p	0,02 s	0,03 s	0,02 s	0,01 s	0,01 s	0,01 s	0,00 s	0,01 s
Verschlüsselung	0,32 s	0,32 s	0,32 s	0,31 s	0,29 s	0,33 s	0,29 s	0,32 s
Entschlüsselung	0,52 s	0,66 s	0,81 s	0,93 s	1,05 s	1,22 s	1,37 s	1,41 s
Gesamt	0,86 s	1,01 s	1,16 s	1,26 s	1,36 s	1,57 s	1,68 s	1,75 s

Schlüssel e_s	3	$3 \cdot 2^{16}$	$3 \cdot 2^{32}$	$3 \cdot 2^{48}$	$3 \cdot 2^{64}$	$3 \cdot 2^{80}$	$3 \cdot 2^{96}$	$3 \cdot 2^{112}$	$3 \cdot 2^{124}$
Schlüssel s_s	$3 \cdot 2^{124}$	$3 \cdot 2^{108}$	$3 \cdot 2^{92}$	$3 \cdot 2^{76}$	$3 \cdot 2^{59}$	$3 \cdot 2^{43}$	$3 \cdot 2^{28}$	$3 \cdot 2^{12}$	3
Schlüssel e_p	0,00 s	0,08 s	0,19 s	0,29 s	0,39 s	0,49 s	0,56 s	0,64 s	0,71 s
Schlüssel s_p	1,42 s	1,24 s	1,08 s	0,92 s	0,70 s	0,51 s	0,33 s	0,16 s	0,02 s
Verschlüsselung	0,92 s	0,83 s	0,76 s	0,67 s	0,57 s	0,45 s	0,34 s	0,28 s	0,22 s
Entschlüsselung	0,37 s	1,12 s	1,88 s	2,60 s	3,43 s	4,19 s	4,85 s	5,44 s	5,93 s
Gesamt	2,71 s	3,27 s	3,91 s	4,48 s	5,09 s	5,64 s	6,08 s	6,52 s	6,88 s

Schlüssel e_s	3	$3 \cdot 2^{32}$	$3 \cdot 2^{64}$	$3 \cdot 2^{96}$	$3 \cdot 2^{128}$	$3 \cdot 2^{160}$	$3 \cdot 2^{192}$	$3 \cdot 2^{224}$	$3 \cdot 2^{254}$
Schlüssel s_s	$3 \cdot 2^{254}$	$3 \cdot 2^{222}$	$3 \cdot 2^{190}$	$3 \cdot 2^{158}$	$3 \cdot 2^{126}$	$3 \cdot 2^{94}$	$3 \cdot 2^{62}$	$3 \cdot 2^{30}$	3
Schlüssel e_p	0,02 s	0,44 s	0,91 s	1,33 s	1,71 s	2,13 s	2,54 s	2.97 s	3,34 s
Schlüssel s_p	6,66 s	5,86 s	5,01 s	4,20 s	3,36 s	2,59 s	1,70 s	0,83 s	0,06 s
Verschlüsselung	3,54 s	3,12 s	2,69 s	2,30 s	1,87 s	1,49 s	1,07 s	0,63 s	0,24 s
Entschlüsselung	0,44 s	2,09 s	3,87 s	5,53 s	7,11 s	8,82 s	10,40 s	12,12 s	13,63 s
Gesamt	10,66 s	11,51 s	12,48 s	13,36 s	14,05 s	15,03 s	15,71 s	16,55 s	17,27 s

Tabelle 6.4: **Rechenzeiten** *bei Änderungen beider geheimer Schlüssel e_s und s_s über E_1 mit $p > 2^{15}$ (oben), E_8 mit $p > 2^{127}$ (mitte) und E_{16} mit $p > 2^{256}$ (unten)*

lige Elliptische Kurve konstant. Die Schwankungen resultieren hier aus der unterschiedlichen Anzahl von Reduktionen, die bei der Berechnung von $(x_2, y_2) = s_s \cdot e_p$ durchgeführt werden müssen. Für die Entschlüsselung gilt: je größer der geheime Schlüssel e_s, desto länger dauert sie. Auch hier liegt der Hauptgrund wieder in der Berechnung des Punktes $(x_2, y_2) = e_s \cdot (x_1, y_1)$. Bleibt nur noch zu klären, warum die Entschlüsselung hier viel mehr Zeit in Anspruch nimmt als im Fall 1 die Verschlüsselung. Das liegt eindeutig daran, daß bei der Entschlüsselung nach Kapitel 4.3 das Inverse berechnet werden muß.
Aus den vorliegenden Untersuchung kann man den Schluß ziehen, daß es von der Rechenzeit günstiger ist, eher einen großen geheimen Schlüssel s_s zu wählen und als öffentlichen Schlüssel einen möglichst kleinen Multiplikator vom Basispunkt B zu haben. Es bietet sich daher an, die zeitliche Abhängigkeit des Verfahrens aufgrund von gleichzeitiger Veränderungen der Schlüssel zu untersuchen. Dabei wird der geheime Schlüssel e_s, ausgehend von $e_s = 3$, des Empfängers und der geheime Schlüssel s_s, ausgehend von $s_s = 3 \cdot 2^{höchstePotenz}$, nach durchgeführter Ver- und Entschlüsselung mit einem Faktor multipliziert bzw. durch einen Faktor dividiert, so daß letztendlich die Schlüssellängen gegeneinander laufen. Die zugehörigen Rechenzeiten der drei Kurven E_1, E_8 und E_{16} aus dem Anhang B sind der Tabelle 6.4 zu entnehmen. Anhand der Tabelle 6.4 kann man sehen, daß die Veränderungen des geheimen Schlüssels s_s des Senders keine Auswirkungen auf die Berechnung des öffentlichen Schlüssels e_p des Empfängers haben. Die Rechenzeiten sind praktisch identisch mit

	Nachricht 1	Nachricht 2	Nachricht 3
Schlüssel e_p	$0,54$ s	$0,54$ s	$0,54$ s
Schlüssel s_p	$1,08$ s	$1,07$ s	$1,08$ s
Verschlüsselung	$0,56$ s	$0,57$ s	$0,56$ s
Entschlüsselung	$0,62$ s	$0,61$ s	$0,61$ s
Gesamt	$2,80$ s	$2,79$ s	$2,79$ s

Tabelle 6.5: Rechenzeiten in Abhängigkeit der Nachricht M über E_{16} mit $p > 2^{256}$

den Rechenzeiten der Tabelle 6.3. Die Unabhängigkeit ist nach der Gleichung $e_p = e_s \cdot B$ auch unmittelbar einzusehen. Vergleicht man die Rechenzeiten des öffentlichen Schlüssels s_p der Tabelle 6.4 mit denen der Tabelle 6.2, so kann man feststellen, daß auch hier die Berechnung von s_p nahezu unabhängig von den Schlüsseln des Empfängers ist. Anhand der Gleichung $s_p = s_s \cdot B$ ist die Berechnung unabhängig, doch muß die Kontrolle nach Gleichung (6.1) durchgeführt werden, die abhängig von dem öffentlichen Schlüssel e_p ist. Die Veränderungen von e_p fallen aber trotzdem nicht ins Gewicht, weil der Rechenaufwand bei $s_s \cdot e_p$ nach Kapitel 5.2.2 durch den Skalar s_s verursacht wird. Mit der gleichen Begründung sind auch bei der Verschlüsselung, bei der die Berechnung von $(x_2, y_2) = s_s \cdot e_p$ hauptsächlich die Rechenzeit in Anspruch nimmt, kaum zeitliche Differenzen zu der vorherigen Untersuchung zu erwarten. Ein Vergleich der Zeilen der Tabellen 6.2 und 6.4 bestätigt die Erwartung. Entsprechendes gilt auch für die Entschlüsselung. Demnach ergibt sich die Gesamtrechenzeit aus der Summe der Rechenzeit für die Bestimmung des öffentlichen Schlüssels s_p des Senders mit der dazugehörigen Rechenzeit der Verschlüsselung und der Rechenzeit des öffentlichen Schlüssels e_p mit dazugehöriger Entschlüsselung.

6.3 Rechenzeit in Abhängigkeit von der Nachricht

Als letzter Parameter bleibt noch die Nachricht M zu betrachten. Nach den Kapiteln 4.2.1 und 4.3 sind von der zu ver- bzw. entschlüsselnden Nachricht M nur die XOR-Verknüpfung und eine Multiplikation abhängig. Im Vergleich zu einer Multiplikation eines Punktes mit einem Skalar sind diese beiden Operationen vernachlässigbar. Es ist also zu vermuten, daß die Nachricht M keinen großen Einfluß auf die Rechenzeiten hat. Dieses vermutete Ergebnis veranschaulicht die Tabelle 6.5. Als geheimer Schlüssel sowohl des Empfängers als auch des Senders diente hier über die Elliptische Kurve E_{16} die Zahl 1234567890. Die drei zu ver- und entschlüsselnden Nachrichten hatten maximale Länge. Insgesamt wurden folgende drei Nachrichten in Binärdarstellung ver- und entschlüsselt:

- Nachricht 1: $2^{256} = 100000\ldots$
- Nachricht 2: $2^{256} - 1 = 111111\ldots$
- Nachricht 3: alternierend 101010101 ...

Die Meßergebnisse aus Tabelle 6.5 bestätigen die Vermutungen. Den größten Einfluß hat die Nachricht bei der Verschlüsselung auf die Rechenzeit, wenn die bitweisen XOR-Verknüpfungen $z_1 = m_1 \oplus y_3$ und $z_2 = m_2 \oplus x_3$ (siehe Kapitel 4.2.1) die Ergebnisse $z_1 = 11111\ldots_2$ und $z_2 = 11111\ldots_2$ zur Folge hätten. Dann werden die abschließenden Multiplikation $c_1 = x_4 * z_1$ und $c_2 = y_4 * z_2$ mit maximalem Multiplikanten durchgeführt. Eine Betrachtung der Entschlüsselung führte zu dergleichen Schlußfolgerung.

6.4 Hardwareabschätzung

Ein Chip, der das in dieser Arbeit analysierte Kryptoverfahren realisiert, muß über einen Permanentspeicher, etwa einem EEPROM (**E**lectrical **E**rasable **P**rogrammable **R**ead **O**nly **M**emory), einen Zwischenspeicher, auch als RAM (**R**andom **A**ccess **M**emory) bezeichnet, ein Steuerwerk und ein Operationswerk verfügen. Im Permanentspeicher werden die Systemparameter abgespeichert. Diese bestehen aus der Primzahl p, den Parametern a und b der Elliptischen Kurve $E : y^2 = x^3 + ax + b$, den Koordinaten x_b und y_b des Basispunktes B und dessen Ordnung $\#B = n$. Bis auf n können alle Parameter in Speichereinheiten der Länge $\log_2 p$, die im folgenden als Einheitslänge angenommen wird, gespeichert werden. Für n ergibt sich nach dem Hasse Theorem (siehe Gleichung (2.9)) eine maximale Länge von $\log_2(p + 1 + 2\sqrt{p})$. Also kann $\#B = n$ in zwei Speichereinheiten abgelegt werden. Die Systemparameter belegen insgesamt also sieben Speichereinheiten: p (1), a (1), b (1), x_b (1), y_b (1) und $\#B$ (2) (in Klammern steht die Anzahl der benötigten Speichereinheiten). Zusätzlich zu den Systemparametern müssen die Schlüssel der Ver- oder Entschlüsselung gespeichert werden, also der geheime und der öffentliche Schlüssel. Im Falle einer Verschlüsselung sind dieses die Schlüssel s_s und e_p und im Falle einer Entschlüsselung die Schlüssel e_s und der übertragende Punkt (x_1, y_1). Insgesamt werden für die Verwaltung der Schlüssel drei Speichereinheiten (einen für den geheimen und zwei für den öffentlichen) verwendet.

Das Steuerwerk besteht aus einem Mikroprogramm, daß die Ansteuerung der Speichereinheiten und die Funktionsweise des Operationswerks übernimmt. Das Steuerwerk hängt sehr stark von den verwendeten Speicherbausteinen und deren Timing ab und ist deshalb nicht Gegenstand der hier durchzuführenden Hardwareabschätzung.

Das Operationswerk berechnet mit Hilfe einer ALU (**A**rithmetic **L**ogical **U**nit) die Ergebnisse der Ver- bzw. Entschlüsselung. Für die Hardwareabschätzung soll die Minimal-ALU folgende Funktionen umfassen:

> Addieren, Subtrahieren, Dekrementieren, Inkrementieren, um ein Bit nach links und rechts Schieben, Vergleichen auf 0 und 1 und die boolschen bitweisen Verknüpfungen AND, OR und XOR.

Für die Umsetzung der Ver- und Entschlüsselung sind einige Operationen von $GF(p)$ und spezielle Funktionen über Elliptische Kurven notwendig. Diese müssen mit den oben erwähnten Funktionen der ALU realisiert werden. Da diese Operationen erheblich umfangreicher als die Grundoperationen der ALU sind, müssen Speichereinheiten zum Speichern von Zwischenergebnissen reserviert werden. Die nun folgende Untersuchung wird eine untere Grenze dieser benötigten zusätzlichen Speichereinheiten liefern. Dazu werden alle Operationen betrachtet und deren zusätzlichen Speichereinheitenbedarf analysiert. Die benötigten Speichereinheiten geben den maximalen Speicherbedarf an. Nach Berechnung des gewünschten Ergebnisses können alle Speichereinheiten bis auf die das Ergebnis enthaltende anderen Operationen zur Verfügung gestellt werden. Für die Operationen über $GF(p)$ ergeben sich folgende Anforderungen:

1. Modulo-Reduktion:
   Die Modulo-Reduktion kann auf die Addition bzw. Subtraktion der ALU zurückgeführt werden. Ist eine Zahl z größer als die Primzahl p, dann wird von z solange p subtrahiert, bis $z < p$ ist. Diese Bedingung läßt sich mit Hilfe der Vergleichsfunktion der ALU realisieren. Ist $z < 0$, dann wird zu z solange p addiert, bis $z > 0$ ist. Diese Bedingung läßt sich ebenfalls mit der Vergleichsfunktion der ALU realisieren. Jegliche Zwischenergebnisse können in der Speichereinheit der Zahl z gespeichert werden, so daß insgesamt keine weiteren Speichereinheiten für die Modulo-Reduktion notwendig sind.

2. Multiplizieren:
   Die Multiplikation läßt sich auf Addition und Schieben zurückführen. In der Binärdarstellung wird der Faktor vom MSB zum LSB abgearbeitet. Ist das gelesene Bit $= 0$, dann wird das Zwischenergebnis um eine Stelle nach links verschoben. Ist das gelesene Bit $= 1$ wird der Multiplikant zum Zwischenergebnis addiert. Für die Speicherung des Zwischenergebnis wird eine Speichereinheit benötigt, so daß insgesamt eine Speichereinheit notwendig ist, um eine Multiplikation mit den Funktionen der ALU umzusetzen.

3. Potenzieren:
   Nach dem Algorithmus 5.1.1 benötigt man drei Variablen, nämlich N, y und z. Damit werden drei Speichereinheiten notwendig. Zusätzlich müssen noch eine Multiplikation ausgeführt werden, die noch eine zusätzliche Speichereinheit in Anspruch nimmt. Die zusätzliche Funktion $\lfloor N/2 \rfloor$ entspricht einer Rechtsverschiebung um 1 Bit, so daß hierfür keine weiteren Speichereinheiten fällig werden. Der abschließende Vergleich gehört zu den elementaren Funktionen der ALU. Insgesamt werden für das Potenzieren also noch vier zusätzliche Speichereinheiten benötigt.

4. Bestimmung des Inversen:
   Für die Bestimmung des Inversen sind nach dem Algorithmus 5.1.2 sieben Variablen notwendig. Für jede muß eine Speichereinheit zur Verfügung gestellt werden. Eine weitere ergibt sich durch eine Multiplikation. Sowohl die Modulo-Reduktion als auch die Div-Funktion, die durch Schieben um 1 Bit nach rechts realisiert werden kann, kommen ohne zusätzliche Speichereinheiten aus, so daß insgesamt für die Berechnung des Inversen acht Speichereinheiten ausreichen.

5. Dividieren:
Die Division läßt sich auf Multiplikation mit dem Inversen des Nenners zurückführen. Daher muß als erstes das Inverse des Nenners bestimmt werden, was acht Speichereinheiten zur Folge hat. Für die anschließende Multiplikation stehen dann von den acht Speichereinheiten, die für die Berechnung desInversen notwendig sind, sieben wieder frei zur Verfügung, wovon nur eine letztendlich benötigt wird. Damit genügen acht zusätzliche Speichereinheiten, um eine Division mit den Funktionen der ALU zu realisieren.

Die Wurzelberechnung und die Berechnung des Legendre Symbols sind nur für die Bestimmung eines Basispunktes, die von einem Systemverwalter vorgenommen wird, erforderlich. Beide Operationen kommen demnach weder bei der Ver- noch bei der Entschlüsselung vor. Sie spielen auch bei der Generierung des geheimen und des öffentlichen Schlüssels keine Rolle. Daher kann hier auf eine Analyse des Speicherbedarfs verzichtet werden.

Bleibt nur noch der Speicherbedarf der Funktionen der Elliptischen Kurven zu analysieren.

1. Addition zweier Punkte: (siehe Kapitel 2.3.3)
Die Addition von Unendlichkeitspunkten, die Addition eines beliebigen Punktes mit dem Unendlichkeitspunkt und die Addition von inversen Punkten bedarf keiner zusätzlichen Speichereinheiten. Diese Fälle sind mit einfachen Vergleichen abzuhandeln. Koordinaten müssen in all diesen Fällen nicht berechnet werden. Anders verhält es sich bei der Punktverdoppelung und bei der Addition von beliebigen Punkten. In beiden Fällen stellt die Berechnung von λ den größten Speicherbedarf dar, denn es muß wieder das Inverse bestimmt werden, was einen Aufwand von acht Speichereinheiten zur Folge hat. Da λ sowohl für die Berechnung der x- als auch der y-Koordinate benötigt wird, sollte λ zwischengespeichert werden. Es bleiben also für die restlichen Operationen noch sechs Speichereinheiten übrig, wenn man schon vorsorglich eine Speichereinheit für die x-Koordinate, die in die Berechnung der y-Koordinate einfließt, reserviert. Die Berechnung der x-Koordinate erfolgt bei der Punktverdoppelung mit zwei Multiplikationen und einer Subtraktion, welche ohne Probleme in den verbleibenden sechs Speichereinheiten durchzuführen sind. Bei der Addition zweier beliebiger Punkte erfolgt die Berechnung der x-Koordinate mittels einer Multiplikation und zweier Subtraktionen, welche ebenfalls ohne Probleme in den verbleibenden sechs Speichereinheiten zu lösen sind. Die Berechnung der y-Koordinate erfolgt in beiden Fällen durch eine Multiplikation und durch zwei Subtraktionen, die auch ohne Probleme in den verbleibenden sechs Speichereinheiten lösbar sind. Demnach werden insgesamt für die Addition zweier Punkte auf einer Elliptischen Kurve acht Speichereinheiten benötigt.

2. Multiplikation eines Punktes mit einem Skalar:
Als erstes muß der Skalar nach 5.2.2 zerlegt werden. Im ungünstigsten Fall kann der Speicherbedarf der Belegung fast doppelt so groß sein wie für den Skalar selbst. Daher sind für die Zerlegung, die mit einem Inkrementer und einer Rechtsverschiebung um 1 Bit umgesetzt werden kann, zwei Speichereinheiten vorzusehen. Jedes Bit

Operation	Speichereinheiten SE	SE für Ergebnis	SE frei nach Operation
Modulo-Reduktion	0	0	0
Multiplizieren	1	1	0
Potenzieren	4	1	3
Invertieren	8	1	7
Dividieren	8	1	7
Addition auf E	8	2	6
Multiplikation auf E	12	2	10

Tabelle 6.6: **Speicheranforderungen** *der Operationen von* $GF(p)$ *und* E.

der Zerlegung hat entweder eine Punktverdoppelung (Bit = 1) oder eine Addition zweier beliebiger Punkte (Bit = 0) zur Folge. Beide Operationen kommen mit acht Speichereinheiten aus. Nach jedem abgearbeiteten Bit muß der berechnete Punkt gespeichert werden, weil er der Ausgangspunkt für die nächsten Punktadditionen ist. Deshalb sind zwei weitere Speichereinheiten für die x- und y-Koordinate des berechneten Punktes notwendig. Am Ende der Berechnung steht in diesen Speichereinheiten das Ergebnis der Multiplikation eines Punktes mit einem Skalar. Insgesamt werden für diese Operation zwölf Speichereinheiten benötigt.

Die Tabelle 6.6 faßt noch einmal alle Speichereinheitenanforderungen ausgehend von einer ALU, die addieren, subtrahieren, vergleichen, schieben und bitweises AND, OR und XOR ausführen kann, pro Operation zusammen. Zusätzlich wird angegeben, wieviele Speichereinheiten nach der Berechnung wieder frei zur Verfügung gestellt werden können. Damit sind die Speicheranforderungen für alle Operationen analysiert, so daß sich jetzt eine Gesamtabschätzung von Speichereinheiten bezüglich der Ver- und Entschlüsselung durchführen läßt.

1. Verschlüsselung:
   Nach Kapitel 4.2 und 4.2.1 sind insgesamt für die Verschlüsselung zwei Multiplikationen eines Punktes mit einem Skalar, eine Potenzierung, zwei Konkatenationen, zwei XOR-Verknüpfungen und zwei Multiplikationen durchzuführen. Die aufwendigste Operation ist dabei nach Tabelle 6.6 die Multiplikation eines Punktes mit einem Skalar, die zwölf Speichereinheiten erfordert. Nach Ausführung der ersten Multiplikation mit einem Skalar stehen von den zwölf notwendigen Speichereinheiten nur noch zehn frei zur Verfügung. Zwei Speichereinheiten werden zur Speicherung der x- und y-Koordinate des berechneten Punktes verwendet. Für die zweite Multiplikation eines Punktes mit einem Skalar sind aber zwölf Speichereinheiten notwendig, so daß man, um beide Operationen ausführen zu können, zwei weitere Speichereinheiten benötigt. Insgesamt kann man die beiden Multiplikationen eines Punktes mit einem Skalar mit einem Speicheraufwand von insgesamt 14 Speichereinheiten bewältigen.
   Von diesen 14 Speichereinheiten stehen nach den Punktberechnungen wieder zehn für

andere Operationen zur Verfügung. Vier Speichereinheiten werden für die Speicherung der Koordinaten der beiden Punkte verwendet. Bleibt nur noch zu analysieren, wieviel Speichereinheiten die übrigen Operationen, die das Einbinden der Nachricht nach Kapitel 4.2.1 erfordern, in Anspruch nehmen. Als erstes werden m_1 und m_2 berechnet, deren Berechnung zwar keine zusätzlichen Speichereinheiten erfordert, wohl aber deren Speicherung. Für die übrigen Operationen sind nun nur noch acht Speichereinheiten verfügbar.
Für alle zu verschlüsselnden Nachrichtenblöcke sind die Werte für x_3, y_3, x_4 und y_4 gleich, so daß es sinnvoll ist, diese einmalig zu berechnen und dann zwischenzuspeichern. Alle hierfür nach Kapitel 4.2.1 notwendigen Operationen können in den verbleibenden acht Speichereinheiten durchgeführt werden. Nach Speicherung der Werte x_3, y_3, x_4 und y_4 stehen nur noch vier Speichereinheiten zur Verfügung, mit denen sich die noch ausstehenden zwei XOR-Verknüpfungen und die restlichen zwei Multiplikationen realisieren lassen.
Zunächst werden mit der XOR-Verknüpfung die Zwischenergebnisse z_1 und z_2 berechnet. Zum Speichern der Ergebnisse werden zwei Speichereinheiten verwendet, so daß nun noch zwei Speichereinheiten zur Verfügung stehen. Die abschließenden Multiplikationen berechnen die zu übertragenden Werte c_1 und c_2. Da aber nach Tabelle 6.6 eine Multiplikation nur eine Speichereinheit erfordert, können die Werte c_1 und c_2 und die beiden Multiplikationen in den verbleibenden zwei Speichereinheiten gespeichert bzw. ausgeführt werden. Insgesamt kann also die gesamte Entschlüsselung mit 14 Speichereinheiten durchgeführt werden.

2. Entschlüsselung:
   Der Speichereinheitenbedarf ist bei der Entschlüsselung ähnlich der der Verschlüsselung. Bis zur Berechnung und Speicherung der Werte x_3, y_3, x_4 und y_4 unterscheidet sich der Speichereinheitenbedarf um vier Speichereinheiten. Bei der Entschlüsselung muß nur ein Punkt gespeichert werden und m_1 und m_2 ergeben sich erst noch aus c_1 und c_2. Damit sind vier Speichereinheiten weniger belegt als bei der Verschlüsselung. Somit stehen den restlichen Operationen, ausgehend von 14 Speichereinheiten der Entschlüsselung, noch acht Speichereinheiten zur Verfügung. Das reicht nach Tabelle 6.6 genau aus, um eine Invertierung zu berechnen. Da aber zwei Invertierungen notwendig sind, um x_4^{-1} und y_4^{-1} zu berechnen, und nach der ersten Invertierung nur noch sieben Speichereinheiten zur Verfügung stehen, erhöht sich der Speichereinheitenbedarf um eine Speichereinheit. Nach der zweiten Invertierung werden sieben Speichereinheiten wieder freigegeben. Diese reichen aus, um die zwei XOR-Verknüpfungen und die beiden Multiplikationen für die Rückgewinnung von m_1 und m_2 durchführen zu können. Danach können einige Zwischenergebnisse wie z_1 und z_2 aus Kapitel 4.3 gelöscht werden, so daß für die Umwandlung von m_1 und m_2 zu der Nachricht M keine zusätzlichen Speichereinheiten benötigt werden. Insgesamt werden demnach für die Entschlüsselung 15 Speichereinheiten benötigt.

Sowohl bei der Ver- als auch bei der Entschlüsselung sind die Speichereinheiten für die Pufferung des Nachrichtenblocks nicht berücksichtigt. Die Anzahl der Speichereinheiten hängt

von der gewählten Primzahl p ab. Da die maximale Länge des Nachrichtenblocks pro Ver- oder Entschlüsselungslauf $2\log_2 p - 3$ ist, reichen zwei Speichereinheiten der Länge $\log_2 p$ aus, um den Nachrichtenblock zwischenzuspeichern.
Soll der Chip in der Lage sein, eigene Schlüssel zu generieren, was notwendig sein kann, wenn die Systemverwaltung bei der Schlüsselgenerierung nicht die Sonderfälle aus Kapitel 4.4 beachtet, so muß ein Zufallsgenerator auf dem Chip integriert werden. Für die dann notwendigen Überprüfungen des zufällig generierten geheimen Schlüssels (siehe Kapitel 4.4) stehen ausreichend Speichereinheiten zur Verfügung.
Abschließend kann man sagen, daß der Chip mit 15 Speichereinheiten für das Operationswerk, zwei Speichereinheiten für die Speicherung des Nachrichtenblocks und zehn Speichereinheiten für die Systemparameter und die Schlüssel, also insgesamt mit 27 Speichereinheiten der Länge $\log_2 p$ auskommt. Bei einer Primzahl $p > 2^{256}$ ergibt sich dadurch ein Speicherbedarf von $27 \cdot \log_2(2^{256}) = 27 \cdot 256 = 6912$ Bit $=$ 864 Byte $\approx$ **1 kByte**.

Kapitel 7

Zusammenfassung und Ausblick

Die in dieser Arbeit vorgenommenen Untersuchungen haben ergeben, daß die Rechenzeit eines ElGamal-Verfahrens über Elliptische Kurven im wesentlichen von zwei Faktoren abhängt: Wahl des Endlichen Körpers $GF(p)$ und Wahl des geheimen Schlüssels (siehe Kapitel 6.1 und 6.2). Je größer die Primzahl ist, desto aufwendiger werden die Körperoperationen. Gleiches gilt bei der Ver- und Entschlüsselung für die Wahl des geheimen Schlüssels. Grund dafür ist die sehr komplexe Multiplikation eines Skalar mit einem Punkt der Elliptischen Kurve. Bereits eine Verdoppelung eines Punktes, also die Multiplikation mit dem Skalar 2, umfaßt nach Gleichung (2.8) vier Additionen, sechs Multiplikationen und eine Divisionen. Besonders die Division, die auf die Bestimmung des Inversen zurückgeführt wird, ist in $GF(p)$ komplex (siehe Kapitel 5.1.2). Ein großer geheimer Schlüssel sorgt aber dafür, daß die komplexen Additionen der Elliptischen Kurven häufig ausgeführt werden müssen. Die Folge sind intensive Rechenzeiten. Es gibt nun mehrere Möglichkeiten, die Rechenzeit zu beschränken.
Zum einen kann man eine kleinere Primzahl wählen. Dadurch wird aber die Berechnung des diskreten Logarithmus (siehe Kapitel 2.4) erleichtert. Das Kryptosystem verliert an Sicherheit.
Zum anderen kann man sich darauf beschränken, zum Verschlüsseln nur kleine geheime Schlüssel zu verwenden. Allerdings lassen die kürzeren Rechenzeiten potentielle Angreifer die Verwendung kürzerer Schlüssellängen vermuten.
Eine dritte Möglichkeit wäre die Reduzierung der Berechnungsschritte. Prinzipiell ist es möglich, für jeden Nachrichtenblock einen anderen Schlüssel zu verwenden. Bei den meisten Anwendungsgebieten ist diese Option nicht erforderlich, so daß sich der Rechenaufwand erheblich reduzieren wird. Anstatt bei jedem Nachrichtenblock die Punkte $(x_1, y_1) = s_s \cdot B$ und $(x_2, y_2) = s_s \cdot e_p$ (siehe Kapitel 4.2) bei der Verschlüsselung zu berechnen, werden diese nur einmal berechnet und dann gespeichert, wie es derzeit im Simulationsprogramm realisiert ist. Weitere Zeitersparnis kann man dadurch erzielen, daß man zusätzlich noch x_4 und y_4 (siehe Kapitel 4.2.1) einmalig berechnet und ebenfalls speichert. Somit entfallen die Schritte 2 bis 5 in Kapitel 4.2.1. Die gleiche Vorgehensweise bei der Entschlüsselung erspart die Berechnung des Punktes $(x_2, y_2) = e_s \cdot (x_1, y_1)$ und die Berechnung der Inversen, wenn man statt x_4 und y_4 (siehe Kapitel 4.3) gleich x_4^{-1} und y_4^{-1} berechnet und speichert.

Die einmalige Berechnung der Zwischenvariablen x_4, y_4 und ihrer Inverser ist im Simulationsprogramm derzeit noch nicht berücksichtigt.
Neben einer zeitlichen Untersuchung ist eine Speicherabschätzung ein wichtiger Aspekt für die Hardwarerealisierung. In Kapitel 6.4 wurde eine Hardwareabschätzung mit einer Minimal-ALU vorgenommen. Danach sind für die Realisierung des hier analysierten Kryptoverfahrens 27 Speichereinheiten der Länge $\log_2 p$ notwendig.
Für eine genauere Hardwareabschätzung wäre die Erstellung eines generischen VHDL-Modells (**V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage) wünschenswert. Über den Parameter p könnte man dann die Länge von p variieren und so den zusätzlichen Hardwareaufwand ermitteln. Außerdem könnten dann auch andere Konzepte für schnelle Reduktionen oder Additionen, wie sie zum Beispiel in [HAH97] beim RSA-Verfahren vorgeschlagen werden, umgesetzt werden. Mit Sicherheit lassen sich auf diese Art und Weise die Laufzeiten weiter reduzieren.
Weitere Verbesserungen lassen sich durch spezielle Hardwarestrukturen erzielen. So werden in [GOL94] und [HAH97] verschiedene Verfahren zur Beschleunigung von arithmetischen Funktionen analysiert. Durch diese Verfahren wäre eine Geschwindigkeitssteigerung der Addition auf Elliptischen Kurven denkbar.
Eine ganz andere Möglichkeit einer Geschwindigkeitssteigerung wäre die Wahl eines Endlichen Körpers der Form $GF(2^m)$. Dieses hätte eine neue Implementierung der Prozeduren zur Folge. Allgemein lassen sich einige Operationen in $GF(2^m)$ schneller berechnen als in $GF(p)$. Dazu gehört auch der diskrete Logarithmus, wodurch die Sicherheit reduziert wird.
Denkbare wäre auch, Ellipticthe Kurven in anderen Public-Key-Verfahren einzusetzen. Interessant wäre dann natürlich ein Vergleich zwischen dem hier vorgestellten und den anderen Verfahren.
Das nächste Ziel wird eine Umsetzung des hier vorgestellten und analysierten Verfahrens auf einen Kryptokontroller sein.

Literaturverzeichnis

[BES91] T. Beth, F. Schaefer, *Non Supersingular Elliptic Curves for Public Key Cryptosystems*, E(uropean) I(nstitut) for S(ystem) S(ecurity), Karlsruhe 1991

[BET83] T. Beth, *Kryptographie: Eine Einführung in die Methoden und Verfahren der geheimen Nachrichtenübermittlung*, Teubner, Stuttgart 1983

[BEU92] A. Beutelspacher, *Kryptologie*, Vieweg Verlag, Braunschweig, Wiesbaden 1992

[BEU95] A. Beutelspacher, J. Schwenk, K.-D. Wolfenstetter *Moderne Verfahren der Kryptographie*, Vieweg Verlag, Braunschweig, Wiesbaden 1995

[COH93] H. Cohen, *A Course in Computational Algebraic Number Theory*, Graduate Texts in Mathematics, Springer-Verlag, Berlin, Heidelberg 1993

[CTT94] J. Chao, K. Tanada, S. Tsujii, *Design of Elliptic Curves with Controllable Lower Boundary of Extension Degree for Reduction Attacks*, *Advances in CRYPTOLOGY - CRYPTO '94*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg 1994, S. 50 - 55

[ElG85] T. Elgamal, *A public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms*, IEEE Trans on Information Theory, Vol. IT-31 1985, S.

[GOL94] D. Gollmann, *Algorithmen-Entwurf in der Kryptographie*, Bibliographisches Institut & F.A. Brockhaus AG, Mannheim 1994

[HAH97] L. Hahn, *Konzeption und Implementation eines Hardwareakzelerators zum RSA-Verfahren*, Diplomarbeit am Fachbereich Informatik der Universität Hamburg, 1997

[HOR85] P. Horster, *Kryptologie*, Bibliographisches Institut Mannheim Wien Zürich, Zürich 1985

[HUS87] D. Husemoller, *Elliptic Curves*, Graduate texts in Math. 111, Springer-Verlag, New York 1987

[IEE95] Dr. a. Menezes, Dr. M. Qu, Dr. S. A. Vanstone, *IEEE P1363 Standard Part 6 Elliptic Curve Systems*, Ontario, 1995

[JUN93] D. Jungnickel, *Finite Fields*, Bibliographisches Institut & F.A. Brockhaus AG, Mannheim 1993

[KOB87] N. Koblitz, *A Course in Number Theory and Cryptography*, Graduate Texts in Mathematics, Springer-Verlag, New York 1987

[KNU81] D. E. Knuth, *The Art of Computer Programming - Seminumerical Algorithms*, Addison Wesley, 1981

[LEM95] R. Lercier, F. Morain, *Counting the Number of Points on Elliptic Curves over Finite Fields: Strategies and Performances*, *Advances in Cryptology - EUROCRYPT '95*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg 1995, S. 79 - 94

[MOR91] F. Morain, *Building Cyclic Elliptic Curves Modulo Large Primes*, *Advances in Cryptology - EUROCRYPT '91*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg 1991, S.

[MOV91] A. Menezes, T. Okamoto, S.A. Vanstone, *Reducing elliptic curves logarithms to logarithms in a finite field*, *Proceedings 23rd Annual ACM Symposium on Theory of Computing (STOC)*, ACM Press, S. 80 - 89

[MUL95] V. Müller, *Ein Algorithmus zur Bestimmung der Punktanzahl elliptischer Kurven über endlichen Körpern der Charakteristik größer drei*, Dissertation, Saarbrücken 1995

[RIE85] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhäuser, Boston 1985

[RIV78] R. L. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public key Cryptosystems*, Communications of the ACM, Vol. 21, No. 2, 1978, S. 120 - 126

[ROB95] M. J. B. Robshaw, *Security Estimate for 512-bit RSA*, RSA Laboratories, 1995

[SIL86] J. Silverman, *The Arithmetic of Elliptic Curves*, Graduate texts in Math. 106, Springer-Verlag, New York 1986

[SOP95] R. Schroeppel, H. Orman, S. O'Malley, O. Spatscheck, *Fast Key Exchange with Elliptic Curve Systems*, *Advances in Cryptology - CRYPTO '95*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg 1995, S. 43 - 56

[WAE71] B. L. van der Waerden, *Algebra I*, Springer-Verlag, Berlin, Heidelber, New York 1971

Anhang A

Installation

Das Simulationsprogramm ist in C++ erstellt worden und besteht aus folgenden Dateien:

1. *krypsy10*
   Kompilierte und ausführbare Programmdatei.

2. *krypsy10.cc*
   Sourcecode des gesamten Programms. Die notwendigen Prozeduren für die Operationen im Endlichen Körper und den Elliptischen Kurven sind hinter dem Hauptprogamm implementiert und entsprechend mit Kommentaren eingeleitet.

3. *ellcurve.dat*
   Diese Datei beinhaltet alle die zur Auswahl stehenden Elliptischen Kurven. Diese Datei kann um weitere Elliptische Kurven ergänzt werden. Zu beachten ist dabei, daß die Zahlen in der Reihenfolge eingegeben werden, wie es die ersten 4 Kommentarzeilen angeben, d.h. zuerst ist die Primzahl p einzugeben, dann sind es die Parameter a und b der Elliptischen Kurve E und schließlich deren Ordnung $\#E$, die bei diesem Simulationsprogramm immer **prim** sein muß. Dann entfällt jegliche Berechnung einer Ordnung eines Punktes, weil alle Punkte dieser Elliptischen Kurve die gleiche Ordnung haben. Die Eingabe muß jeweils mit einem return abgeschlossen werden.

4. *text.txt*
   Datei, die, solange kein anderer Dateiname angegeben wird, standardmäßig verschlüsselt wird. Es kann also entweder der Inhalt dieser Datei geändert oder eine andere Datei verwendet werden.

5. *vertext.txt*
   Standardmäßige Ausgabedatei der Verschlüsselung. Sollte diese Datei nicht vorhanden sein, so wird diese automatisch vom Programm erzeugt. Eine bereits vorhandene *vertext.txt* wird überschrieben, so daß diese Datei vor einer erneuten Verschlüsselung nicht explizit gelöscht werden muß. Vor der Entschlüsselung kann diese Datei manipuliert werden.

6. *enttext.txt*
   Standardmäßige Ausgabedatei der Entschlüsselung. Auch diese Datei wird, sofern noch nicht vorhanden, neu erzeugt. Eine bereits existente Datei wird überschrieben.

7. *readme.txt*
   Datei, die die Installationsanleitung und einige Anmerkungen zum Programm enthält.

Zur Installation des Simulationsprogramms müssen lediglich diese Dateien in ein Verzeichnis kopiert werden. Das Simulationsprogramm benutzt zusätzlich noch eine Klasse *Integer*, die beliebig große Integerzahlen generieren und verwalten kann. Diese Klasse gehört beim GNU C++ zum Standard. Alle C++-Compiler, die diese Klasse nicht zur Verfügung stellen, können den Sourcecode nicht kompilieren. Nach dem Kopieren der Dateien in ein Verzeichnis muß das Programm noch kompiliert werden. Das Programm wird dann etwa 0,5 MB groß sein. Folgender Befehl bewirkt die Kompilation:

g++ -g -Dsunos -Wall krypsy10.cc -lm -o krypsy10

Die Optionen -Wall veranlaßt den Kompiler, alle bei der Kompilation auftretenden Warnungen auf den Bildschirm auszugeben. Die Option -lm und -o binden das Programm und speichern den ausführbaren Programmcode in krypsy10. Die Option -Dsunos legt das Betriebssytem für die Stopuhr fest. Diese Option gilt für alle UNIX basierten Systemen, insbesondere auch für LINUX. Zum Starten des Programms geben Sie in dem Verzeichnis, in das Sie alle Dateien kopiert haben und in dem sich auch die ausführbare Datei *krypsy10* befindet, *krypsy10* ein.

Anhang B

Beispielsitzung

Anmerkung:
Die Textausgabe des Programms ist durch die Verwendung einer Nicht-Proportionalschrift gekennzeichnet, alle Benutzereingabe sind fett und in einer Proportionalschrift dargestellt. Kommentare und Anmerkungen zu den Ein- und Ausgaben des Programms sind an den entsprechenden Stellen in einer Proportionalschrift eingefügt. In der folgenden Beispielsitzung soll der folgende kurze Text verschlüsselt werden: Test!. Damit die Ausgabe nicht zu unübersichtlich wird, wird hier die Elliptische Kurve über $GF(33013)$ (nächst größere Primzahl nach 2^{15}) verwendet. Die zu verschlüsselnde Datei muß sich in dem gleichen Verzeichnis befinden wie das Programm. Alle Ausgabedateien des Programms werden im Programmverzeichnis abgespeichert.

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~                D I P L O M A R B E I T                          ~~*/
/*~~   Simulationsprogramm fuer Verschluesselung mit Elliptischen     ~~*/
/*~~   Kurven in GF(p) mit p < 2^257                                  ~~*/
/*~~   letzte Aenderung: Apr 12 1997 / 11:44:14                       ~~*/
/*~~   Version:          1.0                                          ~~*/
/*~~   Author:           Frank Bohnsack (Universitaet Hamburg)        ~~*/
/*~~                                                                  ~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
-I- Eingabe der zu verwendenden Dateien. Bei RETURN wird der
    [Standardname] uebernommen
```

`Zu verschluesselnde Datei [text.tx]:` **Dateiname** hier: test.txt
Hier erfolgt die Eingabe der zu verschlüsselnden Datei. **Dateiname** ist nur ein Platzhalter für einen richtigen Namen. Es können beliebige Dateien verschlüsselt werden. Anschließend gibt das Programm zur Kontrolle den von Ihnen eingegebenen Dateinamen aus. Wenn bei der Eingabe nicht explizit ein Dateiname spezifiziert worden ist, sondern nur die Eingabetaste gedrückt worden ist, dann wird die Standarddatei `text.txt` verwendet und angezeigt.
`Dateiname:` (Anzeige des eingegebenen Dateinamens, hier: text.txt)
`Ausgabedatei der Verschluesselung [vertext.txt]:` **Dateiname** hier: vertext.txt
Für die Festlegung der Ausgabedatei der Verschlüsselung gilt entsprechendes wie für die Eingabe der zu verschlüsselnden Datei. Die Datei wird, sofern vorhanden ohne Sicherheitsabfrage überschrieben, oder, falls noch nicht vorhanden, neu erstellt.
`Dateiname:` (Anzeige des eingegebenen Dateinamens, hier vertext.txt)
`Ausgabedatei der Entschluesselung [enttext.txt]:` **Dateiname** hier: enttext.txt
Für die Festlegung der Ausgabedatei der Entschlüsselung gilt entsprechendes wie für die Festlegung der Ausgabedatei der Verschlüsselung. Die Datei wird, sofern vorhanden, ohne Sicherheitsabfrage überschrieben oder, falls noch nicht vorhanden, neu angelegt.
`Dateiname:` (Anzeige des eingegebenen Dateinamens, hier: enttext.txt)
Danach erfolgt eine Anzeige der zur Verfügung stehenden Elliptischen Kurven. Sie stehen alle in der Datei *ellkurv.dat.*

```
Wahl 1:
p  = 33013
   = 16 (Laenge in Bits)
a  = 2155
b  = 16008
#E = 33247

Wahl 2:
p  = 2147530453
   = 32 (Laenge in Bits)
a  = 98352635
b  = 436545968
#E = 2147551387

Wahl 3:
p  = 281474976710677
   = 49 (Laenge in Bits)
a  = 189435610941038
b  = 75515067124755
#E = 281474955906763

Wahl 4:
p  = 9223372036854775837
   = 64 (Laenge in Bits)
a  = 448240303058735409
b  = 3780909410277807244
#E = 9223372040660500831

Wahl 5:
p  = 1208925819614629174706189
   = 81 (Laenge in Bits)
a  = 847165314234043257467033
b  = 1021068999450740572098702
#E = 1208925819613908125210647

Wahl 6:
p  = 79228162514264337593543950397
   = 97 (Laenge in Bits)
a  = 25437546580962336819455239926
b  = 3548165146374283422412303990
#E = 79228162514264682683340165871

Wahl 7:
p  = 5192296858534827628530496329220121
   = 113 (Laenge in Bits)
a  = 4568445875805131826206526727510123
b  = 5079242630140372761173830382824241
#E = 5192296858534827604947584259580067

Wahl 8:
p  = 170141183460469231731687303715884105757
   = 128 (Laenge in Bits)
a  = 105373295517709951970325062096778779566
b  = 34674159133580410663812342760227529126
#E = 170141183460469231746769722985284162023

Wahl 9:
p  = 22300745198530623141535718272648361505980591
   = 145 (Laenge in Bits)
a  = 4624846499014720750676808368856495147289261
b  = 18784836602758067039970460875331028899682402
#E = 22300745198530623141531894895957553105920471

Wahl 10:
p  = 1461501637330902918203684832716283019655932542983
```

```
   = 161 (Laenge in Bits)
a  = 1014146891404983229246246055584450554243723674111
b  = 9816001578368016852594719393228270149341239216691
#E = 1461501637330902918203685724056483559965429270731

Wahl 11:
p  = 95780971304118053647396689196894323976171195136475563
   = 177 (Laenge in Bits)
a  = 69784955616295004313504510201170712268743427320358931
b  = 42896401512311994257657733591469645024980295752750630
#E = 95780971304118053647396689261887683934041443669673823

Wahl 12:
p  = 6277101735386680763835789423207666416102355444464034513029
   = 193 (Laenge in Bits)
a  = 4344889333976977750645843808907118147593956219007300291351
b  = 1435686028627319807302680536778062005541797925652065298149
#E = 6277101735386680763835789423164123633814077227112925466289

Wahl 13:
p  = 411376139330301510538742295639337626245683966408394965837152631
   = 209 (Laenge in Bits)
a  = 381351024351511563462792462914982458394472960701079124756911966
b  = 173227377555620450909628010134795980265263647348432144950843172
#E = 411376139330301510538742295639337620660476765245511071660800559

Wahl 14:
p  = 26959946667150639794667015087019630673637144422540572481103610249951
   = 225 (Laenge in Bits)
a  = 10471129709377695273759749972345243642319537898929661542748486876156
b  = 9668287999529710866548454582387131383322070994296966553640030776124
#E = 26959946667150639794667015087019623264221048181407825712621551448779

Wahl 15:
p  = 1766847064778384329583297500742918515827483896875618958121606201292619891
   = 241 (Laenge in Bits)
a  = 997950869081843023842836014021686862592118906110301132662199573588802149
b  = 1756739154637730906730119219288508664446137657526980604110430271918419041
#E = 1766847064778384329583297500742918516708045485283239013300184448058758183

Wahl 16:
p  = 115792089237316195423570985008687907853269984665640564039457584007913129640233
   = 257 (Laenge in Bits)
a  = 95270011154113989137874045642563197904974382476028229496939583044928154917770
b  = 67197498288843822956561798249438724627559132613538319448066573513035828248 2
#E = 115792089237316195423570985008687907852926634160420486457496696723664680664331
```

`Eingabe Wahl [1..16]:` **Wahl** hier: 1

Zur Auswahl einer Elliptischen Kurve geben Sie einfach die Zahl ein, die hinter dem Wort `Wahl` über jeder Elliptischen Kurve angezeigt wird. Anschließend werden zur Kontrolle die gewählten Systemparameter angezeigt. Eine Eingabe außerhalb des gültigen Bereichs führt solange zu einer Fehlermeldung und einer erneuten Eingabeaufforderung der Wahl, bis die eingegebene Zahl im gültigen Eingabebereich liegt.

```
Primzahl   : 33013
Parameter a: 2155
Parameter b: 16008
#E         : 33247
Test, ob E Elliptische Kurve ist (1=ja, 0=Nein): 1
Eingabe der unteren Grenze der x-Koordinate des Basispunktes B: 12
```

Hiermit legen Sie den Startwert der x-Koordinate für die Bestimmung des Basispunktes fest. Dazu wird $y = \sqrt{x^3 + ax + b}$ bestimmt, sofern es zu dem eingegebenen x-Wert eine Lösung gibt. Anderenfalls wird eine Warnung ausgegeben und der x-Wert solange um 1 erhöht, bis ein entsprechendes y bestimmt und damit ein Basispunkt festgelegt wird. Anschließend werden alle Systemparameter des Kryptosystems ausgegeben.

```
-W- 10583 ist kein quadratischer Rest in GF(33013)
-W- 13207 ist kein quadratischer Rest in GF(33013)
```

```
SYSTEMPARAMETER:
Endlicher Koerper   : GF(33013)
Elliptische Kurve E : y = x + 2155 x + 16008
Ordnung von E       : #E = 33247
Basispunkt B        : B(14/13027)       mit #B = 33247
Generierung des oeffentlichen Schluessels vom Empfaenger:

Eingabe des geheimen Schluessels e in [2,33246] : 23
```

Mit der Eingabe einer Zahl im angegebenen Bereich wird der geheime Schlüssel des Empfängers festgelegt, aus dem dann der zur Verschlüsselung notwendige öffentliche Schlüssel generiert wird. Solange die Eingabe außerhalb des Bereichs $[2..\#B-1]$ liegt, wird folgende Fehlermeldung und erneute Eingabeaufforderung angezeigt:

```
-E- Eingabe ist nicht in [2,33246]
Erneute Eingabe des geheimen Schluessels e in [2,33246]:
```

Sollte dabei der öffentliche Schlüssel $e_p = e \cdot B$ eine 0 in der Y-Koordinate haben erscheint folgende Fehlermeldung mit erneuter Eingabeaufforderung:

```
-E- Wahl des Schluessels liefert bei Multiplikation mit
Basispunkt B(14,13027) den Punkt (*,0).
Die Y-Koordinate muss aber != 0 sein!
Erneute Eingabe des geheimen Schluessels e in [2,33246]:

Eingabe des geheimen Schluessels s in [2,33246] : 3
```

Mit der Eingabe einer Zahl im angegebenen Bereich wird der geheime Schlüssel des Senders festgelegt und der öffentliche, der keine weitere Bedeutung für den weiteren Ablauf des Simulationsprogramm hat, berechnet. Solange die Eingabe außerhalb des angegebenen Bereichs $[2..\#B-1]$ liegt, wird folgende Fehlermeldung und erneute Eingabeaufforderung angezeigt:

```
-E- Eingabe ist nicht in [2,33246]
Erneute Eingabe des geheimen Schluessels s in [2,33246]:
```

Sollte dabei der berechnete Punkt $(x_2, y_2) = s \cdot e_p$ eine 0 in der X- oder Y-Koordinate haben, wird folgende Fehlermeldung und erneute Eingabeaufforderung angezeigt:

```
-E- Wahl des Schluessels liefert bei Multiplikation mit
oeffentlichem Schluessel (,) den Punkt (,).
Die X-Koordinate und die Y-Koordinate muss aber != 0 sein!
Erneute Eingabe des geheimen Schluessels s in [2,33246]:
```

Es folgt die Ausgabe der geheimen und öffentlichen Schlüssel des Empfängers und des Senders mit Überprüfung der berechneten öffentlichen Schlüssel, ob diese auf der Elliptische Kurve E (Ausgabe 1) liegen oder nicht (Ausgabe 0). Der Basispunkt wird ebenfalls überprüft.

```
Schluessel Empfaenger :
1. oeffentlich: (5132/6718)
2. geheim     : 23
Schluessel Sender :
1. oeffentlich: (2760/15179)
2. geheim     : 3
y          = 15909
x + ax + b = 15909
Test, ob Basispunkt auf E: 1
y          = 2753
x + ax + b = 2753
Test, ob e_public auf E: 1
y          = 4314
x + ax + b = 4314
Test, ob s_public auf E: 1
Maximale Anzahl von Bytes zur Verschluesselung: 3
```

Die maximale Anzahl gibt an, wie groß der zu verschlüsselnde Nachrichtenblock pro Verschlüsselungszyklus sein darf. Jeder Nachrichtenblock wird folgendermaßen angezeigt (in unserem Beispiel sind dieses 2):

```
Eingegebener Text: Tes
Eingegebener Text als Integerzahl: 5530995
Padding String PS: 63
Nachricht M: 1062495603
Verschluesselte Nachricht:
(2760, 15179, 18938, 24996)

Eingegebener Text: t!

Eingegebener Text als Integerzahl: 7610634
Padding String PS: 63
Nachricht M: 1064575242
Verschluesselte Nachricht:
(2760, 15179, 28326, 21637)
```

`Zu entschluesselnde Textdatei [vertext.txt]:` **Dateiname** hier: vertext.txt
Nach der Verschlüsselung ist eine Manipulation des verschlüsselten Textes möglich, bevor dieser wieder entschlüsselt wird. Damit ist es möglich, gezielte Manipulationen durchzuführen und deren Auswirkungen zu untersuchen. Es ist auch möglich, eine vorher selbst geschrieben Datei entschlüsseln zu lassen, dabei muß jede Zeile mit der Eingabetaste enden und jede Zeile muß aus vier Zahlen, getrennt jeweils durch ein Komma, bestehen. Der gewählte Dateiname wird danach angezeigt. Beim Betätigen der Eingabetaste wird der Standardname in den eckigen Klammern übernommen. Wenn die Datei nicht vorhanden ist, wird das Programm, ohne eine Entschlüsselung durchzuführen, fortgesetzt. Bei der Entschlüsselung wird die Entschlüsselung jedes Nachrichtenblocks wie folgt angezeigt (in unserem Beispiel sind dieses 2):

```
Dateiname: vertext.txt
x1 : 2760
y1 : 15179
c1 : 18938
c2 : 24996
Die entschluesselte Nachricht lautet: 5530995
Entschluesselter Text: Tes

x1 : 2760
y1 : 15179
c1 : 28326
c2 : 21637
Die entschluesselte Nachricht lautet: 7610634
Entschluesselter Text: t!
```

Abschließend werden die folgenden CPU-Rechenzeiten ausgegeben. Alle Benutzereingaben werden in der Zeitmessung nicht berücksichtigt.

```
RECHENZEITEN:
CPU-Rechenzeit Bestimmung B       : 0
CPU-Rechenzeit Schluessel e_public: 0 s
CPU-Rechenzeit Schluessel s_public: 0 s
CPU-Rechenzeit Verschluesselung   : 0.01 s
CPU-Rechenzeit Entschluesselung   : 0.02 s


Weitere Verschluesselung mit gleichen Systemparametern? (j/n):
```

Zum Schluß besteht die Möglichkeit, erneut eine Ver- und Entschlüsselung mit denselben Systemparametern vorzunehmen (Eingabe beginnend mit j oder y). Das Programm beginnt dann bei der Festlegung der geheimen Schlüssel von Sender und Empfänger.

Anhang C

Danksagung

Hiermit möchte ich mich bei Herrn Dr. Volker Müller von der Universität Darmstadt ganz herzlich für die mir zur Verfügung gestellten Elliptischen Kurven bedanken, ohne die eine Analyse nur sehr schwer möglich gewesen wäre, und für die anregenden Diskussionen und Erläuterungen, die mir den Zugang zu einigen Problemstellungen erleichtert und zu einigen neuen Ansätzen angeregt haben. Besonderer Dank gebührt auch der Firma Philips Semiconductors Hamburg, insbesondere den Herren Dr. Steffen Drews und Helmut Hörner, für die Anregungen aus der Praxis, Diskussionen und die Behilflichkeit bei der Literaturrecherche. Auch möchte ich den Mitarbeitern des Arbeitsbereich TECH der Universität Hamburg für die tatkräftige Unterstützung bei Problemen jeglicher Art bedanken. Hierbei ist insbesondere Herr Manfred Grove zu benennen, der mir in der Einarbeitungsphase in die theoretischen Grundlagen Literatur zur Verfügung stellte und bei Fragen hierzu mit Rat und Tat zur Seite stand. Ferner möchte ich meinem Diplomarbeitsbetreuer Dr. Reinhard Rauscher für die vielen Hilfestellungen und fruchtbaren Diskussionen, die immer motivierend waren, bedanken. Insbesondere möchte ich mich auch bei Herrn Prof. Dr. Valk für die Übernahme des Koreferats und die Möglichkeit der Vorstellung meiner Diplomarbeit im Oberseminar des Arbeitsbereichs TGI der Universität Hamburg bedanken.

Erklärung

Hiermit erkläre ich an Eides statt, die vorliegende Arbeit selbständig durchgeführt zu haben und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Uetersen, den 27.06.1997

(Frank Bohnsack)