Diplomarbeit

Konzeption, Evaluierung und Implementation eines Akzeleratorsfür elliptische Kurven

Siegmund Gorr

Universität Hamburg

Fachbereich Informatik

Februar 2000

Inhaltsverzeichnis Seite

1 Einleitung

Die Kryptographie zieht derzeit mit *rsasantem* Tempo in den Alltag ein. Was vor einigen Jahren noch weitestgehend nur Insider oder Randgruppen betraf, ist heutzutage in aller Munde. Fast täglich wird man mit Sicherheitstechnischen Anwendungen konfrontiert: Sei es die PIN für den Geldautomaten, das Handy, Telefonkarten, die Wegfahrsperre bei Autos, die Verwendung von verschlüsselten oder signierten e-mails, oder gesicherte Internetverbindungen - Die Liste kann nahezu endlos fortgesetzt werden. Die Verwendung von Kryptographie ist oftmals benutzertransparent in das Produkt eingebunden, der Benutzer merkt überhaupt nicht, dass verschlüsselt wird. Hinter all diesen Anwendungen stecken kryptographische Methoden, mit denen Informationen gegen *unbefugten* Zugang oder Gebrauch geschützt werden sollen.
Wie bereits gesagt, die Kryptographie ist auf dem besten Wege, ein fester Bestandteil des Alltags zu werden. Einen guten Einblick in die Materie bietet [WOB97] mit dem Titel *Abenteuer Kryptologie*.

Sofern Berechnungen schnell und mit geringer Hardware erfolgen müssen, kommen Akzeleratoren für kryptographische Anwendungen ins Blickfeld des Betrachters: Sie bieten eine hohe Performance für Anwendungen kryptographischer Verfahren.
Diese Diplomarbeit behandelt einen derartigen Hardware-Akzelerator: Das besondere an diesem Akzelerator ist die Unterstützung von Operationen auf elliptischen Kurven.

Kern dieser Arbeit ist die Bestimmung von Realisierungsentscheidungen für die Implementierung eines Akzelerators für elliptische Kurven. Kryptographischen Verfahren, welche auf den Akzelerator aufsetzen, sind nicht Gegenstand der Untersuchung, dennoch enthält diese Arbeit einige Bezüge zu den kryptographischen Anwendungen. Dies resultiert nicht zuletzt aus den unterschiedlichen Anforderungen einzelner Anwendungen an die Sicherheit: Insbesondere der noch relativ junge Anwendungsbereich *Low-Security* (s. Kap. 1.4) scheint für diese Anwendungen prädestiniert zu sein.

Im Folgenden wird zunächst ein kleiner Überblick über die Geschichte der Kryptographie gegeben. Es folgt ein Abschnitt über die Historie dieser Arbeit im Zusammenhang mit anderen Arbeiten, sowie eine Einleitung zu elliptischen Kurven. Dann folgt das Kapitel über den Anwendungsbereich *Low-Security*, gefolgt von einer Spezifikation der in dieser Arbeit verwendeten Algorithmen. Die Einleitung wird durch eine kurze Gliederung der gesamten Arbeit abgeschlossen.

1.1 Historie der Kryptographie

Die Kryptologie ist eine Jahrtausende alte Wissenschaft. Sie beschäftigt sich mit der Lehre von Geheimschriften (Chiffren) und ihrer Entzifferung. Bereits die Römer haben sie angewendet, Cäsar verwendete eine konstante Verschiebung der Buchstaben, wobei das Alphabet in einem Kreis angeordnet wurde. Die Methode wurde nach ihrem populären Anwender Cäsar-Chiffre genannt. Derartige Methoden sind für heutige Anwendungen nicht sicher genug. Heutzutage verwendet man sehr ausgefeilte Methoden, deren Hintergrund in schwierigen mathematischen Problemen verwurzelt ist.

Kryptologie kann man in zwei Bereich unterteilen: Die **Kryptographie**[1], die sich mit der Verschlüsselung von Daten, also dem Geheimhalten beschäftigt und der **Kryptanalyse**. Die Kryptanalyse beschäftigt sich mit dem (unberufenen) Entziffern von Geheimschriften.
Oftmals wird die **Steganographie**[2] ebenfalls zur Kryptologie gezählt, dies ist aber nicht ganz richtig. Steganographie beschäftigt sich mit dem Verstecken einer Nachricht in einer anderen und es existieren nur in einem Teilbereich der Steganographie Berührungspunkte mit der Kryptographie.

Anwendungen kryptographischer Verfahren finden sich sowohl im militärischen, als auch immer mehr im zivilen Bereich. Im militärischen Bereich wird durch sie die Kommunikation einzelner Teilnehmer gegenüber unerwünschten Mithörern bzw. Feinden und Angreifern erschwert. Jedoch sind auch Zugangssysteme mittels kryptographischer Verfahren gesichert, so sind beispielsweise sämtliche größeren Marschflugkörper mit nuklearen Sprengköpfen mit derartigen Verfahren gesichert. Doch die Verfahren werden nicht nur im militärischen Bereich eingesetzt.
Auch im zivilen Bereich bei der Sicherung von finanziellen Transaktionen wurden schon frühzeitig kryptographische Verfahren eingesetzt. Die Verwendung der persönlichen Identifikations Nummer (PIN) bei EC-Karten ist ein weitbekanntes Beispiel für den zivilen Bereich. Mit der Verbreitung der Informationstechnik werden auch zunehmend elektronische Nachrichten mittels kryptographischer Verfahren zur Authentifizierung und Verschlüsselung eingesetzt. Bekanntestes Beispiel hierfür dürfte die für den privaten Bereich kostenlose Software *Pretty Good Privacy* (PGP, eine Beschreibung findet man in [WOB97]) sein.

Es wurde bereits darauf hingewiesen, dass heutzutage schwierige Probleme der Mathematik als Grundlage für Kryptographische Verfahren dienen. Nach [BAU94] finden folgende Disziplinen in der Kryptologie eine oder mehrere Anwendungen:

- Zahlentheorie
- Gruppentheorie
- Kombinatorik
- Komplexitätstheorie
- Ergodentheorie
- Informationstheorie

Als wichtigsten Hintergrund für kryptographische Verfahren sei hier ein Teil der Komplexitätstheorie erwähnt, die strukturelle Komplexitätstheorie. Sie beschäftigt sich mit der Struktur von Problemen. Die Kryptographie benötigt spezielle Funktionen, die im Idealfall so genannte Einwegfunktionen (*trapdoor functions*) darstellen, das sind Funktionen, die nicht umkehrbar sind. Es existieren mehrere Kandidaten für Einwegfunktionen, generell ist jedoch weder die Existenz noch die Nicht-Existenz von Einwegfunktionen bewiesen. Dies zeigt den *offenen* theoretischen Hintergrund der Kryptographie auf und macht deutlich, dass viele Dinge hier noch ungeklärt sind.

Elliptische Kurven haben geeignete Eigenschaften, um sie für kryptographische Verfahren anwenden zu können. Elliptische Kurven sind in der Mathematik seit langem bekannt, der

---

[1] Der Begriff *cryptography* bzw. *Kryptographie* wurde von John Wilkins 1641 eingeführt, vgl. [BAU94].
[2] Der Begriff stammt vermutlich aus dem Jahr 1655 von Caspar Schott, vgl. [BAU94].

Bezug zu Ellipsen entstammt dem Zusammenhang elliptischer Kurven mit elliptischen Integralen. Derartige Integrale treten auf, wenn man Bogenlängen von Ellipsen berechnet.

Elliptische Kurven werden für kryptographische Verfahren verwendet, sie beschreiben kein eigenständiges Verfahren. Vielmehr stellen sie mit der auf ihnen definierbaren abelschen Gruppenoperation ein Hilfsmittel dar, mit dem verschiedene kryptographische Verfahren konzipiert werden können.
Elliptische Kurven sind somit universell in der Kryptographie einsetzbar, ähnlich wie die Restklassenringe (Z/nZ). Und haben entsprechend ihrer breiten Einsetzbarkeit eine große Bedeutung für kommerzielle kryptographische Anwendungen.

Bei allen Verfahren, die auf elliptischen Kurven basieren, steht die Berechnung der **Skalarmultiplikation**:

$$n \cdot B \text{ mit } n \in \mathbb{N},\ B \text{ Punkt auf der elliptischen Kurve } E$$

im Vordergrund.

1.2 Kryptoakzeleratoren

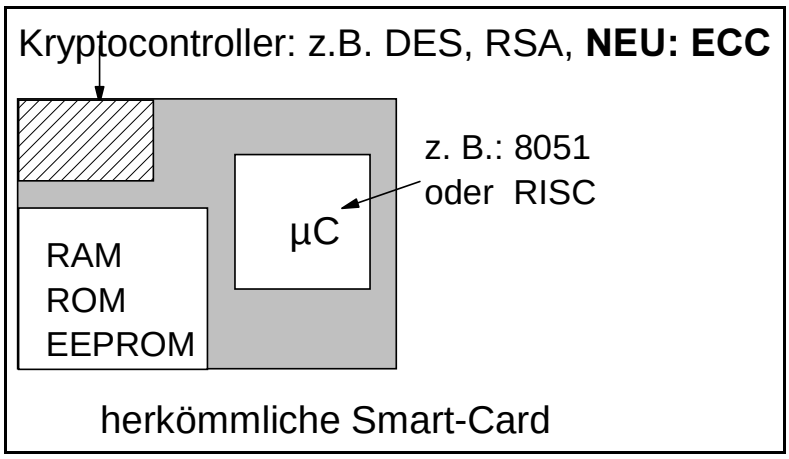

Abbildung 1.1: Übersicht eines Systems mit einem Kryptoakzelerator

Kryptoakzeleratoren sind für eine bestimmte Aufgabe ausgelegt, der Aufgabenbereich lässt sich am besten mit herkömmlichen Co-Prozessoren, die für Fließkommaarithmetik eingesetzt werden, vergleichen. Während ein Standard-Prozessor die rechenintensive Fließkommaarithmetik nur ungenügend bewältigt, wird diese Aufgabe bei entsprechendem Bedarf von einem Co-Prozessor übernommen. Hierdurch werden zwei Ziele verwirklicht: Zum einen ist die eigentliche Fließkomma-Berechnung schneller und zum anderen kann der Haupt-Prozessor in der Zwischenzeit andere Aufgaben übernehmen.
Bei Kryptoakzeleratoren verhält es sich genauso, nur, dass keine Fließkommanumerik, sondern modulare Exponentiation oder eben Berechnungen über elliptischen Kurven beschleunigt werden.
Das Konzept der Verwendung von Kryptoakzeleratoren ist nicht neu und es existieren bereits viele Akzeleratoren, deren Spezialgebiet die modulare Exponentiation ist. Sie werden überwiegend für RSA Implementierungen verwendet.

Oftmals werden Kryptoakzeleratoren auch eingesetzt, um Attacken deutlich zu erschweren: Denn die Erhöhung der Rechenkapazität erlaubt es längere Schlüssellängen zu verwenden, wodurch die Sicherheit erhöht wird. Die Erhöhung der Rechenkapazität für Hardware ist oftmals leichter zu bewerkstelligen als für Software.

Während herkömmliche Kryptoakzeleratoren fast ausschließlich auf die Akzeleration des RSA-Verfahrens (eine Beschreibung des Verfahrens findet man in [FOR96] und [SCH96]) zugeschnitten sind, fehlen weitgehend noch entsprechende Untersuchungen für die Akzeleration elliptischer Kurven. Akzeleratoren für elliptische Kurven benötigen eine andere Struktur als Akzeleratoren für die modulare Exponentiation.

Zudem zeichnet sich in der Industrie eine breite Anwendbarkeit von Verfahren über elliptischen Kurven ab. Dies zeigt, dass ein entsprechender Bedarf an derartigen Akzeleratoren vorhanden ist.
Ein weiterer Vorteil von elliptischen Kurven besteht darin, dass sie vermutlich bei gleicher Schlüssellänge ein höheres Maß an Sicherheit bieten [LV99].

1.3 Vorgeschichte dieser Diplomarbeit

Vorgänger dieser Arbeit und gewisser maßen als Grundlage dient die Diplomarbeit von F. Bohnsack [BOH97] über die Tauglichkeit von elliptischen Kurven für kryptographische Verfahren über dem Körper GF(p), wobei p eine Primzahl ist. Dort wird in Form einer Machbarkeitsstudie analysiert, inwiefern sich elliptische Kurven für Implementierungen eignen und wo Berechnungsengpässe vorliegen. Die Arbeit enthält auch Analysen über den Einfluß der Kurvengröße, die Wahl der Schlüssel und der Nachricht, auf die Rechenzeit.

In der Diplomarbeit von F. Bohnsack wurde der Körper GF(p) verwendet, um ein Verschlüsselungsverfahren zu realisieren. Bohnsack hat verschiedene Analysen über Ausführungszeiten bei der Verschlüsselung vorgenommen.

Mittlerweile hat sich herausgestellt, dass Verfahren über GF(2^k) in der Industrie bevorzugt werden, u. a. Ergebnis eines ersten Gesprächs mit Vertretern der Firma Philips. Die Arbeit von Bohnsack hat gezeigt, dass sehr viel Rechenzeit und auch Implementierungsaufwand für die „mod-P“-Berechnung benötigt wurde. In dieser Diplomarbeit wird demzufolge der Körper GF(2^k) verwendet werden, der zudem auch sehr hardwarenah ist.

Bei dem mathematischen Modell spielt die Rechenzeit zunächst eine untergeordnete Rolle, dies ändert sich bei dem Übergang zu konkreten Realzeitanwendungen. Standardanwendungen, die für die konzipierte Arbeit avisiert werden, sind Low-Security-Anwendungen, wie beispielsweise Telefonkarten, Mautkarten, Pay-Card, etc.
Anregungen und Hinweise aus der Praxis werden von der Firma Philips Semiconductors beigesteuert.

Durch diesen Anwendungsbereich ist der Lösungssuchraum stark eingeschränkt. Ferner lässt sich aus den Vorgaben ableiten, dass Beschränkungen hinsichtlich einer Hardwareimplementierung bezüglich der Kriterien:

- Rechenzeit,
- Taktfrequenz,
- Fläche,
- Leistungsaufnahme,
- Testbarkeit,
- Fehlertoleranz
- und maximale Schlüssellänge

gegeben sind. Die Zielfunktion einer Optimierung verlangt eine Minimierung der ersten vier Kriterien und eine Maximierung der letzten beiden. Unglücklicherweise widersprechen sich bereits die ersten vier Kriterien. In der Praxis muss ein Kompromiss gefunden werden.

Die Kryptographie mittels elliptischer Kurven wurde bereits angerissen. Was bislang noch nicht problematisiert wurde, sind die Realisierungsdetails. Bei elliptischen Kurven unterscheiden sich Implementierungen für unterschiedliche Körper sehr stark voneinander. Ursache hierfür ist, dass elliptische Kurven über unterschiedlichen mathematischen Körpern definiert werden können und die Auswahl des zu verwendenden Körper hat weitreichende Auswirkungen auf die Recheneinheiten bzw. auf den Akzelerator.
Zu den Implementierungsparametern elliptischer Kurven zählen:

- der zugrunde liegende Körper,
- die Kurvenparameter
- und die Zahlenrepräsentation von Elementen aus diesem Körper.

Insbesondere der zugrunde liegende Körper bestimmt die Problematik der Arithmetik auf diesen Kurven. Letztendlich bestimmt die Struktur des zugrunde liegenden Körpers die Berechnungskomplexität von Operationen auf den elliptischen Kurven.

1.4 Bedeutung und Historie elliptischer Kurven

Die Anwendungsgebiete für elliptische Kurven sind vielfältig: Elliptische Kurven spielen eine große Rolle bei der Faktorisierung von natürlichen Zahlen. Sie können ferner für Primzahltests eingesetzt werden, einen Einblick in die Anwendungsfelder verschafft die Diplomarbeit von S. Hamdy [HAM98].
Neben diesen Anwendungsgebieten existieren weitere, noch relativ neue Anwendungen. Unter neuen Anwendungen ist hier die rasante Entwicklung der Bedeutung von elliptischen Kurven im Gebiet der Kryptographie gemeint.
Aus dem Schattendasein eines Randgebietes in der Mathematik ist eine wichtige und umfassende Grundlage für Verschlüsselungsmethoden geworden. Die Bedeutung der elliptischen Kurven wird vermutlich noch weiter steigen, da sie eine hohe Sicherheit mit vergleichsweise geringem Aufwand für die Berechnungen vereinen.
Noch dominiert das RSA-Verfahren bei kryptographischen Verfahren in der Kategorie Public-Key-Systeme. Die Stärke dieses und ähnlicher modular-exponentieller Verfahren ist jedoch mit einem erheblichen Resourcenaufwand an Berechnungen verbunden.
Eine Erhöhung der Sicherheit dieser Verfahren, die auf einer sukzessiven Erhöhung der Schlüssellänge hinausläuft, ist bereits in absehbarer Zeit nicht mehr wirtschaftlich vertretbar bzw. für viele praktische Anwendungen zu langsam. Bei diesen Überlegungen ist die Entwicklung von schnelleren Rechnern bereits berücksichtigt worden.
Verwendet man nun elliptische Kurven für die Verfahren, so ist eine Erhöhung der Schlüssellänge mit wesentlich geringerem Berechnungsmehraufwand verbunden.

1.5 Low-Security Systeme

Unter *Low-Security-System* versteht man Systeme für Anwendungen mit geringeren Sicherheitsanforderungen. Die Sicherheit bei diesen Systemen darf jedoch nicht mit nur scheinbarer Sicherheit erlangt werden. Beispiele für diese Methode der scheinbaren Sicherheit lieferten erste Anwendungen in Softwaresystemen, in denen lediglich die Buchstabenrepräsentationen fest verändert wurden, wie Vertauschung der Bytehälften o.ä. Vielmehr stellt diese Kategorie von Sicherheitsanwendungen andere Anforderungen an die Rechenkapazität, dies bedeutet im Allgemeinen, dass der Schlüsselraum erheblich eingeschränkt ist.
Generell gilt jedoch auch für Low-Security Systeme das gleiche wie für Systeme der höchsten Sicherheitsstufe, von denen man im Allgemeinen verlangt, dass der Schlüsselraum auch in absehbarer Zeit nicht zu durchsuchen ist. Über den Begriff des Schlüsselraums kann man die Forderung aufstellen, dass die vollständige Durchdringung des Schlüsselraums zugleich auch die beste Angriffs-Methode sein sollte. Effektiv bedeutet dies, dass es neben dem einfachen Durchprobieren aller möglichen Schlüssel keine bessere Methode gibt, um die Sicherheit des Systems zu kompromittieren.
Der Markt für Sicherheitssysteme ist nicht zuletzt durch die Verbreitung von vernetzten Dienstleistungen enorm gestiegen. Die Sicherheitsanforderungen für Güter oder Informationen bestimmen sich nicht zuletzt aus dem Wert, den sie verkörpern. Hochwertige Güter bedürfen einer *besseren* Sicherung als geringwertige Güter. Die letztgenannte Gruppe lässt sich dem Schlagwort Low-Security zuordnen.

Nachfrage Low-Security

Die Nachfrage nach Sicherheit im Low-Security Bereich ist erheblich und gerade hier besteht der Bedarf an kostengünstigen, einfach handhabbaren Systemen. Für Massenprodukte auf diesem Gebiet bieten sich je nach Ausgangslage Software- oder Hardwarelösungen an. Steht beispielsweise ein Rechensystem zur Verfügung, so kann Software implementiert werden. Steht ein derartiges System nicht zur Verfügung oder möchte man die Zugriffsmöglichkeiten einschränken, so bietet sich eine Hardwareimplementierung an.
Da es sich zugleich oftmals um Massenware handelt, bieten sich Chips auf Basis von Full-Custom oder Standard-Zellen Entwürfen an. Für einfachste Fälle, bei kleinen Stückzahlen, können auch programmierbare Logik-Zellen (FPGA, etc.) verwendet werden. Wobei angemerkt sei, dass die Art des verwendeten Entwurfs sehr stark von den benötigten Stückzahlen abhängt: Bei geringen Stückzahlen und häufigen Änderungen im Design, sind programmierbare Logikzellen sinnvoll. Bei hohen Stückzahlen kommen nur Standard-Zellen oder Full-Custom Entwürfe in Betracht.

Low-Security Anforderungen

Die Anforderungen in dem Anwendungsgebiet Low-Security sind: minimaler Hardware, mit minimaler Leistungsaufnahme auf möglichst kleiner Chipfläche. Dies ist nur gültig, sofern keine sonstige Hardware vorhanden ist: Bei Internetapplikationen ist stets eine hohe Rechenleistung vorhanden. Es sind jedoch spezielle Vorgaben einzuhalten. Hierzu gehören absolute Forderungen wie die Reaktionszeit und relative Anforderungen an die Leistungsaufnahme mit der verwendeten Fläche. Die einzelnen Parameter schließen sich, wie bei allen Bewertungsfunktionen für Chip-Entwicklungen, gegenseitig aus, so dass eine geeignete Kompromisslösung gefunden werden muss.

1.6 Notation von Algorithmen

Für die Notation von Algorithmen in dieser Arbeit wurde eine *synthetische Sprache* (Pseudo Code) verwendet. Sie lässt sich auf beliebige imperative Programmiersprachen portieren. Hierbei wurde besonders darauf geachtet, dass die Algorithmen einfach verständliche Anweisungen verwenden und möglichst mit einer minimalen Zahl von Sprach-Befehlen auskommen. Im Folgenden sind **optionale** Teile in eckigen Klammern [*optional*] und einzusetzende Ausdrücke in spitzen Klammern und kursiver Schrift angegeben. Eine **Auswahl** an Alternativen wird durch runde Klammern und dem „|"-Strich als Begrenzer zwischen den Alternativen notiert. Dies entspricht einem Teil der Bakus-Naur-Syntax Notation, wie sie oftmals zur Beschreibung von Programmiersprachensyntax verwendet wird.
Für Schleifen existieren die Konstrukte, wobei die Schleifenvariable stets **ganzzahlige** Werte annimmt:

for <*Schleifenvariable*>:=<*Initialwert*> (**to**|**downto**) <*Zielwert*>
    [*Anweisung(en*)]
next <*Schleifenvariable*>

Fallentscheidungen erfolgen mit:

if <*Ausdruck*> **then** [*Anweisung*] [**else** [*Anweisung*]]

oder

if <*Ausdruck*> **then**
    [*Anweisungen*]
[**else**
    [*Anweisungen*]]
end

Folgen einer Fallentscheidung mehere Anweisungen so sind diese um eine Stufe im Text eingerückt, in diesem Fall ist zur klaren Trennung auch ein „**end**" hinter der letzten Anweisung innerhalb der Fallunterscheidung angegeben.

Als unbedingte Verzweigung wird der Sprungbefehl:

goto <*Marke*>

verwendet. Marken für die Sprungbefehle entsprechen den Nummern vor den Anweisungen.

Variablenzuweisungen erfolgen mit dem ← Operator:

<*Variable*> ← <*Ausdruck*>

Die Terminierung und das Resultat eines Algorithmus wird mit dem Befehl **stop with** angegeben, dem Befehl kann ein Ausdruck folgen:

stop with <*Ausdruck*>

Mit dieser Notation sind alle wesentlichen Angaben für die in dieser Arbeit verwendeten Algorithmen erklärt.

1.7 Aufteilung der Diplomarbeit

Die Diplomarbeit teilt sich in folgende Bereiche auf:

- Grundlagen elliptischer Kurven über GF(2^m) (Kapitel 2 und 3),
- Evaluierung verschiedener Berechnungsalternativen und Zeitvergleiche (Kapitel 4 und 5),
- und Implementierung eines Akzelerators für Verfahren über elliptische Kurven als ASIC (**A**pplication **S**pecific **I**ntegrated **C**ircuit) (Kapitel 6 und 7).
- Zusammenfassung der Ergebnisse und Ausblick (Kapitel 8).

2 Mathematische Grundlagen

In diesem Kapitel sollen die mathematischen Grundlagen für das Berechnungsmodell elliptischer Kurven vorgestellt werden. Einleitend werden algebraische Strukturen wie Gruppen, Körper und Körpererweiterungen beschrieben. Hierauf aufbauend werden die elliptischen Kurven zunächst allgemein beschrieben, um anschließend auf Sonderfälle elliptischer Kurven bei endlichen Körpern einzugehen.

2.1 Körper

Zu Beginn der mathematischen Grundlagen seien hier kurz die benötigten algebraischen Strukturen definiert, einen tieferen Einblick in Körper findet man u. a. in [LID82], [LN94] und [LEU96]:

Definition: Eine abelsche **Gruppe** ist ein Paar $(M,\bullet)$ einer Menge M und einer Verknüpfung $\bullet$, für die folgendes gelte:

i)	$\forall a,b\ \exists c : c = a \bullet b$	$a,b,c \in M$	
ii)	$\forall a\ \exists a^{-1} : a \bullet a^{-1} = e$	$a,a^{-1},e \in M$	(*Existenz inverser Elemente*)
iii)	$\forall a\ : a \bullet e = a$	$a,e \in M$	(*e ist neutrales Element*)
iv)	$\forall a,b \in M : a \bullet b = b \bullet a$		(*Kommutativität*)
v)	$\forall a,b,c \in M : a \bullet (b \bullet c) = (a \bullet b) \bullet c$		(*Assoziativität*)

e bezeichnet das neutrale Element und für jedes Element a existiert ein Inverses a^{-1}.

Definition: Ein **Körper** ist ein Tripel $(M,\oplus,\otimes)$ mit einer Menge M und zwei Operationen $\oplus$ und $\otimes$.
$(M,\oplus)$ ist ein *abelsche* Gruppe mit neutralen Element 0.
$(M\backslash\{0\},\otimes)$ ist ein *abelsche* Gruppe mit neutralen Element 1.
Es gelte das Distributivgesetz:
$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ $\quad a,b,c \in M$

Körpereigenschaften

Um die *Charakteristik* von Körpern definieren zu können, ist es erforderlich den Begriff der *Vielfachen* von Körperelementen zu definieren.

Definition: Das **positive Vielfache** v eines Körperelements $a \in K$ sei $v \circ a$:

$\circ : \mathbb{N}_+ \times K \rightarrow K$ mit $v \in \mathbb{N}_+$, $a \in K$

$$v \circ a = \begin{cases} a & \text{falls } v = 1 \\ ((v\text{-}1) \circ a) \oplus a & \text{falls } v > 1 \end{cases}$$

Definition: Die **Charakteristik** $\chi(K)$ eines Körpers K ist das kleinste positive Vielfache k für das gilt:

$\chi: K \rightarrow \mathbb{N}$

$$\chi(K)= \begin{cases} k & \text{falls } k \text{ das kleinste positive Vielfache ist, für das gilt: } k \circ a = 0 \ \forall\, a \in K \\ 0 & \text{falls kein derartiges positive Vielfaches existiert} \end{cases}$$

Anmerkung: Für $k > 0$ ist k stets eine Primzahl [LEU96].

So gilt beispielsweise: $\chi(\mathbb{R})=0$, $\chi(\mathbb{C})=0$, $\chi(GF(2))=2$, $\chi(GF(4))=4$

Definition **Unterkörper**:

Eine Teilmenge U eines Körpers K mit $U \subseteq K$ heißt Unterkörper, wenn U eine Teilmenge von K ist und bezüglich der Operationen auch ein Körper ist.
Dementsprechend wird K Erweiterungskörper von U genannt.

Ein Körper heißt *Primkörper*, wenn er außer sich selbst keine weiteren Unterkörper besitzt. Der Durchschnitt aller Unterkörper bildet den Primkörper (vgl. [LID82], S. 14).

Körper kann man bezüglich ihrer Elementzahl charakterisieren, man unterscheidet *endliche* und *unendliche* Körper.
In dieser Arbeit werden ausschließlich elliptische Kurven über endlichen Körpern behandelt. Diese Einschränkung resultiert daraus, dass generell nur endliche Körper für Kryptoverfahren verwendet werden können. Bei unendlichen Körpern ist bereits der Speicherplatz für die Repräsentation der Körperelemente nicht begrenzt, d. h. für alle praktisch realisierbaren Varianten muss eine Beschränkung stattfinden.

Körpererweiterungen

Neben der Charakteristik kann man Körper auch bezüglich ihrer Erweiterung klassifizieren. Die Charakteristik bei Erweiterungskörpern bezieht sich nur auf den Primkörper, also dem Körper, der dem Erweiterungskörper zugrunde liegt. Endliche Erweiterungskörper werden oftmals in folgender Form notiert:

$$GF(p^q) \text{ oder } F_{p^q}$$

Hierbei steht GF für Galois-Field, benannt nach dem Mathematiker Evariste Galois. Er war einer der ersten Mathematiker, der sich intensiv mit endlichen Körpern beschäftigte und eine entsprechende Theorie zur Verfügung gestellt hat. Field ist die englische Bezeichnung für mathematische Körper. Bei dieser Notation ist p der Grundkörper und q der Erweiterungsgrad. p steht für eine Primzahl und beschreibt zugleich die Charakteristik des Körpers. q gibt nun an, wieviele Tupel ein Element dieses Körpers hat. Im Folgenden werden nur endliche Körper und deren endliche Erweiterungen betrachtet. Nimmt man den $GF(3^4)$, so haben die Elemente folgende Form:

$$(a_1, a_2, a_3, a_4) \text{ mit } a_i \in \{0, 1, 2\}$$

Eine einfache Möglichkeit mit diesen Elementen zu rechnen, besteht in der Verwendung von Polynomen des Grades q-1. Die Elemente des Körpers GF(3^4) können wie folgt aufgefasst werden:

$$a_1 + a_2 x + a_3 x^2 + a_4 x^3$$

Die a_i sind Koeffizienten aus dem Körper GF(3). Wie man mit diesen Elementen rechnen kann, wird im folgenden Abschnitt beschrieben.

Arithmetik im Primkörper

Im Primkörper, also ohne eine Erweiterung, ist die Addition und Multiplikation wie bei den ganzen Zahlen definiert, jedoch mit dem Zusatz, dass das Ergebnis einer Rechenoperation *reduziert* werden muss, falls es größer oder gleich p ist.
Reduktion ist hier der Vorgang, eine natürliche Zahl auf den Rest einer Division abzubilden, wobei der Divisor der Primzahl entspricht:

$$f : \mathbb{N} \rightarrow \{0, 1, ..., p-1\}$$
$$x \mapsto x \bmod p \Leftrightarrow x - \left\lfloor \frac{x}{p} \right\rfloor p$$

Die Division, bei der nur der Rest verwendet wird, nennt man auch **Modulo-Operation**, ihre abgekürzte Notation ist mod. Folgendes Beispiel soll den Sachverhalt im Körper GF(3) verdeutlichen:

2+2 ergibt 4, was jedoch noch reduziert werden muss: 4 mod 3 = 1.
Also gilt im Körper GF(3): 2+2 = 1.

Ähnlich der Addition wird auch die Multiplikation ausgeführt:

Erst wird das Produkt wie gewohnt berechnet und anschließend reduziert:
2×2 ergibt 4 und 4 mod 3 = 1, also gilt $2 \times 2 = 1$.

Schwieriger wird es jedoch, die Division zu beschreiben: Die Theorie für die Existenz multiplikativer Inverse zu jedem Element außer der Null ist recht umfangreich, aus diesem Grund sei an dieser Stelle erneut auf entsprechende Lehrbücher der Mathematik [LID82] verwiesen. Die multiplikativen Inversen werden für die Berechnung mit elliptischen Kurven benötigt, was weiter unten in diesem Kapitel noch konkretisiert wird.

Es existieren Verfahren, mit denen man die multiplikativen Inversen berechnen kann. Von diesen Verfahren werden hier zwei vorgestellt:

1. Potenzierung der zu invertierenden Zahl a mit der Anzahl der Körperelemente minus zwei.

$$a^{2^k-2} \equiv a^{-1} \bmod p_{irr} \quad \text{mit } a \in \text{GF}(2^k) \tag{2.1}$$

Die Zahl der Körperelemente im GF(2^k) ist 2^k und p_{irr} sei das *irreduzible Polynom*. (Der Begriff irreduzibles Polynom wird im folgenden Abschnitt *Arithmetik im Erweiterungskörper*, auf S. 15 erläutert.)

(2.1) ist eine Anwendung des so genannten kleinen Satzes von Fermat [FOR96]:

$$a^{p-1} \equiv 1 \mod p \tag{2.2}$$

Wobei p eine Primzahl und a eine ganze Zahl ($a \in$ Z) sei. Ferner gelte die Bedingung, dass a nicht durch p teilbar sei ($a \nmid p$).
Spaltet man nun auf der linken Seite von Gleichung (2.2) ein a ab, so wird ersichtlich, dass a^{p-2} das multiplikative Inverse von a ist, da das Produkt 1 ergibt:

$$a^{p-1} \equiv a \cdot a^{p-2} \equiv 1 \pmod p$$

Und somit gilt:

$$a^{p-2} \equiv a^{-1} \pmod p$$

Somit ist die Beziehung zwischen (2.2) und (2.1) aufgezeigt.

2. Verwendung des erweiterten euklidischen Algorithmus.

Der erweiterte euklidische Algorithmus (vgl. [KNU98]) ist im Anhang (Anhang: 1.1) angegeben. Er berechnet den größten gemeinsamen Teiler zweier Zahlen in linearer Darstellung:

$$1 \equiv p\,u + x\,v \pmod p \tag{2.3}$$

Der erweiterte euklidische Algorithmus berechnet die Variablen u und v. p ist die Körpercharakteristik und x die zu invertierende Zahl. In der Gleichung ist der linke Summenterm stets 0 modulo p, da p ein Faktor ist. Die Gleichung kann also verkürzt werden:

$$1 \equiv x\,v \pmod p \tag{2.4}$$

Dies bedeutet aber, dass v das multiplikative Inverse zu x ist, welches gesucht wurde.

Der erweiterte euklidische Algorithmus berechnet den größten gemeinsamen Teiler zweier Zahlen. Bei der Bestimmung des multiplikativen Inversen wird der größte gemeinsame Teiler (*gcd* = greatest common divisor) des irreduziblen Polynoms und der zu invertierenden Zahl berechnet:

$$\gcd(irr, x)$$

In dieser Anwendung ist eine Zahl jedoch prim bzw. irreduzibel und hat somit per Definition nur 1 als größten gemeinsamen Teiler. Diese Tatsache wird bei der Berechnung des multiplikativen Inversen ausgenutzt. Eine detaillierte Beschreibung sowohl für ganze Zahlen als auch für Polynome findet man in [KNU98].

Arithmetik im Erweiterungskörper

Die Arithmetik des Erweiterungskörpers baut im wesentlichen auf der des Primkörpers auf, sie setzt sich aus Methoden für die Addition und Multiplikation zusammen und wird in diesem Abschnitt erklärt.

Die Arithmetik im Erweiterungskörper ist für die **Addition** und Subtraktion einfach eine komponentenweise Berechnung mit den Methoden für den Primkörper.

Beispiel für GF(3^2): ($2x$+1) + ($1x$+1) = $0x$+2 = 2
Beispiel für GF(2^2): ($1x$+1) + ($1x$+0) = $0x$+1 = 1

Die Multiplikation kann bei der Betrachtung der Elemente als Polynome einfach durch Ausmultiplizierung erfolgen. Im allgemeinen Fall muss anschließend noch reduziert werden. Zur Reduktion benötigt man ein so genanntes **irreduzibles Polynom**.

Definition: Ein Polynom über einem Körper K heißt genau dann **irreduzibel**, wenn es sich nicht als Produkt zweier Polynome über K vom Grad > 0 darstellen lässt ([ZEI96], S. 709).

Definition: Es sei $f(x) := \sum_{i=0}^{k} a_i x^i \neq 0$ ein Polynom aus dem Körper K: $f(x) \in K$.

Dann heißt die größte natürliche Zahl k, für die $a_k \neq 0$ ist, **Grad** von $f(x)$ bzw. **deg**($f(x)$). Der zugehörige Koeffizient heißt höchster Koeffizient des Polynoms. Ein Polynom wird *normiert* genannt, wenn der höchste Koeffizient 1 ist.

Für jeden endlichen Erweiterungskörper gibt es ein irreduzibles Polynom (vgl. [LID82], Satz 2.6, Kap. VII), jedoch ist seine Bestimmung nicht einfach: Es existieren verschiedene Algorithmen, um irreduzible Polynome zu ermitteln. Im Prinzip kann zufällig ein Polynom gewählt werden und geprüft werden, ob es keine Faktoren enthält. In der Praxis verwendet man schnellere Algorithmen, deren Laufzeitkomplexität nichtdeterministisch polynomiell ist (vgl. [D7-98], Annex A). Für Praxisanwendungen kann man auf bestehende Tabellen von irreduziblen Polynomen (z.B. in [D7-98]) zurückgreifen.
Das irreduzible Polynom ist mit der Körpercharakteristik im Primkörper vergleichbar, welche ebenfalls eine Primzahl ist (vgl. Vgl. [LEU96], S. 25, 30 f.).

Die Bestimmung des irreduziblen Polynoms hat Ähnlichkeit mit dem Auffinden von Primzahlen bei natürlichen Zahlen. Man kann sich ein Polynom wählen und für alle Polynome geringeren Grades prüfen, ob sie Faktoren dieses Polynoms sind. Kann kein Faktor gefunden werden, so ist das Polynom irreduzibel. Der Test ist also analog dem direkten Verfahren zur Primzahlverifikation bei natürlichen Zahlen.

Ist das irreduzible Polynom zugleich normiert, so wird es auch als **Primpolynom** bezeichnet. Normiert bedeutet, dass der Koeffizient des höchsten Terms 1 ist. Im GF(2) ist jedes Polynom normiert und daher jedes irreduzible Polynom ein Primpolynom.

Zurück zur Multiplikation im endlichen Körper. Sie kann durch Ausmultiplizierung der beiden Multiplikanden, in Form von Polynomen, erfolgen. Das Ergebnis ist ein Polynom, dessen Grad gleich der Summe der beiden Multipikandengrade ist:

$$A \cdot B = C \qquad \begin{aligned} A &= a_m x^m + \ldots + a_1 x + a_0 \\ B &= b_n x^n + \ldots + b_1 x + b_0 \\ C &= c_{m+n} x^{m+n} + \ldots + c_1 x + c_0 \end{aligned} \tag{2.5}$$

Die Reduktion soll nun an einem Beispiel dargestellt werden. Verwendet wird der endliche Körper GF(2^4):

Das irreduzible Polynom sei $x^4 + x + 1$ und das folgende Produkt soll berechnet werden:

$$(x^2 + 1)(x^3 + x + 1) = x^5 + x^2 + x + 1 \tag{2.6}$$

Das Ergebnispolynom hat den Grad 5. Der Grad 5 kann nicht mehr in GF(2^4) dargestellt werden, da er größer gleich dem Grad des irreduziblen Polynoms ist. Das Zwischenergebnis muss noch *reduziert* werden. Die *Reduktion* ist der Rest bei der Division durch das irreduzible Polynom. Die Polynomdivision lautet:

$$(x^5 + x^2 + x + 1) \div (x^4 + x + 1) = x \text{ Rest } 1 \tag{2.7}$$

Der Rest ist 1, also ist das Ergebnis von $(x^2 + 1)(x^3 + x + 1)$ = 1 und liegt im GF(2^4).

Eine andere einfache Methode besteht in der Betrachtung der Polynomkoeffizienten, es handelt sich um die gleiche Methode, lediglich die Repräsentation ist unterschiedlich. Für das vorige Beispiel ist die Notation in Koeffizienten der folgenden Abbildung 2.1 zu entnehmen:

5	4	3	2	1	0	← Potenzen von x
1	0	0	1	1	1	Dividend in Koeffizienten
1	0	0	1	1		Irreduzibles Polynom mal x
					1	Subtraktionsergebnis

Abbildung 2.1: Reduktion von $x^5 + x^2 + x + 1$ in GF(2^4), Koeffizientenansicht

Man sieht sogar, dass die Elemente in diesem Beispiel offenbar gegenseitig multiplikativ invers sind, da ihr Produkt 1 ergibt.
Das multiplikative Inverse kann nun direkt für die Realisierung der Division herangezogen werden: Möchte man also beispielsweise durch $(x^2 + 1)$ dividieren, so multipliziert man einfach mit dem Inversen $(x^3 + x + 1)$ und bekommt das entsprechende Ergebnis:

$$x \div (x^2 + 1) = x \cdot (x^2 + 1)^{-1} = x \cdot (x^3 + x + 1) = x^4 + x^2 + x = x^2 + 1$$

Zusammenfassend betrachtet, wird die Arithmetik im Wesentlichen durch den Primkörper bestimmt: Die Addition und Subtraktion erfolgt Komponentenweise. Bei der Multiplikation

und Division kann man die gleichen Verfahren wie bei der Schulmethode verwenden - mit dem einzigen Unterschied, dass keine Überträge existieren.

Endliche Körper der Charakteristik 2

Hier soll kurz der Körper GF(p^q) in der Parametrisierung $p = 2$ betrachtet werden. Körper der Charakteristik 2 haben einige Besonderheiten, die sich besonders für die Hardwarerealisierung eignen.
Dies ist zum einen die *Selbstinversität* bezüglich der Addition. Durch die Addition zweier gleicher Zahlen a entsteht ein Faktor 2, welcher im GF(2) 0 entspricht. Somit ergibt die Addition zweier Zahlen mit identischen Komponenten im Erweiterungskörper stets die 0:

Addition im GF(2): $\forall\, a \in \mathrm{GF}(2)\colon a + a = 2\,a = 0\;\; a = 0$
Addition im GF(2^k): $\forall\, a \in \mathrm{GF}(2^k)\colon (a_1,\ldots, a_k) + (a_1,\ldots, a_k) = (2a_1,\ldots, 2\,a_k) = (0, \ldots, 0) = 0$

Hierdurch sind positive und negative Zahlen identisch, da es nur zwei Zahlen (0,1) gibt ist -1 = 1. Die Addition entspricht der Subtraktion wodurch eine weitere Unterscheidung zwischen Addition und Subtraktion überflüssig ist.
Und des Weiteren sind die Koeffizienten einfach als Bits handhabbar. Aus diesem Grund ist die Addition eine einfache komponentenweise Exklusiv-Oder-Verknüpfung. Die Betrachtung von zeitaufwendigen Additionsüberläufen einzelner Stellen kann entfallen.

Die Addition erfolgt also sehr schnell und ohne die Verzögerung eines gegebenenfalls nachlaufenden Übertrags, wie dies bei der Addition von natürlichen Zahlen erforderlich ist.

Als Beispiel dient der bereits oben verwendete GF(2^4): Die Addition zweier Zahlen soll veranschaulicht werden:

Es soll die Summe der Polynome $x^2 + 1$ und $x^3 + x + 1$ berechnet werden:

$$(x^2 + 1) + (x^3 + x + 1) = x^3 + x^2 + x$$

Auch in dem Fall der Addition kann man alternativ nur die Koeffizienten der Polynome betrachten:

3	2	1	0	← Potenzen von x
1	0	1	1	Summand 2 in Koeffizienten
0	1	0	1	Summand 1 in Koeffizienten
1	1	1	0	Summe

Abbildung 2.2: Addition von $x^3 + x + 1$ und $x^2 + 1$, Koeffizientenansicht

Erweiterungskörper und Basen

Erweiterungskörper können auch als Vektorräume betrachtet werden, wobei die Koeffizienten der Polynome als Vektoren betrachtet werden. Vektoren und Vektorräume sind Bestandteile der linearen Algebra.

Bei Vektorräumen können verschiedene Basen verwendet werden: Die Existenz einer Basis ist für jeden Vektorraum über einem Körper gegeben. Jede maximale linear unabhängige Teilmenge von Vektoren stellt eine Basis dar (vgl. [JÄN91], S. 58 f.; [DTV82], S. 87).
Betrachtet man die Elemente als Polynome, so ist die Basis *B* eine Menge von Vektoren aus Potenzen der Unbekannten *x*:

$$B = \{x^0, x^1, ..., x^{k-1}\} \text{ für den } GF(2^k) \tag{2.8}$$

Diese Basis wird als **polynomielle Basis** bezeichnet.

Für den GF(2^k) sind jedoch auch andere Basen von großem Interesse. In der Praxis sind spezielle Basen der Form:

$$B = \{a, a^2, a^{2^2}, a^{2^4}, ..., a^{2^{k-1}}\} \text{ mit } a \in GF(2^k) \tag{2.9}$$

besonders wichtig. Diese Basisform wird als **normale Basis** bezeichnet. Elemente des GF(2^k) mit dieser Basis werden wie folgt allgemein dargestellt:

$$b = \sum_{i=0}^{m-1} b_i a^{2^i} \qquad b \in GF(2^k) \tag{2.10}$$

Die Null wird durch den Nullvektor repräsentiert und die Eins durch einen Vektor bzw. Tupel mit Einsen:

$$0 \Rightarrow (0, ..., 0), \; 1 \Rightarrow (1, ..., 1) \tag{2.11}$$

Der Vorteil der normalen Basis liegt darin, dass eine Elementquadrierung einfach eine zyklische Komponentenverschiebung darstellt:

$$b^2 = (b_0, ..., b_{m-1})^2 = (b_{m-1}, b_0, ..., b_{m-2}) \tag{2.12}$$

Dieses vielleicht überraschende Ergebnis resultiert aus zwei Dingen, die im Körper GF(2^k) gelten:

1. Die Quadrierung eines Körperelements hat keine Auswirkung, d. h.

   $$c^2 = c \quad \forall c \in GF(2) \tag{2.13}$$

   Man bedenke, dass die Komponenten hier nur 0 und 1 sind.

2. Und zweiter Grund ist, dass die Potenzierung eines Elements mit 2^k der Potenzierung mit 1 im GF(2^k) entspricht:

   $$d^{2^k} = d \quad \forall d \in GF(2^k) \tag{2.14}$$

   Der Hintergrund hierfür liegt in der Zahlentheorie und hängt mit der Gruppenordnung zusammen. Die Problematik wird in [LEU96] ausführlich behandelt.

Nun soll der Vorgang der Quadrierung kurz im Detail wiedergegeben werden:

$$b^2 = (b_0, ..., b_{m-1})^2 = \left(\sum_{i=0}^{m-1} b_i \alpha^{2^i} \right)^2 = \sum_{i=0}^{m-1} b_i^2 \alpha^{2^{i+1}} = \sum_{i=0}^{m-1} b_i \alpha^{2^{i+1}} = (b_{m-1}, b_0, ..., b_{m-2})$$

Wobei b ein Element aus GF(2^k) sei und die oben angeführte normale Basis verwendet wird.

Mit der obigen Repräsentation von Elementen erhält man zugleich Formeln für die Multiplikation von zwei Körperelementen. Die folgende Formel ist bereits umgeformt und vereinfacht, eine Herleitung findet man in [HAM98] (S. 43):

$$a \cdot b = c, \; c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_{i+k} b_{j+k} \lambda_{ij} \tag{2.15}$$

Die zusätzlichen Koeffizienten λ_{ij} müssen nur einmal für einen bestimmten Körper GF(2^k) berechnet werden. Der Aufwand für eine Multiplikation zweier Körperelemente, unter Verwendung der Multiplikationsmatrix λ, beträgt O(m^3).
Der tatsächliche Multiplikationsaufwand kann durch eine möglichst große Zahl von Nullen unter diesen Koeffizienten verringert werden. Man kann so bis zu einer Größenordnung von O(m^2) kommen. Jedoch existiert eine derartige Basis nicht für jeden Erweiterungskörper, eine eingehende Übersicht findet man in [D7-98] im Annex A.

Die Existenz von geeigneten Basen ähnelt dem Problem der dünn besetzten irreduziblen Polynome: Auch bei den dünn besetzten Polynomen existieren nicht für alle Erweiterungsgrade des GF(2^k) entsprechende Trinome. Eine äquivalente Einschränkung existiert auch bei den normalen Basen, jedoch geht es hier um die dünnere Besetzung der Multiplikationsmatrix λ.

Vergleicht man die normale Basis mit der polynomiellen, so erkennt man, dass bei dieser Zahlenrepräsentation ein irreduzibles Polynom auf Kosten der Koeffizientenmatrix entfällt. Bei der polynomiellen Basis wurde das irreduzible Polynom lediglich bei der Multiplikation benötigt, bei normalen Basen ist es bereits in der Multiplikationsmatrix integriert.

Als weiterführende Literatur für normale Basen für elliptische Kurven empfiehlt sich [D7-98, MEN93]. In der ersten Quelle befinden sich auch praktische Algorithmen für Tests auf die Existenz von bestimmten Basen und verschiedene Multiplikationsalgorithmen.

Wichtigster Unterschied zwischen normalen und polynomiellen Basen ist, dass erstere einen erheblich größeren Speicher für die Multiplikationsmatrix benötigen. Dieser Speicherbedarf liegt bei mindestens m^2 Einheiten für Elemente aus dem GF(2), sofern eine dünne Besetzung existiert, andernfalls werden m^3 Elemente des GF(2) benötigt.

Durch den im Vergleich zu polynomiellen Basen außerordentlichen Speicherbedarf, wird der Ansatz mit normalen Basen in dieser Arbeit nicht weiter verfolgt.

2.2 Elliptische Kurven

Nachdem die algebraischen Strukturen behandelt wurden, werden nun die elliptischen Kurven und die Operationen auf ihnen besprochen.

Eine *elliptische Kurve* ist ein Tripel *(E,O,K)*:

$$E = \{(x,y) : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{O\} \quad x,y,a_i \in K \tag{2.16}$$

$O \in E$, O ist der Unendlichkeitspunkt

K ist ein Körper, über den die Kurve E definiert ist. Die Kurzform lautet E/K

Da E über K definiert ist, stammen die Konstanten a_1, a_2, a_3, a_4 und a_6, sowie Variablen x und y, aus dem Körper K. Als Körper K dient in dieser Arbeit ausschließlich der GF(2^k). Des Weiteren kann auf den Punkten der elliptischen Kurve eine *Gruppenoperation* $\oplus$ definiert werden. Die Gruppenoperation wird nach der Beschreibung der elliptischen Kurven im Kapitel 2.3 näher erläutert.

Der **Unendlichkeitspunkt** O liegt auf der Gerade der Kurve E zum Unendlichen. Genauer gesagt ist er im Unendlichen. Dies ist kein Widerspruch zu der Tatsache, dass man elliptische Kurven, wie auch in diesem Fall, über endlichen Körpern definieren kann.
Wichtig für die Verarbeitung in Rechenanlagen ist jedoch, dass O eine andere Kodierung als die restlichen Punkte der Kurve benötigt, um ihn als Unendlichkeitspunkt identifizieren zu können.

Die Anzahl der Punkte einer elliptischen Kurve wird **Kurvenordnung** genannt, die Bestimmung der Kurvenordnung einer gegebenen Kurve wird in Kapitel 3.1 (S. 39 f.) behandelt.

Der Unendlichkeitspunkt spielt eine wesentliche Rolle bei der weiter unten definierten **Gruppenoperation** (s. Kap. 2.3), die ohne diesen Punkt nicht möglich wäre. Die Bedeutung des Unendlichkeitspunktes soll jedoch zuvor in der projektiven Ebene besprochen werden.

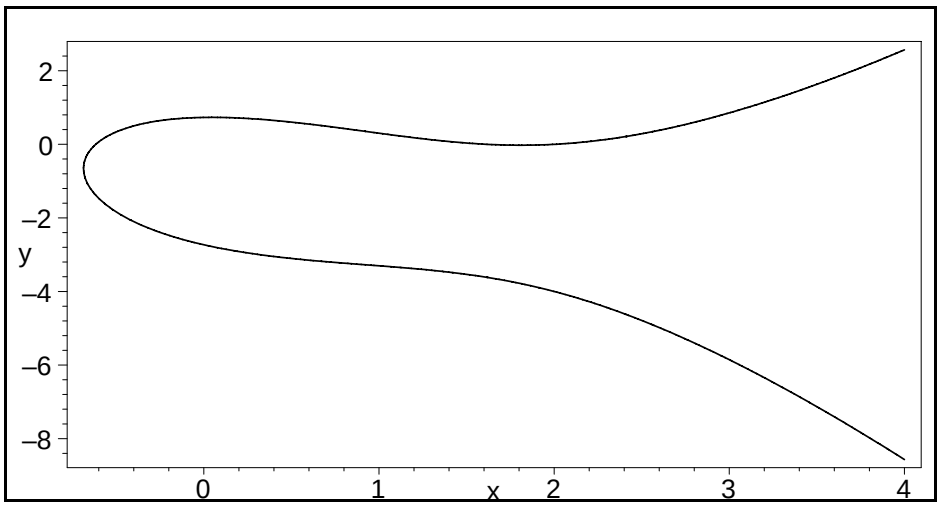

Abbildung 2.3: Elliptische Kurve über $\mathbb{R}$ $y^2 + xy + 2y = x^3 - 3x^2 + x + 2$

Projektive Ebene

In *projektiven Ebenen* können die elliptischen Kurven inklusive der Unendlichkeitspunkte beschrieben werden. Hier wird nicht mehr zwischen *normalen* und *unendlichen* Punkten unterschieden.

Zunächst sei an dieser Stelle die Definition einer affinen Ebene in Erinnerung gerufen:

Eine Menge von Punkten und Geraden heißt **affine Ebene** (vgl. [DTV87, S. 139]), wenn gilt:

i) Zu je zwei verschiedenen Punkten *A* und *B* gibt es genau eine Gerade *g*, die mit beiden inzidiert (Inzident bedeutet, dass mind. ein gemeinsamer Punkt existiert.)

ii) Zu jeder Geraden *g* gibt es einen Punkt *P*, der nicht mit ihr inzidiert.

iii) Zu jeder Geraden *g* und jedem Punkt *P*, der nicht mit *g* inzidiert, gibt es genau eine Gerade *h*, die mit *P* inzidiert und die mit *g* keinen Punkt gemeinsam hat (Parallele zu *g* durch *P*).

iv) Liegen die Ecken eines Sechsecks abwechselnd auf zwei Geraden (*g* und *h*) und sind zwei Paare von Gegenseiten parallel, so auch das dritte Paar.
Anmerkung: Dieser Punkt ist in Abb. 2.4 skizziert. *g* und *h* müssen nicht parallel zueinander sein.

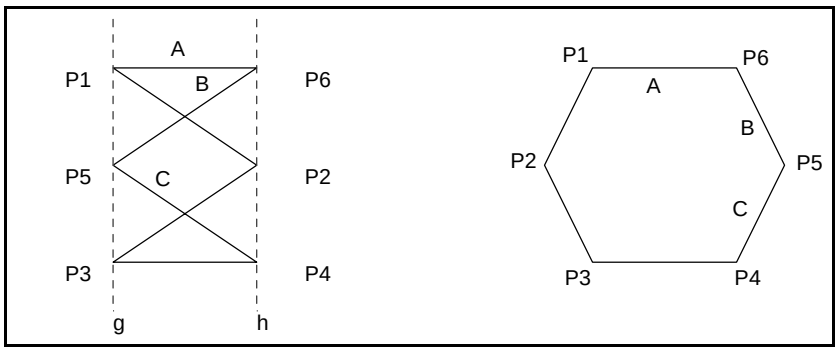

Abbildung 2.4: Sechseck zur Definition der affinen Ebene (Punkt 4)

In der affinen Ebene stört die Unterscheidung zwischen sich schneidenden und parallelen Geraden. Mit der Erweiterung der affinen Ebene zu einer projektiven Ebene, kann diese Unterscheidung entfallen. Bevor auf die Hintergründe dieser geforderten Eigenschaft eingegangen werden kann, erfolgt eine Definition der projektiven Ebene:

Eine Menge von Punkten und Geraden heißt **projektive Ebene** (vgl. [DTV82], S. 139), wenn gilt:

- Zu je zwei verschiedenen Punkten A und B gibt es genau eine Gerade g, die mit beiden inzidiert.

- Zu je zwei verschiedenen Geraden g und h gibt es genau einen Punkt P, der mit beiden inzidiert.

- Es gibt vier Punkte, von denen keine drei mit einer Geraden inzidieren.

- Liegen die Ecken eines Sechsecks abwechselnd auf zwei Geraden, so sind die Schnittpunkte der Gegenseiten kollinear. (Kollinear bedeutet linear abhängig, d.h. die Schnittpunkte der Gegenseiten liegen auf einer Geraden.)

Der letzte Punkt ist eine Fortführung von Abb. 2.4, da nun zwei unterschiedliche Geraden genau einen Schnittpunkt haben. In Abb. 2.4 liegen diese Schnittpunkte auf der *uneigentlichen Geraden* (s.u.) bzw. im Unendlichen.

Alternativ zu obiger Definition kann man die projektive Ebene auch konstruktiv aus einer affinen Ebene erzeugen:

Eine **projektive Ebene** kann leicht von einer *affinen* Ebene aus konstruiert werden und zwar fügt man pro Richtung einer Geraden einen *uneigentlichen Punkt* hinzu, der auf allen Geraden mit dieser Richtung liege. Die Menge dieser uneigentlichen Punkte werde als *uneigentliche Gerade* bezeichnet.
Umgekehrt kann man auch von projektiven zu affinen Ebenen gelangen. Man entferne eine beliebige Gerade und alle auf ihr liegenden Punkte. Das Ergebnis ist eine affine Ebene.

Nach dieser Definition soll nun auf die Hintergründe der geforderten Eigenschaft eingegangen werden. Gefordert wurde, dass jeweils zwei verschiedene Geraden, insbesondere auch parallele Geraden, genau einen gemeinsamen (inzidenten) Punkt haben sollen, dies ist bei projektiven Ebenen der Fall.
Betrachtet man eine elliptische Kurve, und nimmt zwei (nicht notwendiger Weise verscheide) Punkte der Kurve und verbinde diese, so erhält man stets einen dritten Schnittpunkt auf der Kurve. Dies ist in der affinen Ebene nicht immer der Fall. Dort benötigt man den Unendlichkeitspunkt, um einen Schnittpunkt mit Parallelen zur Y-Achse zu erhalten.
Die Konstruktion dieser Schnittpunkte ist Grundlage der Gruppenoperation auf elliptischen Kurven. Bevor diese Gruppenoperation beschrieben werden kann, ist es notwendig kurz auf die Koordinaten in der projektiven Ebene einzugehen.

Homogene Koordinaten

In der projektiven Ebene können Koordinaten wie in der affinen Ebene definiert werden.

Man erhält Koordinaten der projektiven Ebene über einem Körper *K*, wenn man Tripel aus diesem Körper betrachtet, deren Elemente nicht alle null sind. Diese Tripel werden nach folgender Relation, in proportionale Klassen eingeteilt und wahlweise als *homogene* Punkt- oder Geraden-Koordinaten bezeichnet:

$$[x, y, z] := \{(\lambda x, \lambda y, \lambda z) \in K^3 : \lambda \neq 0,\ \lambda \in K\} \backslash \{(0,0,0)\} \tag{2.17}$$

Transformationen

Für die weiter unten folgenden Betrachtungen ist es notwendig, eine Transformation zwischen Koordinaten in affinen und projektiven Ebenen zu definieren. Die angegebene Definition stellt nur eine unter vielen Möglichkeiten dar:

> Affine Koordinaten erhalten als dritte Koordinate (*Z*-Koordinate) eine 1, wenn sie in die projektive Ebene transformiert werden und umgekehrt werden die ersten beiden Koordinaten auf die dritte Koordinate normiert, falls diese ungleich null ist. In dem Fall, dass die dritte Koordinate null ist, liegt ein *uneigentlicher* Punkt vor, für den es keine Entsprechung in der affinen Ebene gibt. Ein *uneigentlicher* Punkt kann jedoch einer Richtung zugeordnet werden, wie in der obigen Konstruktion aus der affinen Ebene beschrieben.

Die Transformation ist in folgender Übersicht wiedergegeben:

Projektive Ebene		Affine Ebene und uneigentliche Punkte
$[a, b, c]$	$\rightarrow$	$\begin{cases} (a/c, b/c) \in A^2 & \text{falls } c \neq 0 \\ [a, b] \in P^1 & \text{falls } c = 0 \end{cases}$
$[x, y, 1]$	$\leftarrow$	$(x, y) \in A^2$
$[A, B, 0]$	$\leftarrow$	$[A, B] \in P^1$

> A^2 bezeichnet die affine Ebene und P^1 die Richtungen in der projektiven Ebene. Diese Richtungen, bei denen die Z-Koordinate null ist, können den *uneigentlichen* Punkten zugeordnet werden.

Tabelle 2.1: Transformation affiner und projektiver Ebenen

Die Auflösung der *Z*-Koordinate ist nicht zwingend, es kann nach jeder der drei Variablen *X*, *Y* oder *Z* aufgelöst werden.

Der Unendlichkeitspunkt *O* einer elliptischen Kurve entspricht dem Schnittpunkt mit der uneigentlichen Geraden (Z-Koordinate 0); Er hat die Koordinate [0,1,0].

2.3 Gruppenoperation elliptischer Kurven

Ausgehend von den Punkten einer elliptischen Kurve wird nun eine Gruppenoperation ⊞ definiert. Wenn keine Verwechselungsgefahr zwischen Operationen im GF(2^k) besteht, wird im Folgenden statt ⊞ einfach + verwendet.
Die Gruppenoperation ⊞ definiert die Addition zweier Punkte einer elliptischen Kurve *E*/*K*:

$$E = \{(x,y) : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{O\} \quad x,y,a_i \in K \tag{2.16}$$

Hierfür ist es notwendig, eine weitere Eigenschaft der elliptischen Kurven zu fordern und zwar die Nichtexistenz von *Singularitäten*.
Ein Punkt einer elliptischen Kurve heißt **singulär**, falls die Tangente (Steigungskurve) für diesen Punkt nicht eindeutig ist.

Definition: Die **Tangente** einer ebenen Kurve *F* mit $F(x, y) = 0$ lautet:

$$F_x(x - x_0) + F_y(y - y_0) = 0 \tag{2.18}$$

wobei F_x und F_y die partiellen Ableitungen von *F* bzgl. *x* und *y* sind:

$$F_x = \frac{\partial F(x,y)}{x} \quad F_y = \frac{\partial F(x,y)}{y} \tag{2.19}$$

Für eine elliptische Kurve *E* in Normalform und den Punkt (x_0, y_0) bedeutet dies:

$$\frac{\partial E}{\partial X}(x_0, y_0) = \frac{\partial E}{\partial Y}(x_0, y_0) = 0 \tag{2.20}$$

Die Ableitungen der Kurve nach beiden Variablen müssen 0 an diesem Punkt ergeben. Singularitäten können einen isolierten Punkt, einen Punkt durch den die Kurve mehrmals geht, oder aber sonstige Komplikationen bedeuten. Die Arten von singulären Punkten sollen hier nicht weiter betrachtet werden, da hier auch nur deren Nichtexistenz verlangt wird. Details findet man in [DTV-82, ZEI96].
Es ist jedoch wichtig zu wissen, warum Singularitäten unerwünscht sind. Nun das hängt mit der Tangente der elliptischen Kurve zusammen: Bei Singularitäten kann sie nicht eindeutig bzw. gar nicht bestimmt werden, bei der Gruppenoperation wird jedoch von der Existenz der Tangente an jeder *beliebigen* Stelle der elliptischen Kurve Gebrauch gemacht. Um dies garantieren zu können, wird die Abwesenheit von Singularitäten gefordert.

Unter der Voraussetzung, dass die Kurven in der projektiven Ebene definiert sind und keine gemeinsamen Komponenten haben, kann mittels dem Theorem von Bézout (vgl. [SIL92]; [ZEI96], S. 854 f.) gezeigt werden, dass eine Gerade und eine elliptische Kurve stets drei Schnittpunkte haben.
Die Schnittpunkte müssen allerdings nicht alle verschieden sein, lediglich die Summe der *Multiplizitäten* ist stets drei. Mit dem Begriff der *Multiplizität* können mehrfachen Schnittstellen deren Häufigkeiten zugeordnet werden: Eine doppelte Schnittstelle hat die Multiplizität zwei, usw.

Der Begriff **Multiplizität** kann anhand eines Beispiels verdeutlicht werden:

Das Polynom $f(x) = x^3 - 4x^2 + 5x - 2$ hat die folgende Nullstellen über den reellen Zahlen: 1,1,2. Das Polynom ist also gleich dem Produkt: $(x-1) \cdot (x-1) \cdot (x-2)$.
Nullstelle $x = 1$ hat Multiplizität 2 und die Nullstelle $x = 2$ hat eine Multiplizität von 1.

Ausgehend von Bézouts Satz kann nun die Gruppenoperation $\boxplus$ auf elliptischen Kurven beschrieben werden:

Der Operator $\circ$ bezeichne den dritten Schnittpunkt auf einer singularitätenfreien elliptischen Kurve E in nicht-homogenen Koordinaten. So wird die **Addition** zweier Punkte P und Q wie folgt definiert:

$$P \boxplus Q = O \circ (P \circ Q) \quad P, Q \in E/K \tag{2.21}$$

wobei O das neutrale Element der Addition ist.

Das Verfahren kann im Körper der reellen Zahlen $\mathbb{R}$ veranschaulicht werden: In den folgenden Abbildungen sind zwei unterschiedliche elliptische Kurven über den reellen Zahlen verwendet, um die Gruppenoperation zu veranschaulichen.

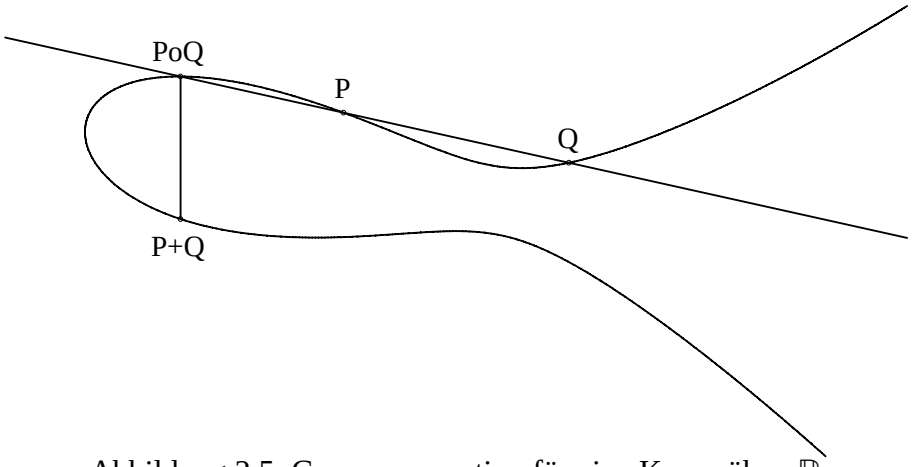

Abbildung 2.5: Gruppenoperation für eine Kurve über $\mathbb{R}$

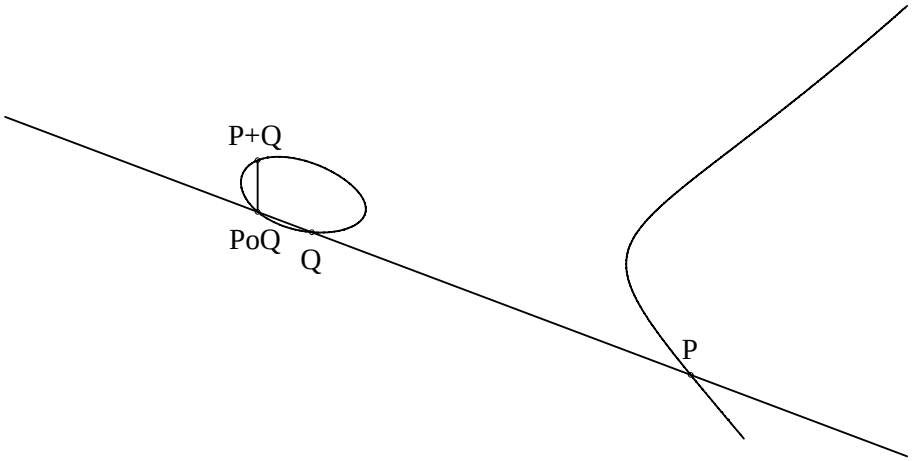

Abbildung 2.6: Gruppenoperation für eine Kurve über $\mathbb{R}$

Bei obigen Abbildungen (2.5 und 2.6) wurden jeweils zwei Punkte P und Q gewählt, auf diese Punkte der Schnittoperator $\circ$ zweifach angewendet und die Punkte $P \circ Q$ und $P \boxplus Q$ eingezeichnet.

An dieser Stelle sei noch einmal darauf hingewiesen, wozu die Gruppenoperation benötigt wird: Mit ihrer Hilfe soll die Skalarmultiplikation ausgeführt werden. Die genaue Abbildung der Skalarmultiplikation auf die Gruppenoperation und der Einsatz der Subtraktion erfolgt in Kapitel 2.7.
Bereits vorangestellt sei, dass hierfür sowohl die Addition als auch die Subtraktion von Punkten verwendet werden kann. Die Addition wurde bereits beschrieben, es folgt nun eine kurze Beschreibung der Subtraktion bzw. die Konstruktion negativer Punkte. Addiert man zu einem Punkt sein Negativ oder additives Inverses, so erhält man das neutrale Element der Addition, also den Unendlichkeitspunkt O.

Die Neutralität von O ergibt sich durch das Konstruktionsverfahren zu: $-P = P \circ S,\ S = O \circ O.$
Die Konstruktion der **inversen Elemente** wird am anschaulichsten durch die Addition von einem Punkt P und seinem Inversen ($-P$) gezeigt:

$$-P = P \circ (O \circ O) \tag{2.22}$$

$$P \boxplus (-P) = O \circ (P \circ (-P)) = O \circ S = O. \tag{2.23}$$

Es kann gezeigt werden, dass die Gruppenoperation $\boxplus$ abelsch bzw. kommutativ ist. Der Beweis der Assoziativität ist kompliziert, die Beweise findet man beispielsweise in [SIL86] und [KOB98].
Hiermit ist die Gruppenoperation von der Konstruktion aus gesehen ausreichend beschrieben, was noch fehlt ist eine Abbildung der Gruppenoperation auf Koordinaten. Die Formeln für Koordinaten werden im folgenden Abschnitt gegeben.

Explizite Formeln für die Gruppenoperationen

Bisher wurde die Gruppenoperation ($\boxplus$) nur anhand von Schnittpunkten, mit dem Schnittoperator $\circ$, behandelt. Im Folgenden sollen nun Formeln für die Darstellung von Punkten in Koordinatenform angegeben werden. D. h. der zuvor definierte Schnittoperator $\circ$ muss in Koordinaten aufgelöst werden. Hierzu werden die Elemente Geraden und Punkte in Gleichungsform angegeben. Für einen Punkt sind dies zwei Konstanten und für eine Gerade eine lineare Gleichung. Die Angabe erfolgt in affinen Koordinaten, wodurch eine Fallunterscheidung bei dem Schnittverhalten mit dem Unendlichkeitspunkt auftritt.

Negativer Punkt

Sei $P_0 = (x_0, y_0) \in E$ als ein beliebiger Punkt der elliptischen Kurve E gegeben. Der Punkt $-P_0$ ergibt sich nach obiger Definition als dritter Schnittpunkt der Geraden L durch P_0 und O mit der elliptischen Kurve. Diese Gerade L verläuft parallel zur Y-Achse und ist wie folgt gegeben:

$$L : x = x_0 \tag{2.24}$$

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.25}$$

Setzt man (2.24) in die Kurvengleichung von E (2.25) ein, so ergibt sich ein quadratisches Polynom (2.26).

(2.24) in (2.25) $$y^2 + a_1x_0y + a_3y = x_0^3 + a_2x_0^2 + a_4x_0 + a_6 \tag{2.26}$$

Das allgemeine quadratische Polynom (2.27) hat zwei Wurzeln, sie seien mit $y_0, \acute{y}_0$ bezeichnet und es gelte $-P_0 = (x_0, \acute{y}_0)$.

$$c(y - y_0)(y - \acute{y}_0) = c(y^2 - (\acute{y}_0 + y_0)y + y_0\acute{y}_0) \tag{2.27}$$

Ein Koeffizientenvergleich des allgemeinen Polynoms mit (2.26) für y^2 ergibt $c = 1$ und der Koeffizientenvergleich mit y ergibt $\acute{y}_0 = -y_0 - a_1x_0 - a_3$ somit ergibt sich für den Schnittpunkt:

$$-P_0 = (x_0, -y_0 - a_1x_0 - a_3) \tag{2.28}$$

Addition zweier Punkte

Bei der Addition zweier Punkte wird zunächst geprüft, ob sie additive Inverse sind. Liegt dieser Fall vor, so ist das Ergebnis der Unendlichkeitspunkt O. Im Folgenden wird hier davon ausgegangen, dass die Punkte nicht gegenseitig invers zueinander sind, d.h. der Ergebnispunkt der Addition sei nicht O.
L sei die Gerade durch die Punkte P_1 und P_2. Falls die Punkte gleich sind, so wird als Gerade die Tangente eines der Punkte zur elliptischen Kurve genommen.

$$L : y = \lambda x + v \tag{2.29}$$

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.30}$$

Bildet man nun die Schnittmenge der Geraden L mit der elliptischen Kurve E, ergibt sich ein Polynom dritten Grades mit drei Wurzeln $x_i \ (i = 1, 2, 3)$, wobei die dritte Wurzel den dritten Schnittpunkt mit L darstellen soll.
Das zunächst unbekannte Polynom dritten Grades sei:

$$\begin{aligned} &c(x - x_1)(x - x_2)(x - x_3) \\ &= c(x^3 - (x_1 + x_2 + x_3)x^2 + (x_1x_2 + x_1x_3 + x_2x_3)x - x_1x_2x_3) \end{aligned} \tag{2.31}$$

Die Summe der drei Schnittpunkte ergibt nach dem Konstruktionssatz der Gruppenoperation den Unendlichkeitspunkt O bzw. das neutrale Element:

$$P_1 \boxplus P_2 \boxplus P_3 = O \tag{2.32}$$

Die Schnittmenge der Geraden mit der Kurve wird durch Gleichsetzen der beiden Kurvengleichungen (2.29) und (2.30) vorgenommen, dies führt zu folgender Gleichung:

$$(\lambda x+v)^2+a_1x(\lambda x+v)+a_3(\lambda x+v)=x^3+a_2x^2+a_4x+a_6 \qquad \text{(2.29) in (2.30)}$$

$$\Leftrightarrow -x^3+x^2(\lambda^2+a_1\lambda-a_2)+x(2\lambda v+a_1v+a_3\lambda-a_4)+v^2+a_3v-a_6=0 \qquad (2.33)$$

Diese Schnittmenge vergleicht man mit dem allgemeinen Polynom (2.31) und erhält durch Koeffizientenvergleich:

$c=-1$ und für x^2 folgt

$$\lambda^2+a_1\lambda-a_2=x_1+x_2+x_3 \qquad (2.34)$$

Der Ausdruck von x_3 kann nun in die Geradengleichung L (2.31) eingesetzt werden und es ergibt sich $\hat{y}_3$ zu:

$$\hat{y}_3=\lambda x_3+v \qquad (2.35)$$

Der Punkt $(x_3,\hat{y}_3)$ entspricht dem Schnittoperator. Um den resultierenden dritten Punkt der Gruppenoperation

$$P_1 \boxplus P_2 = P_3$$

zu erhalten, muss dieser Punkt noch invertiert werden. Dies geschieht mit der Formel aus dem vorigen Abschnitt.

(*Invertierung*)

$$\begin{aligned} P_3 &= -(x_3,\hat{y}_3)=(x_3,-\hat{y}_3-a_1x_3-a_3)\\ &=(x_3,-\lambda x_3-v-a_1x_3-a_3)\\ &=(x_3,-(\lambda+a_1)x_3-v-a_3) \end{aligned} \qquad (2.36)$$

Es gilt somit:

$$x_3=\lambda^2+a_1\lambda-a_2-x_1-x_2$$

$$y_3=-(\lambda+a_1)x_3-v-a_3 \qquad (2.37)$$

Es verbleibt noch, die Parameter für die Gerade durch die Punkte P_1 und P_2, anzugeben. Es wird hierbei zwischen der Tangente im Fall gleicher X-Koordinaten unterschieden:

$$\lambda=\frac{y_2-y_1}{x_2-x_1} \qquad v=\frac{y_1x_2-y_2x_1}{x_2-x_1} \qquad \text{falls } x_1\neq x_2$$

$$\lambda=\frac{3x_1^2+2a_2x_1+a_4-a_1y_1}{2y_1+a_1x_1+a_3} \qquad v=y_1-\lambda x_1 \qquad \text{falls } x_1=x_2$$

Nun sind alle notwendigen Formeln für die Berechnung von Koordinaten aufgeführt. Ausgegangen von den elliptischen Kurven, als eine Punktmenge, die eine Gleichung erfüllen, inklusive eines Unendlichkeitspunktes *O*.
Die Kurve ohne den Unendlichkeitspunkt *O* kann in affinen Ebenen dargestellt werden, mit der Erweiterung auf projektive Ebenen ist eine Beschreibung inklusive des Unendlichkeitspunktes möglich. In der projektiven Ebene stehen ebenfalls Koordinaten zur Verfügung, diese homogenen Koordinaten sind jedoch nicht eindeutig. Das verwendete projektive Modell wird im Kapitel 2.6, S. 34 ff., näher beschrieben.

2.4 Vereinfachung der Normalform

Die allgemeine Form einer elliptischen Kurve kann unter bestimmten Einschränkungen vereinfacht werden. Vereinfachungen entstehen durch strukturerhaltende Substitutionen (Isomorphismen), wobei die Faktoren und Formen der Substitutionen nur für bestimmte zugrunde liegende Körper möglich ist. Hieraus resultieren Vereinfachungen für Körper der Charakteristiken zwei, drei und für Körper anderer Charakteristik. Eine tiefer gehende Abhandlung findet man in [SIL86].

Im Folgenden werden die wesentlichen Vereinfachungen für die drei genannten Fälle angegeben und in verkürzter Form dargestellt.

J-Invariante und Diskriminante

J oder *J(E)* ist die sogenannte **J-Invariante** und beschreibt einen Teil der Gleichung, die unter allen isomorphen Abbildungen invariant ist. Dies ist jedoch nicht ihre einzige Bedeutung. Sie wird zudem herangezogen, um ein wichtiges Kriterium für die Kryptographie zu beschreiben. Kurven mit J-Invariante null ($J(E) = 0$) werden als **supersinguläre** Kurven bezeichnet.
Für diese Kurvenart ist die Berechnung des diskreten Logarithmus einfacher (vgl. [MOV93]), sie sind somit für die Verwendung in Kryptographischen Verfahren unsicherer und sollten daher nicht verwendet werden. Im Folgenden wird **stets** von elliptischen Kurven ausgegangen, die nicht supersingulär sind.

Auf der Basis dieser Invarianten können Aussagen über alle möglichen Umformungen der Gleichung für elliptische Kurven getroffen werden. Die Formel für die Invariante ist jedoch umfangreich und für diese Arbeit nicht weiter von Bedeutung, weshalb die Formeln für die J-Invariante lediglich im Anhang wiedergegeben sind.

Neben der J-Invarianten existiert die **Diskriminante** (Formelzeichen Δ) als eine weitere Charakterisierungsgröße für elliptische Kurven, sie wird mit Δ bezeichnet. Die Diskriminante darf nicht null werden, in diesem Fall liegt eine **singuläre** Kurve vor und bei singulären Kurven ist die Gruppenoperation nicht definiert (vgl. Kap. 2.3). Aus diesem Grund ist in den folgenden Kurvengleichungen ebenfalls die Diskriminante angegeben. Eine allgemeine Definition der Diskriminante findet man in [ZEI96] (S. 236).

Körper der Charakteristik 2

Ausgehend von der allgemeinen Form einer elliptischen Kurve *E/K*:

$$E = \{(x,y) : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{O\} \quad x,y,a_i \in K \tag{2.16}$$

Kann für elliptische Kurven *E* über Körpern der Charakteristik 2, folgende Substitution vorgenommen werden:

$$\begin{aligned} &(x,y) \vdash (a_1^2x - a_3/a_1, a_1^3y + (3a_3^2 + a_1^2a_4)/a_1^3) \\ &E_{(x,y)} \vdash y^2 + xy = x^3 + b_2x^2 + b_6 \end{aligned} \tag{2.38}$$

Die Zusammenfassung der neuen Koeffizienten vor den Variablen erfolgt mit gleicher Index-Systematik, wie bei der Ausgangsgleichung. Um auf die Abweichung gegenüber der allgemeinen Form hinzuweisen, wurde der Buchstabe *b* statt *a* verwendet. Um diesen Teil der Arbeit möglichst knapp halten zu können, wurde die detaillierte Übersicht über die Transformationen in den Anhang der Arbeit verschoben.
Die allgemeine Form für elliptische Kurven über GF(2^k) lautet:

$$E = \{(x,y) : y^2 + xy = x^3 + ax^2 + b\} \cup \{O\} \quad x,y,a,b \in K\text{=GF}(2^k) \text{ mit b} \neq 0 \qquad (2.39)$$

Die Diskriminante, welche nicht 0 sein darf, ist b: $\Delta = b$

Körper der Charakteristik ungleich 2 und 3

Dieser Fall wurde bereits in der Diplomarbeit von Bohnsack [BOH97] behandelt und wird aus diesem Grund hier nur kurz erwähnt. Das Resultat lautet:

$$y^2 = x^3 + b_4 x + b_6 \quad \Delta = -16(4b_4^3 + 27b_6^2) \qquad (2.40)$$

Die Substitution ausgehend von der Normalform lautet:

$$y \vdash y - (a_1 x + a_3)/2$$

und liefert aus dem Original:

$$y^2 = x^3 + x^2(a_1^2/4 + a_2) + x(a_4 + \tfrac{a_1 a_3}{2}) + a_3^2/4 + a_6$$

Fasst man die Koeffizienten mit neuer Bezeichnung *c* zusammen, so kann ferner der quadratische *x*-Teil mit folgender Substitution aufgelöst werden:

Zusammenfassung der Koeffizienten:

$$y^2 = x^3 + c_2 x^2 + c_4 x + c_6$$

Substitution:

$$x \vdash x - c_2/3$$

Nach Umformung und erneuter Zusammenfassung der Koeffizienten zu *b*, ist das gewünschte Ziel erreicht:

$$y^2 = x^3 + b_4 x + b_6 \quad \Delta = -16(4b_4^3 + 27b_6^2) \qquad (2.40)$$

2.5 Auswirkungen des GF(2^k) auf die Gruppenoperation

Die Wahl eines bestimmten Körpers zieht Vereinfachungen für die Normalform der elliptischen Kurven (vgl. S. 20, 2.16) nach sich. Die Vereinfachungen der Normalform wurde im vorige Abschnitt (Kap. 2.4) für die Charakteristiken zwei und größer drei, vorgestellt.
Vereinfachungen in der Normalform drücken sich in der Reduzierung von Termen aus: So enthalten die Normalformen (2.39) und (2.40) weniger Ausdrücke als die allgemeine Variante (2.16). Wichtiger als die Reduzierung von Termen für die Punktoperationen, sind Reduzierungen für die Gruppenoperation.
Mit Hinblick auf den zu erstellenden Akzelerator, der die Gruppenoperation ausführen soll, ist es wichtig, die Zahl der Operationen im Körper möglichst einfach zu halten und deren Anzahl so stark wie möglich zu reduzieren.

Für den Körper GF(2^k) kann die Anzahl der Multiplikationen und Additionen verringert werden. Dies ergibt sich durch die Belegung der Konstanten in der Normalform für elliptische Kurven. Betrachtet man die Normalform einer elliptischen Kurve (2.16) und vergleicht diese mit der Normalform für den Körper GF(2^k):

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.16}$$

$$y^2 + xy = x^3 + ax^2 + b \tag{2.39}$$

So stellt man fest, dass die Koeffizienten für (2.16) wie folgt belegt werden können:

$$a_1 = 1,\ a_3 = 0,\ a_4 = 0.$$

Offensichtlich fallen hier Additionen aufgrund der Nullbelegungen (a_3, a_4) weg und Multiplikationen durch a_1. Die Gruppenoperation $\boxplus$ kann wie folgt berechnet werden, wobei die Punktindizierung aus obiger Beschreibung der Gruppenoperation übernommen wurde. Für die elliptische Kurve E und die Punkte P_1 und P_2 gilt:

$$P_1 + P_2 = P_3 \quad \text{mit} \quad P_1 = (x_1, y_1),\ P_2 = (x_2, y_2),\ P_3 = (x_3, y_3),\ P_1, P_2, P_3 \in E$$

Auf die Koordinaten angewendet ergibt dies:

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

$$x_3 = \lambda^2 + \lambda + a + x_1 + x_2$$

$$y_3 = (x_3 + x_1)\lambda + x_3 + y_1$$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \qquad \text{falls } x_1 \neq x_2$$

$$\lambda = x_1 + \frac{y_1}{x_1} \qquad \text{falls } x_1 = x_2$$

(2.39a)

Bei der neuen X-Koordinate entfallen zwei Additionen, wenn die Additionspunkte identisch ($x_1 = x_2$) sind, da die Summe der beiden X-Koordinaten null ergibt. Des Weiteren fällt auf, dass die Zwischenvariable v entfallen konnte.

Die Negation eines Punktes ergibt sich aus (2.28) mit (2.39) zu:

(allgemein) $$-P_0 = (x_0, -y_0 - a_1 x_0 - a_3) \tag{2.28}$$

(für GF(2^k)) $$-P_0(x_0, y_0) = (x_0, x_0 + y_0) \tag{2.41}$$

Diese Vereinfachungen haben insbesondere auch Auswirkungen auf angestrebte Hardware-Implementierungen, diese Vereinfachungen werden in den folgenden Kapiteln (4 und 5) behandelt.

Die Einsparungen im Vergleich zur allgemeinen Normalform sind in folgender Übersicht wiedergegeben, hierbei wurden Multiplikationen mit den Konstanten 2 und 3 als Additionen gezählt:

Fall		**Divisionen**	**Multiplikationen**	**Additionen/ Subtraktionen**
allgemeine Normalform	$x_1 \neq x_2$	1	7	10
	$x_1 = x_2$	1	7	17
GF(2) Normalform	$x_1 \neq x_2$	1	2	9
	$x_1 = x_2$	1	2	8
Ersparnis bei GF(2)	$x_1 \neq x_2$	0	5	1
	$x_1 = x_2$	0	5	9

Tabelle 2.2: Übersicht der Einsparungen allgemeine Normalform und GF(2)

Beispiel für elliptische Kurven in GF(2^4)

Nun sollte das Beschriebene kurz anhand eines Beispiels (aus [ANS98], S. 19, entnommen) veranschaulicht werden. Hier wird der Körper GF(2^4) und das irreduzible Polynom $x^4 + x + 1$ gewählt. Für den GF(2^4) existieren auch weitere irreduzible Polynome, wie beispielsweise $x^4 + x^3 + 1$ oder $x^4 + x^3 + x^2 + x + 1$.

Als *Primitivwurzel* soll hier $a = x$ verwendet werden. Dieses Element wird *Primitivwurzel, Erzeugendes Element* oder auch *Generator* genannt, da seine Potenzen sämtliche Elemente des Körpers aufzählen, lediglich die Null ist ausgenommen. Mit andern Worten, es werden alle Elemente der zyklischen Gruppe des Körpers aufgezählt (vgl. [FOR96], S. 62f):

$$GF(2^4) = \{x^k | 0 \leq k < 15, k \in \mathbb{N}\} \cup \{0\} \tag{2.42}$$

Um die Notation zu verkürzen, wird oftmals eine Tupel-Darstellung der Zahlen bzw. Elemente des GF(2^k) verwendet. Hierbei werden nur die Koeffizienten der Polynome verwendet:

So kann $x^3 + x + 1$ eindeutig durch das Tupel (1,0,1,1) dargestellt werden.

Bei dieser Notation muss jedoch eindeutig angegeben werden, an welcher Stelle sich die höheren Grade des Polynoms befinden. Hier beginnt die Darstellung von links nach rechts innerhalb der Tupel mit den hohen Potenzen:

$$z = z_3x^3 + z_2x^2 + z_1x + z_0 = \sum_{i=0}^{k-1} z_i x^i = (z_3, z_2, z_1, z_0) \tag{2.43}$$

Mit der Primitivwurzel $\alpha = x$, werden folgende Elemente erzeugt:

$\alpha^0 = (0,0,0,1)$	$\alpha^4 = (0,0,1,1)$	$\alpha^8 = (0,1,0,1)$	$\alpha^{12} = (1,1,1,1)$
$\alpha^1 = (0,0,1,0)$	$\alpha^5 = (0,1,1,0)$	$\alpha^9 = (1,0,1,0)$	$\alpha^{13} = (1,1,0,1)$
$\alpha^2 = (0,1,0,0)$	$\alpha^6 = (1,1,0,0)$	$\alpha^{10} = (0,1,1,1)$	$\alpha^{14} = (1,0,0,1)$
$\alpha^3 = (1,0,0,0)$	$\alpha^7 = (1,0,1,1)$	$\alpha^{11} = (1,1,1,0)$	$\alpha^{15} = \alpha^0 = (0,0,0,1)$

Nun sind die Elemente des GF(2^4) für das oben genannte irreduzible Polynom vorhanden. Es bedarf nur noch einer elliptischen Kurve, um das Beispiel zu vervollständigen. Als elliptische Kurve diene folgende Gleichung:

$$y^2 + xy = x^3 + \alpha^4 x^2 + 1, \quad \text{wobei } 1 = \alpha^0 \text{ eingesetzt wurde.} \tag{2.44}$$

Nun soll einmal ermittelt werden, welche Punkte überhaupt auf der Kurve liegen.
Um in dieser Liste nicht mit den oben genannten relativ großen Tupeln arbeiten zu müssen, werden die Elemente der Einfachheit halber nummeriert und wie folgt auf die natürlichen Zahlen abgebildet:

$$\pi : GF(2^k) \rightarrow \mathbb{N}$$

$$\pi(x) = \begin{cases} i+1 & \text{für } x = \alpha^i \\ 0 & \text{für } x = 0 \end{cases} \tag{2.45}$$

Die Funktion bildet auf den Exponenten ab und berechnet den Logarithmus zur Primitivwurzel.

Die Gleichung der elliptischen Kurve hat folgende Lösungen beinhaltet:

$$\left\{ \begin{array}{c} (0,1),(1,7),(1,14),(4,9),(4,14),(6,4),(6,12),(7,9),(7,15),(10,11),(10,14),(11,2), \\ (11,9),(13,0),(13,13) \end{array} \right\}$$

Die Lösung enthält 15 Elemente. Die Anzahl der Punkte, inklusive des unendlichen fernen Punktes *O*, wird auch *Ordnung* der elliptischen Kurve genannt.

Obige Punktliste repräsentiert nun die Beispielkurve. Problematisch ist allerdings, wie man sich die Kurve veranschaulichen soll und die geometrischen Eigenschaften der Gruppenoperation an ihr demonstrieren kann. Bei Kurven über den reellen Zahlen war dies ja einfach möglich (Vgl. Abb. 2.5 und 2.6 auf S. 25).
Bei endlichen Körpern ist dies so nicht möglich. Um dennoch einen Eindruck zu erhalten, werden obige Elemente nach ihrer Potenzsumme mit der Basis 2 bewertet, dies entspricht der Interpretation der Tupel als gewöhnliche Binärzahlen:

$$z = z_3 2^3 + z_2 2^2 + z_1 2 + z_0 = \sum_{i=0}^{k-1} z_i 2^i \tag{2.46}$$

Für die Potenzen der Primitivwurzel $a^0, ..., a^{14}$ ergibt sich folgende Liste: {1, 2, 4, 8, 3, 6, 12, 11, 5, 10, 7, 14, 15, 13, 9}, wobei die Null ausgenommen wurde. Auf die Punkte der elliptischen Kurve angewendet entstehen folgende Koordinaten, wobei die Reihenfolge mit obiger Punktliste übereinstimmt:
(0, 1), (1, 12), (1, 13), (8, 5), (8, 13), (6, 8), (6, 14), (12, 5), (12, 9), (10, 7), (10, 13), (7, 2), (7, 5), (15, 0), (15, 15).

Diese Zahlen wurden in den folgenden Darstellungen als Koordinaten verwendet. Hier sollte noch erwähnt werden, dass diese Abbildung eine willkürliche Festlegung ist und dass sich mit anderen Abbildungen ganz andere Bilder ergeben.

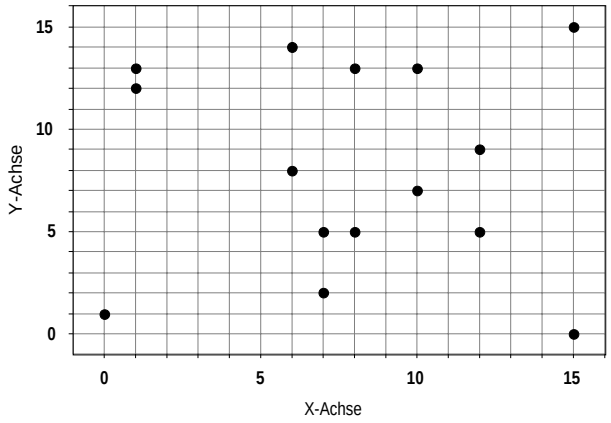

Abbildung 2.6: Versuch, eine elliptische Kurve über GF(2^4) graphisch darzustellen

Abb. 2.6 zeigt die oben behandelte Beispielkurve. Man erkennt deutlich die inversen Punkte, da sie sich auf der gleichen x-Koordinate befinden. Eine Ähnlichkeit mit den elliptischen Kurven über den reellen oder rationalen Zahlen ist jedoch praktisch nicht erkennbar.

Das Einzige, was für die Anschauung bleibt, ist dass die Punkte die algebraische Gleichung (2.39) erfüllen. Eine grafische Deutung der Gruppenoperation ist hier nicht mehr möglich, wenn man davon absieht, dass maximal zwei Punkte auf einem X-Achsenwert sind.

2.6 Projektives Modell

Eine wesentliche Beschleunigung von Operationen auf elliptischen Kurven besteht in der Verwendung von projektiven Koordinaten anstatt von affinen. Hier werden projektive Koordinaten verwendet und für die Gruppenoperationen werden Fallunterscheidungen vorgenommen.

Affine Punkte werden in projektiven Koordinaten wiedergegeben. Wie im Folgenden gezeigt wird, sind die Koordinaten nicht mehr eindeutig, d. h. es existieren mehrere Darstellungen für einen bestimmten Punkt auf der Kurve.
Die Einsparung von Berechnungsresourcen tritt bei der Berechnung von den multiplikativen Inversen auf: Die Invertierung wird erst benötigt, wenn man von den projektiven Koordinaten zu affinen wechselt. Dieser Wechsel findet, sofern er notwendig ist, erst am Ende einer

Skalarmultiplikation statt. Die Notwendigkeit einer Rücktransformation hängt von der jeweiligen Anwendung ab und kann nicht pauschal beantwortet werden.

Es hat sich herausgestellt, dass die Bestimmung des multiplikativen Inversen den zeitintensivsten Engpass verursacht. Dies ist ein wesentliches Resultat aus der Diplomarbeit von Bohnsack. Gezeigt werden kann dies mit der Betrachtung von Aufwandsabschätzungen für die Berechnung des Inversen.

Der erweiterte euklidische Algorithmus wird maximal mit der Gradzahl des Erweiterungskörpers aufgerufen und bei jedem Aufruf wird eine Division (mit Rest), eine Multiplikation und eine Addition benötigt. Diese Operationen benötigen wieder die gleiche Zahl an Bearbeitungsschritten, ihr Aufwand entspricht also der Zahlengröße. Eine ausführliche Analyse des euklidischen Algorithmus findet man beispielsweise in [KNU98], Kap. 4.5.3, eine algorithmische Fassung befindet sich im Anhang dieser Arbeit.

Die Erleichterung bei der Berechnung von Gruppenoperationen auf elliptischen Kurven, hervorgerufen durch die Einsparungen an Divisionen, erkauft man sich mit entsprechend komplizierten Additionsformeln. Die Zahl der Berechnungen liegt jedoch weit unter der für den erweiterten euklidischen Algorithmus.
Im Folgenden sind die Formeln für die Addition nur informativ angegeben, für den Hintergrund muss hier auf weiterführende Literatur, wie etwa dem zweiten Kapitel von [HUS87], verwiesen werden.

Das projektive Modell benötigt drei Koordinaten (2.17), die Transformationen müssen jedoch nicht direkt in ein Modell mit den Koordinaten X, Y und Z erfolgen, sondern es können auch kompliziertere Varianten verwendet werden. Genau einer dieser Fälle führt zu einem sehr effizienten System an Gleichungen für die Gruppenoperation der elliptischen Kurven.
Das in [D7-98] vorgeschlagene projektive Modell ist Ergebnis von Untersuchungen (vgl. [CC87]), von denen (2.47) den geringsten gesamten Aufwand an Multiplikationen und Additionen, im GF(2^k) aufweist. Dieses komplexere Modell wird im Folgenden beschrieben.

Das einfache oder direkte projektive Modell hat Koordinaten der Form:

$$[x, y, z] := \{(\lambda x, \lambda y, \lambda z) \in K^3 : \lambda \neq 0,\ \lambda \in K\} \backslash \{(0, 0, 0)\} \tag{2.17}$$

Das komplexere projektiven Modell aus [D7-98] hat folgende Form:

$$(\lambda^2 X, \lambda^3 Y, \lambda Z) \text{ mit } \lambda \in GF(2^k) \backslash \{0\} \tag{2.47}$$

Der unendlich ferne Punkt O hat die Koordinaten:

$$(\lambda^2, \lambda^3, 0) \text{ mit } \lambda \neq 0 \tag{2.48}$$

Die Umwandlung in affine Koordinaten ergibt sich zu:

$$(X, Y, Z) \vdash \left(\frac{X}{Z^2}, \frac{X}{Z^3}\right) \Leftrightarrow (x, y) \quad \text{für Z} \neq 0 \tag{2.49}$$

Und umgekehrt sind die projektiven Koordinaten eines affinen Punktes:

$$(x,y) \vdash (x,y,1) \quad \text{bzw.} \quad X \leftarrow x,\ Y \leftarrow y,\ Z \leftarrow 1 \tag{2.50}$$

Wie man in der Umwandlungsgleichung (2.50) sieht, existieren jetzt drei Koordinaten *X, Y* und *Z*. Man kann diese Transformation auch als rationales Modell mit gemeinsamen Nenner betrachten. Das rationale Modell wird später erklärt. Es beschäftigt sich mit getrennter Zähler- und Nennerdarstellung, wodurch die Division entschärft wird.

Zunächst sollen die Formeln für die Gruppenoperation (⊞) unter Verwendung des projektiven Modells angegeben werden. Für die Punktaddition ungleicher Punkte und die Verdopplung eines Punktes sind die Formeln aus [D7-98] im Folgenden wiedergegeben:

Punktaddition

Für eine elliptische Kurve E und die Punkte $P_0, P_1, P_2 \in E$:

$$P_0 = (X_0, Y_0, Z_0), P_1 = (X_1, Y_1, Z_1), P_2 = (X_2, Y_2, Z_2)$$

lauten die projektiven Formeln der Gruppenoperation (⊞) nach [D7-98]:

Für eine **Punktverdopplung** (P_1+P_1 = P_2) ergibt sich:

$$P_2 = 2(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

$$\begin{aligned} c &= b^{2^{k-2}} \\ Z_2 &= X_1 Z_1^2 \\ X_2 &= (X_1 + cZ_1^2)^4 \\ U &= Z_2 + X_1^2 + Y_1 Z_1 \\ Y_2 &= X_1^4 Z_2 + U X_2 \end{aligned} \tag{2.51}$$

Anmerkung: Zu den Formeln (2.51) und (2.52) gelangt man durch Anwendung von (2.49) auf die Formeln der Gruppenoperation (2.37). Anschließend müssen die Ausdrücke gleichnamig gemacht und vereinfacht werden. Diese algebraische Umformung ist jedoch umfangreich und aus diesem Grund hier nicht aufgeführt.

Für ungleiche Punkte bei der **Addition** (P_0+P_1 = P_2) ergibt sich:

$$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2) \ = P_2$$

$$\begin{aligned} U_0 &= X_0 Z_1^2 & W &= U_0 + U_1 & L &= Z_0 W & T &= R + Z_2 \\ S_0 &= Y_0 Z_1^3 & S_1 &= Y_1 Z_0^3 & V &= RX_1 + LY_1 & X_2 &= aZ_2^2 + TR + W^3 \\ U_1 &= X_1 Z_0^2 & R &= S_0 + S_1 & Z_2 &= LZ_1 & Y_2 &= TX_2 + VL^2 \end{aligned} \tag{2.52}$$

In den oben angegebenen Formeln sind zur besseren Übersicht verwertbare Teilergebnisse in Zwischenvariablen bzw. Teilausdrücken aufgeteilt.

Auch wenn obige Formeln recht umfangreich und undurchschaubar erscheinen, so entsprechen sie weitgehend einer einfachen Einsetzung der Transformationsgleichung in den allgemeinen Formeln für den GF(2^k). Lediglich die Aufteilung der gesamten Berechnung auf wiederverwertbare Teilergebnisse ist schwieriger nachzuvollziehen.

Der mit den Formeln verbundene Aufwand ist Gegenstand des folgenden Kapitels, dort werden Berechnungsalternativen untersucht. In diesem Kapitel wird lediglich das Gerüst der Berechnungen angegeben. Für die Implementierung in Hardware ist es erforderlich, den Berechnungsfluss im Einzelnen zu untersuchen. Hierbei können Teilberechnungen parallelisiert und Zwischenergebnisse mehrfach verwendet werden.

Nachdem das projektive Modell vorgestellt wurde und konkrete Formeln (2.51) und (2.52) vorhanden sind, lohnt es sich, dieses Berechnungsprinzip zu untersuchen. Eine ähnliche *Umgehung* der Division kann analog zu Operationen bei den rationalen Zahlen erfolgen. Dieses Modell wird im Folgenden als **rationales Modell** bezeichnet.
Hier verwendet man eine getrennte Zähler-Nenner-Darstellung der einzelnen Zahlen:

$$x = \frac{a_Z}{a_N} \qquad x, a_Z, a_N \in GF(2^k) \tag{2.53}$$

Man erhält eine mehrdeutige Darstellung von Zahlen über GF(2^k). Nun können die Rechenoperationen wie folgt behandelt werden:

Addition und Subtraktion:

$$a \pm b = \frac{a_Z}{a_N} \pm \frac{b_Z}{b_N} = \frac{a_Z b_N \pm a_N b_Z}{a_N b_N} \tag{2.54}$$

Multiplikation:

$$a \cdot b = \frac{a_Z}{a_N} \cdot \frac{b_Z}{b_N} = \frac{a_Z b_Z}{a_N b_N} \tag{2.55}$$

Division:

$$a \div b = \frac{a_Z}{a_N} \div \frac{b_Z}{b_N} = \frac{a_Z b_N}{a_N b_Z} \tag{2.56}$$

Dies hat den Vorteil, dass die Division bei keiner der obigen Operationen berechnet werden muss. Lediglich bei dem Verlassen der Zähler-Nenner-Darstellung ist eine Division erforderlich. Der Mehraufwand bei einer Verwendung dieses Systems ist in der folgenden Tabelle zusammengefasst:

Operation	**Rationales Modell**
1 Addition/ Subtraktion	3 Mult. + 1 Add.
1 Multiplikation	2 Mult.
1 Division	2 Mult.

Tabelle 2.3: Mehraufwand rationales Modell gegenüber affinen Koordinaten

Ein scheinbar erstaunliches Ergebnis ist, dass die zuvor einfachen Operationen + und - nunmehr komplizierter als die Multiplikation und Division sind.
Zu dem Mehraufwand in Tab. 2.4 kommt, dass nun der doppelte Speicherplatz pro Körperelement benötigt wird.
Ein wesentlicher Vorteil im Vergleich zu den Rechenoperationen bei den rationalen Zahlen ist, dass die Ergebnisse nicht beliebig groß werden, da hier ein endlicher Körper verwendet wird.

2.7 Skalarmultiplikation

Unter Skalarmultiplikation bei elliptischen Kurven versteht man die *k*-fache Addition eines Punktes auf der Kurve, wobei *k* eine natürliche Zahl ist:

$$k \cdot B \text{ mit } k \in \mathbb{N},\ B \in E \text{ (Punkt auf der Kurve)} \tag{2.57}$$

Die Skalarmultiplikation ist zugleich die zentrale Operation für Ver- und Entschlüsselungen. Die Anwendung eines kryptographischen Verfahrens wird exemplarisch im Kapitel 3 betrachtet.
Die Skalarmultiplikation wird auf die Punktaddition, -subtraktion und -verdopplung zurückgeführt: Die Aufteilung bei der Berechnung von $k \cdot B$ kann beispielsweise wie folgt vorgenommen werden:

1) $5 \cdot B = B + B + B + B + B$
2) $5 \cdot B = 2 \cdot (2 \cdot B) + B$
3) $7 \cdot B = 2 \cdot (2 \cdot ((2 \cdot B))) - B$

Eine detaillierte Betrachtung der Aufteilungsmöglichkeiten findet in Kapitel 4.4 statt.

An dieser Stelle soll es bei Beispielen belassen werden. Für die folgenden Betrachtungen dieser Arbeit ist es jedoch wichtig zu wissen, dass eine derartige Abbildung der Skalarmultiplikation auf die Punktaddition, -verdopplung und -subtraktion existiert.

2.8 Zusammenfassung

In Kapitel 2 wurden die mathematischen Grundlagen für das Verständnis des Akzelerators behandelt: Ziel ist es, die Skalarmultiplikation auf elliptischen Kurven über endlichen Körpern der Form GF(2^k), zu verstehen. Hinzu kam die Verwendung von homogenen Koordinaten der projektiven Ebene, in der keine parallelen Geraden existieren.
Die Arithmetik von endlichen Erweiterungskörpern, speziell der Form GF(2^k), wurde gewählt, da sie sich besonders gut bzw. einfach in Hardware realisieren lässt.
Die Division bzw. die Berechnung der multiplikativen Inversen im GF(2^k) ist extrem rechenintensiv (vgl. (2.1) und (2.3)). Um Verwechslungen und Missverständnissen vorzubeugen, sei noch einmal darauf hingewiesen, dass die "Division" und die "Division mit Rest" zwei grundlegend verschiedene Dinge sind: Bei der zweiten können Reste auftreten, während bei der Division bzw. Multiplikation mit dem multiplikativen Inversen keine Reste auftreten können.

Die projektive Ebene erlaubt die Reduzierung der benötigten Divisionen im GF(2^k), welche einen erheblichen Zeitaufwand bei der Berechnung verursachen. Lediglich bei der Rücktransformation in affine Koordinaten ist eine Division erforderlich.

3 Grundlagen zu elliptischen Kurven

In diesem Kapitel werden im Anschluss zu den Grundlagen aus Kap. 2, ergänzende Aspekte von elliptischen Kurven behandelt. Hierbei handelt es sich um Fragen zur *Kurvenordnung* der elliptischen Kurven und die Frage, wie man eine ausreichende Zahl von Testkurven mit bestimmter Qualität erhalten kann.
Die sicherheitstechnischen Aspekte der verwendeten elliptischen Kurven werden nicht vom Akzelerator behandelt, sie werden vom Systembetreuer des späteren Anwenders, beispielsweise eines Trust-Centers, abgedeckt. Diese Instanz (Trust-Center) ist für die Güte und Sicherheit der verwendeten Systemparameter, zu denen primär die elliptischen Kurven gehören, zuständig.
Im Folgenden wird auf die Berechnung der Ordnung von elliptischen Kurven eingegangen. Insbesondere ein Spezialfall, bei dem die sonst schwierige Ermittlung einfach ist, soll kurz erläutert werden. Hierfür wird im Folgenden ein Algorithmus angegeben, welcher die Kurvenordnung effizient berechnen kann.

In Kapitel 3.2 wird auf Probleme bei der Anwendung dieser Methode in praktisch relevanten Sicherheitssystemen hingewiesen. Dieser Abschnitt gehört zwar nach obigen Verweis der Trust-Center für die Sicherheit nicht zu den Kernpunkten dieser Arbeit, dennoch erscheint es geboten, diesen Hinweis aus der Praxis in diese Arbeit aufzunehmen.

Im Anschluss (Kap. 3.3) wird die Problematik der anderen Systemparameter, der eigentlichen elliptischen Kurve und des auf ihr liegenden Basispunktes, behandelt. Dort wird die Frage behandelt, wie schwer es ist, einen geeigneten Punkt auf der elliptischen Kurve zu finden.

Die **Gruppenoperation** $\boxplus$ der elliptischen Kurve E macht die eigentliche **Nutzbarkeit** der elliptischen Kurven für kryptographische Verfahren aus:
Allgemein geht es bei kryptographischen Verfahren darum, eine Information so zu kodieren, dass es schwierig ist, die Information ohne Kenntnis etwaiger Schlüssel-(Informationen) zu erhalten.
Bei elliptischen Kurven wird die Information im wesentlichen mit der **Skalarmultiplikation** kodiert. Die Skalarmultiplikation elliptischer Kurven ist auf die Addition von Punkten der elliptischen Kurve definiert. Die Information steckt im wesentlichen in dem Skalar, also der Anzahl von Additionen eines bestimmten Punktes.
Ein weiterer wichtiger Punkt für die Anwendung elliptischer Kurven ist ihre Ordnung, sie wird im folgenden Abschnitt behandelt.

3.1 Ordnung elliptischer Kurven

Die Berechnung der Ordnung einer elliptischen Kurve ist für ihre kryptographische Anwendung von großem Interesse. **Die Ordnung entspricht der Punktezahl der Kurve** (3.1). Diese Ordnung muss bestimmten Anforderungen entsprechen, um sicher mit ihr arbeiten zu können bzw. gegen spezielle Angriffe immun zu sein. An dieser Stelle sei hier auf die Arbeit von S. Hamdy [HAM98] verwiesen, dort werden derartige Kriterien ausführlich behandelt und sind dort im Kapitel 7.5 zusammenfassend aufgezählt.

Neben den sicherheitstechnischen Aspekten ist die Kurvenordnung jedoch auch aus Gründen der rechentechnischen Effizienz von Interesse, dies wird nach der allgemeinen Betrachtung der Kurvenordnung, im folgenden Kapitel (Kap. 3.2), behandelt.

Die **Ordnung** $\#(E)$ einer elliptischen Kurve E ist:

$$\#(E) = |\{(x,y) \mid y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{O\}| \tag{3.1}$$

Die **Ordnung eines Punktes** $\#(B)$ einer elliptischen Kurve E ist definiert als die Gruppenordnung der Teilkurve, auf der sich B befindet. Anschaulich betrachtet ist dies die Zahl, wie oft ein Punkt B einer elliptischen Kurve E zu sich selbst addiert werden muss, bis das Ergebnis O ist:

$$\#(B) = k \quad \text{mit} \quad k = \min\{\, i \mid i \cdot B = O\}, \quad k, i \in \mathrm{N}, B \in E \tag{3.2}$$

Die Bestimmung der Kurvenordnung für allgemeine elliptische Kurven, also nicht für Spezialfälle, ist sehr schwierig. An dieser Stelle sei auf die Dissertation von Müller [MUE95] verwiesen.
Es existiert aber ein relativ einfacher Algorithmus, um Kurven mit bestimmter Ordnung zu erzeugen bzw. deren Ordnung zu berechnen. Der Algorithmus ist Gegenstand dieses Abschnitts, da er für die Bereitstellung von ausreichendem Testmaterial in Form von Kurven und deren Ordnung für Zeit- und Verhaltensbetrachtungen des Akzelerators dient. Dieses Testmaterial ist auch für den funktionalen Test des ASICs sehr wichtig.

Theorem von Hasse

Grundlage für die Berechnung der Kurvenordnung einer elliptischen Kurve ist das Theorem von Hasse. Die Vermutung dieses Theorems stammt von Riemann, der Beweis der Vermutung wurde 1934 von Hasse veröffentlicht.
Das Theorem (3.3) besagt, dass die Ordnung einer elliptischen Kurve in einem Intervall liegt. Eine tiefergehende Herleitung des Theorems findet man in [HUS87, SIL86]. An dieser Stelle wird nur auf die Anwendung für die konkrete Bestimmung der Kurvenordnung eingegangen:

$$|\#E(F_q) - q - 1| \le 2\sqrt{q} \tag{3.3}$$

F_q ist der Körper, wobei $q = p^m$ gilt und p prim sei

$\#E(F_q)$ ist die Ordnung bzw. Punktzahl der elliptischen Kurve E über GF(q)

Ferner sei N die Punktzahl für eine elliptische Kurve E über GF(q) es gilt dann:

$$N = q + 1 - a \tag{3.4}$$

a ist eine Konstante, die von der ausgewählten elliptischen Kurve abhängt

Aus (3.3) und (3.4) kann nun die Ordnung für einen Erweiterungskörper der Kurve *E* bestimmt werden. Das praktische Vorgehen bei der Bestimmung der Kurvenordnung eines Erweiterungskörpers kann wie folgt zusammengefasst werden:

1. Auswahl einer elliptischen Kurve über einem kleinen Erweiterungskörper GF(2^k)
2. Bestimmung der Kurvenordnung (beispielsweise durch Abzählung der Punkte)
3. Anwendung des folgenden Algorithmus, um die Ordnung für den Erweiterungskörper GF(2^{kr}) zu bestimmen.

Algorithmus für die Kurvenordnung des Erweiterungskörpers

Die elliptische Kurve *E* über GF(*p*) behält ihre Parameter und wechselt in den Körper GF(q^r). Die Ordnung der Kurve ändert sich (3.4) auf folgende Formel. Einen Beweis hierzu steht in [SIL86]:

$$N_r = q^r + 1 - \alpha^r - \bar{\alpha}^r \tag{3.5}$$

α und $\bar{\alpha}$ sind die Wurzeln des quadratischen Polynoms $T^2 - aT + q$. Die Konstante *a* stellt die Verbindung über die Identitäten (3.6) und (3.7) zur der Anwendung des folgenden Algorithmus dar.

Nun können ausgehend von einer bestimmten elliptischen Kurve sämtliche Erweiterungs-Ordnungen berechnet werden:

Algorithmus für die Berechnung der Kurvenordnung über GF(q^r) für $r > 1$

1. $a \leftarrow q + 1 - N_1$
2. $u \leftarrow 2$
3. $v \leftarrow a$
4. **for** $i \leftarrow 1$ **to** r-1
5. $w \leftarrow a \cdot v - q \cdot u$
6. $u \leftarrow v$
7. $v \leftarrow w$
8. **next** *i*
9. **stop with** $q^r + 1 - w$

Der Algorithmus verwendet folgende Identitäten:

$$(\alpha^r - 1)(\bar{\alpha}^r - 1) = q^r + 1 - \alpha^r - \bar{\alpha}^r \text{ unter Verwendung von } \alpha \cdot \bar{\alpha} = q \tag{3.6}$$

$$\alpha^i + \bar{\alpha}^i = a(\alpha^{i-1} + \bar{\alpha}^{i-1}) - q(\alpha^{i-2} + \bar{\alpha}^{i-2}) \; \forall i \geq 2 \text{ mit Hilfe von } a = \alpha + \bar{\alpha} \tag{3.7}$$

Für ein Beweis des Theorems von Hasse, sei hier nochmals auf [SIL86] §V.2 sowie ergänzend auf [KOB98] S. 126 f. und [HAM98] S. 37, verwiesen.

Beispiel einer Erweiterung des Körpers

$$y^2 + xy = x^3 + x^2 + 1 \text{ über GF(2)} \tag{3.8}$$

Die Bestimmung der Kurvenordung eines Erweiterungskörpers wird am besten anhand eines Beispieles verdeutlicht. Nimmt man die Kurve (3.8), dann sieht das Vorgehen wie folgt aus:

1. Berechnung der Kurvenordnung durch Abzählung der Punkte. Bei einem endlichen Körper mit geringer Ordnung (Elementzahl) können systematisch alle Koordinatenpaare auf Zugehörigkeit anhand der Gleichung geprüft werden.
2. In diesem Fall existiert neben dem Unendlichkeitspunkt *O* nur der Punkt (0,1), die Kurve hat die Ordnung 2.
3. Nun kann mit obigen Algorithmus, in Schritt 1, der Koeffizient *a* berechnet werden, was hier 1 ergibt.
4. Anschließend wird der Wiederholungsblock solange ausgeführt, bis der gewünschte Grad erreicht ist.

Die Variable *w* durchläuft die Werte: -3, -5, 1, etc. Womit die Ordnungen durch die Formel

$$f(w) = q^r + 1 - w, \tag{3.9}$$

zugeordnet werden können. Es ergeben sich folgende Ordnungen: 8 für GF(2^2), 14 für GF(2^3) und 16 für GF(2^4).

3.2 Anmerkung zur Sicherheit von Erweiterungskurven

Die im obigen Beispiel verwendete Kurve über GF(2), d. h. alle Koeffizienten sind entweder 0 oder 1, gehört zu den **anomalen binären Kurven**, sie werden auch **Koblitz Kurven** genannt. (Koblitz war der erste, der sich mit diesen Kurven beschäftigt hat.)

Diese Kurven eignen sich unter sicherheitstechnischen Aspekten **nicht** für reale kryptographische Anwendungen, da sich bei ihnen das zugrunde liegende sicherheitstechnische Problem, nämlich die Berechnung des diskreten Logarithmus, drastisch reduzieren lässt.
Ähnliches gilt für allgemeinere Kurven, bei denen die Koeffizienten der elliptischen Kurvengleichung für einen Körper mit geringerem Grad verwendet wurden:

Bei der Verwendung einer elliptischen Kurve über dem Körper $GF(2^e)$, bei dem die Kurvenordnung noch praktisch durch abzählen aller Punkte möglich ist (durch ein entsprechend kleines *e*), soll nun in der eigentlichen Anwendung der Erweiterungskörper $GF(2^{ed})$ verwendet werden. Bei dieser Konstellation ist es mit obigen Verfahren einfach, die resultierende Kurvenordnung zu berechnen.

Leider wird es Angreifern hierdurch auch einfacher gemacht, das System zu *brechen* bzw. den diskreten Logarithmus zu berechnen. Der Reduktionsfaktor bei einem Angriff ist:

$$\sqrt{2d}$$

Bei einer anomalen binären elliptischen Kurve über $GF(2^{163})$ reduziert sich der Aufwand auf 2^{77} Berechnungen auf der elliptischen Kurve. Für Details sei der Leser auf [WZ98] verwiesen.

Diese Schwäche mindert zwar den Einsatz obiger Methode für praktische Anwendungen, für Testkurven behält die Methode jedoch ihre Daseinsberechtigung. Besonders für die Generierung von Testvektoren ist die Methode sehr gut geeignet. Möchte man sie auch für reale Sicherheitsanwendungen einbeziehen, so muss die angestrebte Sicherheit in einer größeren Kurvenordnung, gegenüber Kurven die nicht aus einer Kurvenerweiterung bestehen, Berücksichtigung finden.

3.3 Ermittlung elliptischer Kurven

Da man an einer Kurvenordnung mit möglichst großen Primteilern interessiert ist, die Ordnung selbst aber keine Primzahl sein sollte, ist es günstig, eine Primfaktorzerlegung der Kurvenordnung vorzunehmen und den größten Primteiler auszuwählen.
Bei einer elliptischen Kurve, deren Ordnung prim ist, kann ein spezieller Angriff angewendet werden, der die Sicherheit des Verfahrens stark reduziert. Auf die Hintergründe dieses Angriffes soll hier nicht eingegangen werden, der Leser sei auf [MOV93] verwiesen.
Um diesem Angriff zu entgehen, wird gefordert, dass die Ordnung zumindest aus zwei Primteilern besteht (vgl. [D7-98], Annex A). Für die Verwendung in kryptographischen Verfahren wird dann die größere Teilkurve verwendet, d. h. der Basispunkt B sollte in dieser Teilkurve liegen.

Die Ordnung der Kurve E in Primfaktoren ist:

$$\#(E) = k = \prod_{i \in I} a_i^{l_i} \tag{3.10}$$

a_i seien die Primfaktoren von k und l_i die Potenzen dieser Faktoren

Für einen Punkt B der Kurve gilt nun:

$$a_i B = O \rightarrow B \text{ ist Teilkurve mit der Ordnung } a_i, \text{ für } B \in E, B \neq O$$

Multipliziert man einen Punkt B einer elliptischen Kurve E mit seiner Ordnung, so ist das Ergebnis der Unendlichkeitspunkt O.

Der Zusammenhang zwischen der Ordnung eines Punktes und der Kurvenordnung wird durch folgende Gegenüberstellung aufgezeigt:

Kurvenordnung: Anzahl aller Punkte einer Kurve E. Funktionssymbol: #(E)

Punktordnung: Anzahl der Punkte einer Teilkurve, in der sich der Punkt befindet. Diese Teilkurve bildet eine Untergruppe. Jede Untergruppe entspricht einem Primfaktor in der Faktorzerlegung der Kurvenordnung.

Man kann leicht überprüfen, ob ein Punkt sich in dieser größeren Teilgruppe bzw. Teilkurve befindet, indem man ihn mit der Ordnung der Teilkurve, also dem Primfaktor, multipliziert. Das Ergebnis muss der unendlich ferne Punkt O sein, ist das Ergebnis $\neq O$, so wählt man sich zufällig einen anderen Punkt, bis man einen gesuchten Punkt der Teilkurve findet.

Das Problem, einen geeigneten Basispunkt einer elliptischen Kurve zu ermitteln, ist bis heute nicht zufrieden stellend gelöst (vgl. [KOB98]). Es existiert ein probabilistischer Algorithmus mit der Laufzeitkomplexität $O(\ln^3 q)$. Leider existiert kein allgemeiner deterministischer Algorithmus in Polynomialzeit, lediglich für Spezialfälle existieren schnelle Algorithmen.

Für die Praxis sind die probabilitischen Algorithmen jedoch praktikabel, da die Wahrscheinlichkeit für einen Treffer nach wenigen Versuchen sehr hoch wird. In [KOB98], Kap. 6.1.8, wird die Problematik der Punktfindung, sowie die Laufzeit des probabilistischen Algorithmus untersucht.

Das Problem, einen Punkt in einer bestimmten Teilgruppe zu finden hängt allerdings von der Größe dieser Gruppe ab. Die Wahrscheinlichkeit kann anteilig zur Körperkardinalität (für endliche Körper) berechnet werden:

$$P(GF(q), \#(E)) = \frac{\#(E)}{q} \cdot p_t \tag{3.11}$$

Hierbei ist p_t die Wahrscheinlichkeit für einen Treffer nach t Versuchen.

Bei einem Körper GF(2^{160}) und einer Kurvenordnung von ~2^{145} ist dies $2^{145} / 2^{160} \cdot p_t$.

Die Betrachtung des größten Primteilers relativ zur Kurvenordnung gibt Aufschluss darüber, wie effizient die Kurve ausgenutzt wird. Ist die Teilkurve zu klein im Verhältnis zur Kurvenordnung, so operiert man in einer Kurve, die auch mit wesentlich kleineren Zahlen und Körpern behandelt werden könnte. Dieser Zustand ist natürlich sehr ineffizient und sollte vermieden werden.

Wie groß im einzelnen die Differenzen sein können ist der folgenden Abbildung 3.1 zu entnehmen. Auf ihr ist die Kurvenordnung und die Ordnung des jeweils größten Primteilers, in logarithmischer Darstellung, zu sehen. Als Beispiel dient hier die bereits vorgestellte elliptische Kurve im GF(2^k) mit den Parametern a=1 b=1:

$$y^2 + xy = x^3 + x^2 + 1 \tag{3.8}$$

Die Größenordnung des Primteilers schwankt sehr stark. Die kleinste Abweichung ist 1 Bit und die größte Differenz zwischen dem größten Primteiler und der Kurvenordnung im betrachteten Bereich zwischen 1 und 130 liegt bei ca. 107 Bit für den Erweiterungsgrad 126. Der Mittelwert der Abweichungen zur Kurvenordnung liegt bei ca. 36,5 Bit. Die Kurvenordnung und der jeweils größte Primteiler sind in den Abbildungen 3.1 und 3.2 wiedergegeben.

Nahezu optimale Verhältnisse liegen vor, wenn die Differenz nahe eins ist. Eine Differenz von eins liegt beispielsweise für die Fälle: 11, 17, 19, 23, 101, 107, 109 und 113 vor. Was diese Differenz für die Kurvenordnung bedeutet, soll anhand einzelner Fälle in Tabelle 3.1 veranschaulicht werden:

Körpergröße/ Erweiterungsgrad	Kurvenordnung[+)]	Primfaktoren der Kurvenordnung
11	1982	2 991
101	2535301200456461772285617016022	2 126765060022823088614280850801 1
102	50706024009129206130947394067 76	2^3 7 137 239 65587 2452358563 17192986687
113	10384593717069655255793407666935014	2 519229685853482762789670383346750 7

Tabelle 3.1: Kurvenordnung und deren Primfaktoren für Beispielskurve (3.7)

[+)] Die Kurvenordnungen sind ganze Zahlen, die jedoch über mehrere Zeilen umgebrochen sind.

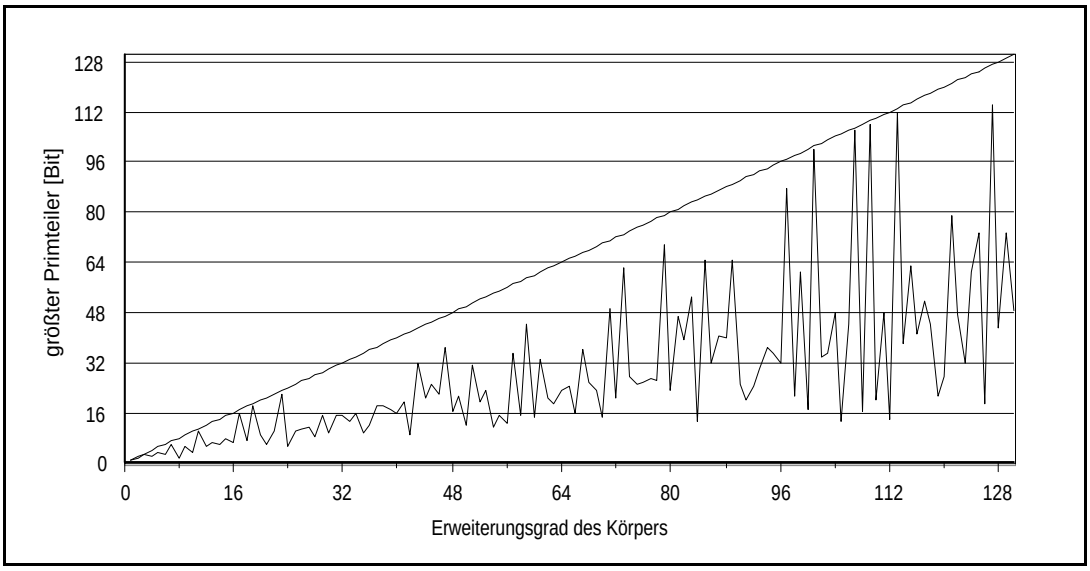

Abbildung 3.1: Größe der Primteiler relativ zur Kurvenordnung für (3.8)

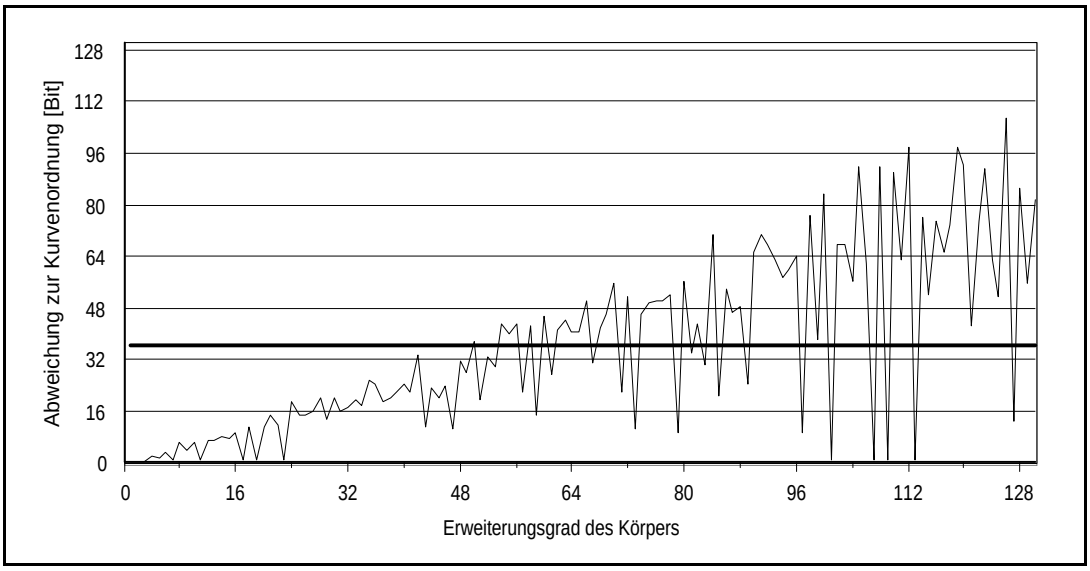

Abbildung 3.2: Abweichungen des größten Primteilers zur Kurvenordnung für (3.8)
Der Mittelwert ist als **fette Linie** hervorgehoben.

Der Vollständigkeit halber seien hier die irreduziblen Polynome für einige Fälle, in denen die Kurvenordnung aus zwei Primfaktoren besteht und von denen genau einer 2 ist, angegeben:

Ordnung	**irreduzibles Polynom**
11	$x^{11}+x^2+1$
107	$x^{107}+x^9+x^7+x^4+1$
109	$x^{109}+x^5+x^4+x^2+1$
113	$x^{113}+x^9+1$

Tabelle 3.2: Auswahl irreduzibler Polynome

Mit dieser vorgestellten Methode ist es nun einfach eine ausreichende Anzahl von Testmustern zu erzeugen. Dies erleichtert den funktionalen Test des ASICs.

Unterschiede elliptischer Kurven im Vergleich zu

3.4 Unterschiede elliptischer Kurven im Vergleich zu anderen Public Key Verfahren (RSA)

Elliptische Kurven sind in der Kryptographie eine relativ innovative Erscheinung. Erst jetzt dringen sie in erste kommerzielle Anwendungen vor. Es wurde bereits erwähnt, dass elliptische Kurven allein noch kein kryptographisches Verfahren darstellen, sondern vielmehr eine Grundlage für kryptographische Verfahren bieten. Mit elliptischen Kurven wird als asymmetrisches (public key) Verfahren meist ElGamal verwendet. Das ElGamal Verfahren (vgl. [BOH97,SCH96]) setzt direkt auf der Gruppenoperation der elliptischen Kurven auf.

Ein anderes sehr prominentes und weit verbreitetes asymmetrisches Verfahren ist RSA, welches auf der **modularen Exponentiation** aufsetzt. Die modulare Exponentiation findet bei RSA über einem Restklassenring *Z/nZ* statt, wobei *n* das Produkt aus zwei Primzahlen (*p* und *q*) ist. Die Hauptanweisung für modulare Exponentiation lautet:

$$x^k \bmod n, \text{ mit } x < n, n = p \cdot q$$

Hierbei ist insbesondere die Modulo-Operation sehr zeit- und rechenaufwendig. Die Sicherheit bei dem RSA-Verfahren hängt direkt von der Schlüssellänge und indirekt von dem Modul *n* ab.

Da RSA das mit Abstand am häufigsten eingesetzte asymmetrische Verfahren ist, soll es an dieser Stelle zum Vergleich mit den elliptischen Kurven herangezogen werden. Die kryptographischen Verfahren werden beispielsweise in [SCH96] näher erläutert.
An dieser Stelle sollen keine kryptographische Verfahren erklärt werden, hier geht es um die Gegenüberstellung von Vor- und Nachteilen.

Ein wesentlicher Vorteil von elliptischen Kurven gegenüber RSA ist die geringere Schlüssellänge bei vergleichbarer Sicherheit. Dies wird durch eine aktuelle Studie [LV99], die sich erstmals nur mit dem Thema der vergleichbaren Sicherheit beschäftigt, bestätigt.

Ein Vergleich von Schlüssellängen für RSA und elliptische Kurven ist in Abb. 3.3 dargestellt:

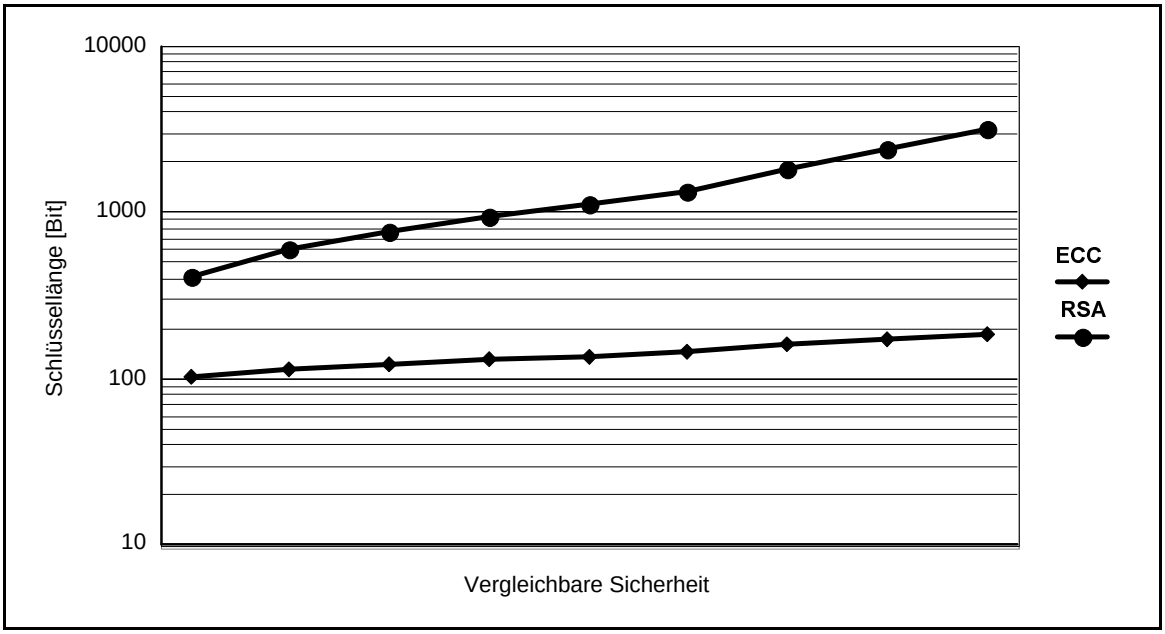

Abbildung 3.3: Vergleichbare Sicherheit für elliptische Kurven und RSA

Abb. 3.3 zeigt die Schlüssellänge auf einer logarithmischen Skala, diese Skala ermöglicht den Vergleich über einen weiten Bereich der Schlüssellängen. Es wird deutlich, dass die Schlüssellänge in Abb. 3.3 bei RSA mindestens **linear stärker** steigt, als bei elliptischen Kurven (EC):

So entsprechen 105 Bit bei EC 417 Bit bei RSA; bei 132 Bit EC werden bereits 952 Bits RSA benötigt. Insgesamt, also ohne den logarithmischen Maßstab, steigt die Schlüssellänge bei RSA exponentiell im Vergleich zu EC.
Dieser Vergleich zeigt, warum man sich von der Einführung von EC viel verspricht. Die kürzeren Schlüssellängen haben jedoch noch einen weiteren wesentlichen Vorteil. Dieser liegt in den **kürzeren Nachrichten** für die Schlüsselübermittlung und für Signaturen.

Abb. 3.3 zeigt auch, dass die Diskrepanz zwischen beiden Verfahren mit der Schlüssellänge wächst. So mag die Differenz für kürzere Schlüssellängen noch klein sein, bei längeren Schlüsseln wird der Unterschied jedoch deutlich.

3.5 Zusammenfassung

Ziel von Kapitel drei ist es, die Erzeugung von Testkurven zu ermöglichen (vgl. Kap. 3.1). Hierzu gehört die Bestimmung der Kurvenordnung, sowie das Auffinden eines geeigneten Basispunkts (Kap. 3.3). In Kap. 3.4 wurden elliptische Kurven mit ihrem enormen kryptographischen Potential in einen Vergleich zu RSA vorgestellt.
Im folgenden Kapitel werden Realisierungsalternativen von elliptischen Kurven behandelt.

4 Analyse von Realisierungsoptionen

In diesem Kapitel sollen verschiedene Berechnungsalternativen für die Skalarmultiplikation auf elliptischen Kurven untersucht werden.
Ausgehend von den mathematischen Betrachtungen des vorigen Kapitels und der Verwendung von projektiven Koordinaten für elliptische Kurven, werden in diesem Kapitel konkrete Berechnungsvorschriften analysiert. Die verwendeten Berechnungsalternativen werden dann im folgenden Kapitel evaluiert.

Die Skalarmultiplikation kann, wie im Kapitel 2.7 beschrieben, auf der Grundlage von Punktadditionen und Punktverdopplungen berechnet werden. Der Aufwand für die Skalarmultiplikationen hängt durch diese Aufteilung wesentlich von dem Aufwand der Addition und Verdopplung von Punkten ab. Unter Aufwand soll hier die Anzahl der *primitiven* Berechnungen im $GF(2^k)$ verstanden werden. Unter *primitiven* Berechnungen sollen Additionen und Multiplikationen im $GF(2^k)$ verstanden werden.
Grundlage für weitere Analysen ist zunächst die Beschreibung der unterschiedlichen Operationen, dies ist Gegenstand des folgenden Abschnitts.

4.1 Aufwand für Addition und Verdopplung

Der Aufwand für die Berechnung der Punktaddition (ungleicher Punkte) und der Punktverdopplung für elliptische Kurven mit projektiven Koordinaten ist in Tabelle 4.1 angegeben. Es werden die Multiplikationen und Additionen im allgemeinen Fall, sowie in Spezialfällen angegeben. Die Art des Spezialfalls ist als Ausdruck in Klammern ergänzt. Zusätzlich ist auch die Zahl der benötigten temporären Variablen notiert.

	Operation auf elliptischer Kurve	**Multiplikationen** in $GF(2^k)$	**Additionen** in $GF(2^k)$
1a	**Addition**	**20**	**7**
1b	Addition (a=0)	18	6
1c	Addition (Z_1=1)	15	7
1d	Addition (a=0 und Z_1=1)	13	6
2a	**Punktverdopplung**	**10**	**4**
2b	Addition (gleiche Punkte)	18	6

Tabelle 4.1: Aufwandsübersicht

Beschreibung der Spezialfälle

1a Normalfall der Punktaddition

1b Der Kurvenparameter a sei null

1c Die Z-Koordinate des zweiten Punktes sei ungleich null

1d Kombination aus 1b und 1c

2a Normalfall der Punktverdopplung

2b In diesem Fall wurde die Additionsroutine mit gleichen Punkten aufgerufen und eine Punktgleichheit wurde zuvor nicht geprüft. Die Additionsroutine bricht nach einem bestimmten Berechnungsteil ab und verwendet darauf die Punktverdopplung.
Es entsteht ein Gemisch aus Addition und Punktverdopplung, das jedoch noch ein Quentchen (2 Multiplikationen und 1 Addition weniger) *günstiger* als die einfache Addition ist.

Im Folgenden werden die oben angegebenen Berechnungen (Tabelle 4.1) durch Algorithmen aufgeschlüsselt. Hierbei wird zunächst auf die Punktaddition und Verdopplung und im Anschluss auf die Skalarmultiplikation eingegangen werden.

4.2 Aufwandsreduzierung durch Mehrfachbelegungen

Ausgangslage für die folgenden Betrachtungen sind die Gruppenoperationen für elliptische Kurven mit Koordinaten in der projektiven Ebene. Die Formeln hierfür befinden sich in Kapitel 2.6. Die Formeln sind im Folgenden zur besseren Übersicht abermals aufgeführt.

Bei der Implementation eines Algorithmus, müssen alle Variablen (der Berechnungsvorschrift) eindeutig (temporären) **Registern** und Operationen eindeutig **Operationseinheiten** (ALUs) zugeordnet werden.
Bei dem Abbildungsvorgang von Variablen auf Register ist die zeitliche Reihenfolge der belegten Register zu ermitteln; Eine Hilfe hierfür ist der später beschriebene Datenflussgraph der Berechnungsvorschriften.
Im Folgenden werden die beiden möglichen Additionen, ungleiche und gleiche Punkte, jeweils mit ihren Berechnungsvorschriften und deren Abbildungen auf Register und Operationseinheiten beschrieben.

Addition verschiedener Punkte

Die Berechnungsvorschrift für die Addition von verschiedenen projektiven Punkten besteht aus folgenden Teilen:

$$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2) \tag{4.1}$$

Die Variablenbelegung ist in Abb. 4.1 wiedergegeben, neben (4.1) geht auch der Kurvenparameter a aus (4.2) ein.

$$E : y^2 + xy = x^3 + ax^2 + b \quad \text{mit} \quad a, b, x, y \in GF(2^k) \tag{4.2}$$

$U_0 = X_0 Z_1^2$	$W = U_0 + U_1$	$L = Z_0 W$	$T = R + Z_2$
$S_0 = Y_0 Z_1^3$	$S_1 = Y_1 Z_0^3$	$V = RX_1 + LY_1$	$X_2 = aZ_2^2 + TR + W^3$
$U_1 = X_1 Z_0^2$	$R = S_0 + S_1$	$Z_2 = LZ_1$	$Y_2 = TX_2 + VL^2$

Abbildung 4.1: Berechnungsvorschrift mit Variablen für die projektive Punktaddition unterschiedlicher Punkte

Die Berechnungsformeln für die Addition von verschiedenen Punkten (Abb. 4.1) besteht aus folgenden Teilen:

- 7 Variablen für die Eingabe, die sich aus je 3 für einen Punkt, sowie einer Kurvenkonstante zusammensetzen.
  Variablen: X_0, Y_0, Z_0, X_1, Y_1, Z_1, a (4.2)
- 9 Variablen für die Berechnung der Zwischenergebnisse.
  Variablen: U_0, U_1, S_0, S_1, W, R, L, V, T (4.3)
- 3 Variablen für das Ergebnis.
  Variablen: X_2, Y_2, Z_2 (4.4)

In Summe sind dies 19 Variablen aus dem Wertebereich des Körpers GF(2^k). Bei der Implementation eines Algorithmus, müssen alle Variablen eindeutig Registern und Operationen eindeutig ALUs zugeordnet werden. Bei der Abbildung auf Register werden weniger als 19 Register benötigt, da einige Register mehrfach, d.h. für unterschiedliche Variablen, verwendet werden können:

Bei der Abb. 4.1 ergeben sich folgende Einsparungen:

1. Die Ergebnisvariablen (4.4) können mit den Variablen für die Berechnung der Zwischenergebnisse (4.3) verbunden werden.
   Einsparung: 3 Register
2. Die Eingangsvariablen des ersten Punktes (4.2) können mit den Variablen der Zwischenergebnisse (4.3) verbunden werden.
   Dies ist durch den Algorithmus der Skalarmultiplikation (vgl. Kap. 4.4) möglich, da jeweils ein Punkt bei der Addition überschrieben werden darf.
   Einsparung: 3 Register
3. Mehrere Variablen für die Berechnung des Zwischenergebnisses (4.3) können auf ein Register abgebildet werden. Folgende Variablen können in dem unten angegebenen Algorithmus (*Algorithmus Projektive Punktaddition*) jeweils auf ein Register abgebildet werden: $W = U_0, S_1 = U_1, R = S_0, L = S_1$.
   Die konkrete Belegung kann dem Algorithmus entnommen werden.
   Einsparung: 4 Register

Somit reduziert sich der Aufwand von 19 Variablen auf 9 Register. Derartige Einsparungen sind mit Blick auf die benötigte Chiplayout-Fläche sehr positiv zu bewerten.

In [D7-98] wird die im Folgenden als Algorithmus *Projektive Punktaddition* wiedergegebene Ausführung vorgeschlagen.
Die Bedingungen aus Tabelle 4.1 finden sich als bedingte Verzweigungen (if-statements) wieder. A, B, ..., I stellen die Register der Berechnung dar.

Algorithmus: Projektive Punktaddition

A, B, ..., I seien Register und die Rechenoperationen "·" und "+" beziehen sich auf Elemente aus dem GF(2^k).

Begonnen wird mit der Initialisierung:

$A \leftarrow X_0$; $B \leftarrow Y_0$; $C \leftarrow Z_0$; $D \leftarrow X_1$; $E \leftarrow Y_1$; $F \leftarrow Z_1$

1. I ← *a* (*a ist Parameter der elliptischen Kurve vgl.* (*4.1*))
2. **if** F≠1 **then** (*Falls* F=1 *gilt, würde nur mit* 1 *Multipliziert*)
3. G ←F·F (= Z_1^2)
4. A ←A·G (= U_0)
5. G ←F·G
6. B ←B·G (= S_0)

end

7. G ←C·C
8. H ←D·G (= U_1)
9. A ←A+H (= *W*)
10. G ←C·G
11. H ←E·G (= S_1)
12. B ←B+H (= *R*)
13. **if** A=0 **then**
14. **if** B=0 **then**

stop with „Punktverdopplung“ (*Identische Punkte erkannt*)

15. **else** **stop with** (1,1,0) (*Ergebnis ist O*)
16. **end**
17. D ←B·D
18. C ←A·C (= *L*)
19. E ←C·E
20. D ←D+E (= *V*)
21. E ←C·C
22. G ←D·E
23. **if** F≠1 **then** C←C·F (= Z_2 *falls* F=1 *ist bereits* Z_2 = L)
24. D ←B+C (= *T*)
25. B ←B·D
26. E ←A·A
27. A ←A·E
28. **if** I≠ *0* **then**
29. H ←C·C

30. I ←H·I
31. A ←A+I
end
32. A ←A+B (= X_2)
33. D ←A·D
34. B ←D+G (= Y_2)
35. stop with (A,B,C)

Visualisierung der Berechnung

Der Algorithmus ist nicht übersichtlich und der Berechnungsfluss ist nur schwer nachvollziehbar. Eine Abhilfe schafft die Darstellung als Graph mit einzelnen Berechnungsknoten.

Einen Überblick der tatsächlichen Datenabhängigkeiten der Punktaddition, liefert der ***data dependency graph*** (DDG), der auch als Datenflussgraph bekannt ist.

In dem DDG werden ausgehend vom Ergebnis, sämtliche Zwischenoperationen und ihre Abhängigkeiten von den Eingangswerten in Form eines Berechnungsbaums, dargestellt. Am oberen Rand befinden sich die Eingangswerte, am unteren die Ergebnisse. In dem Mittelteil sind die Berechnungsknoten: In diesem Fall ⊗ für die Punktverdopplung und ⊕ für die Punktaddition.
Jeder Berechnungsknoten bekommt seine Eingangswerte von oben und liefert das Ergebnis nach unten. **Fette Linien** kennzeichnen, dass die Eingangswerte beider Operanden gleich sind.

Die Abhängigkeit der Variablen bzw. Register des Algorithmus *Projektive Punktaddition* ist im folgenden Datenflussgraphen (Abb. 4.2) wiedergegeben:

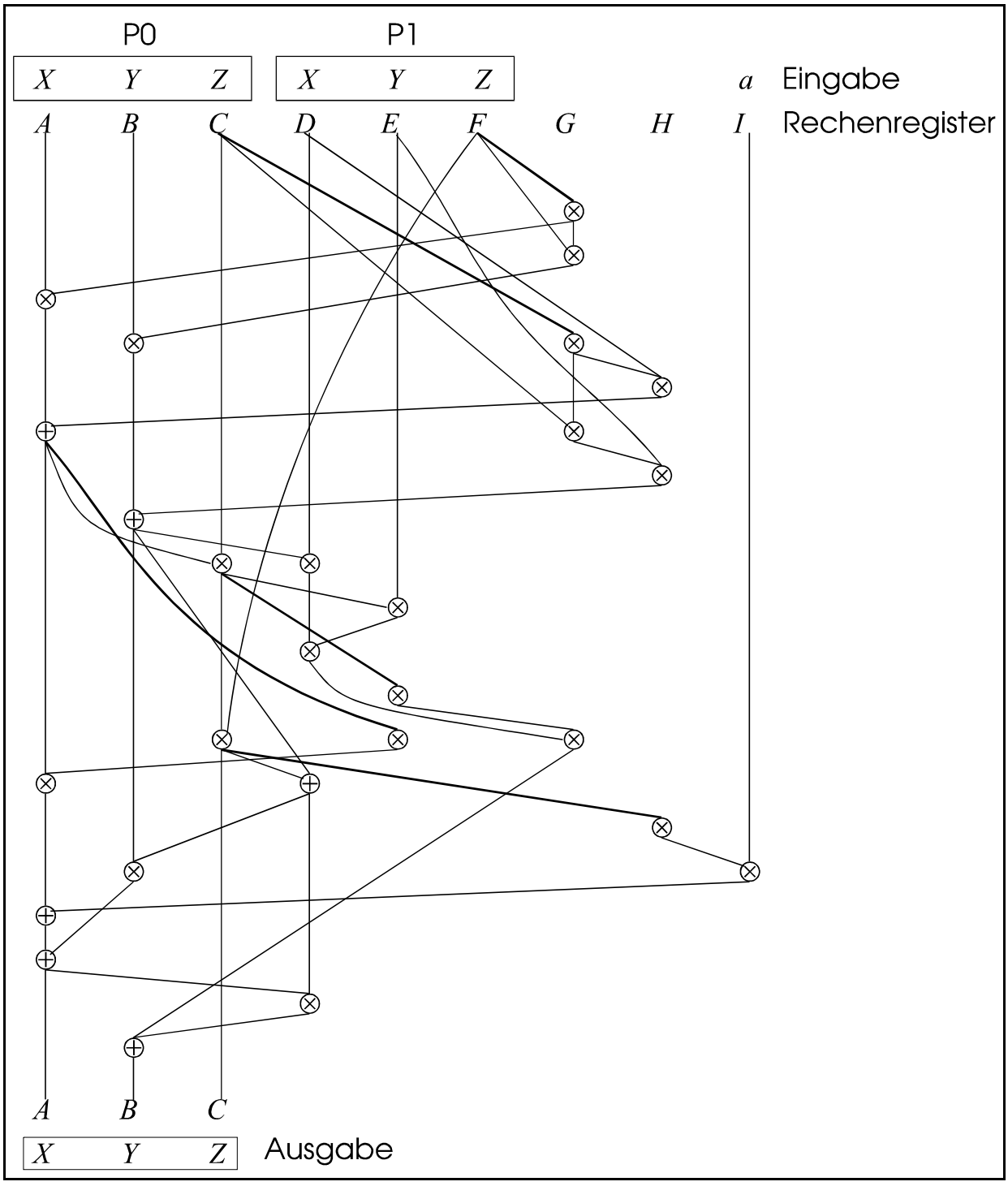

Abbildung 4.2: Datenflussgraph projektive Punktaddition (ungleiche Punkte)

Punktverdopplung

Die Berechnungsvorschrift für die Addition von gleichen projektiven Punkten (Punktverdopplung) besteht aus folgenden Teilen:

$$2(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2) \text{ über dem Körper GF}(2^k) \tag{4.5}$$

$$\begin{aligned} c &= b^{2^{k-2}} = b^{1/4} \\ Z_2 &= X_1 Z_1^2 \\ X_2 &= (X_1 + cZ_1^2)^4 \\ U &= Z_2 + X_1^2 + Y_1 Z_1 \\ Y_2 &= X_1^4 Z_2 + U X_2 \end{aligned} \tag{4.6}$$

Die Berechnungsformeln für die Addition von verschiedenen Punkten (4.5) und (4.6) besteht aus folgenden Teilen:

- 3 Variablen für die Eingabe, die sich aus je 3 für einen Punkt, sowie einer Kurvenkonstante zusammensetzen.
  Variablen: X_1, Y_1, Z_1, c (4.7)
- 4 Variablen für die Berechnung der Zwischenergebnisse.
  Variablen: U, X_2, Y_2, Z_2 (4.8)
- 3 Variablen für das Ergebnis.
  Variablen: X_2, Y_2, Z_2 (4.9)

In Summe sind dies 10 Variablen aus dem Wertebereich des Körpers GF(2^k). Bei der Abbildung ergeben sich folgende Einsparungen:

1. Die Ergebnisvariablen (4.9) können mit den Variablen für die Berechnung der Zwischenergebnisse (4.8) verbunden werden.
   Einsparung: 3 Register
2. Die Eingangsvariablen des Punktes (4.7) können mit den Variablen der Zwischenergebnisse (4.8) verbunden werden.
   Dies ist durch den Algorithmus der Skalarmultiplikation (vgl. Kap. 4.4) möglich, da bei der Punktverdopplung der Ausgangspunkt überschrieben werden darf.
   Einsparung: 3 Register

Die Punktverdopplung benötigt nur ein zusätzliches Register, also insgesamt 4 Register.

Die Datenabhängigkeiten werden im folgenden Datenflussgraph (data dependency graph) veranschaulicht:

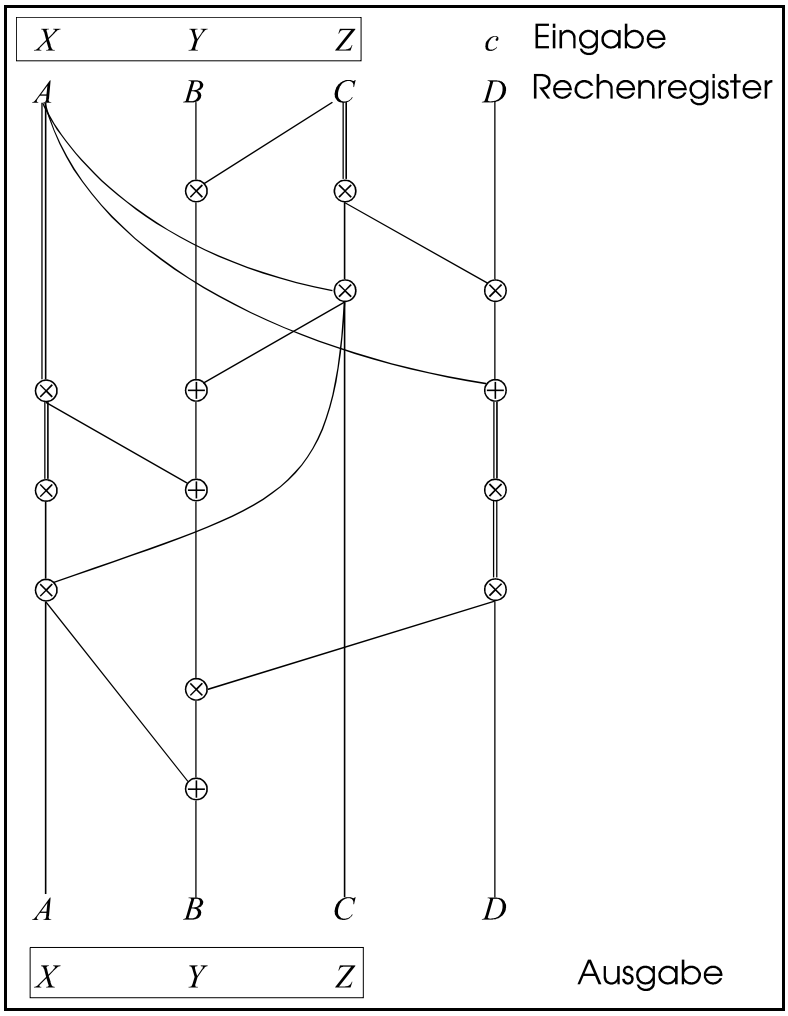

Abbildung 4.3: Datenflussgraph projektive Punktverdopplung

Am oberen Rand sind die Eingangsgrößen auf die Register *A* bis *D* angegeben. Die Knoten mit den Operationssymbolen haben jeweils zwei Eingänge. Eine **Doppellinie** bedeutet, dass beide Operanden gleich sind.
Die horizontale Position gibt das verwendete Register wieder. Am unteren Rand befinden sich nach der Berechnung das Ergebnis.
Anhand des Datenflussgraphen lässt sich ablesen, welche Register zu welcher Zeit benötigt werden und welche Reihenfolgen bei der Berechnung möglich sind.

Algorithmus: Projektive Punktverdopplung

$$2(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

A, B, C, D seien Register und die Rechenoperationen "·" und "+" beziehen sich auf Elemente aus dem GF(2^k).

Begonnen wird mit der Initialisierung:

$$A \leftarrow X_1;\ B \leftarrow Y_1;\ C \leftarrow Z_1$$

1. D ← *c* (*c ist ein abgeleiteter Parameter der elliptischen Kurve vgl.* (*4.6*))
2. B ←B·C
3. C ←C·C
4. D ←C·D
5. C ←A·C (= Z_2)
6. D ←A+D
7. A ←A·A
8. B ←B+C
9. D ←D·D
10. B ←A+B (= U)
11. A ←A·A
12. A ←A·C (= X_2)
13. D ←D·D
14. B ←B·D
15. B ←A+B (= Y_2)
16. **stop with** (A,B,C)

Zusammenfassung Addition und Verdopplung

Fasst man nun die Punktaddition (ungleicher Punkte) und -verdopplung von projektiven Punkten zusammen, so werden insgesamt **9 Register** benötigt. Dies ergibt sich aus der maximal benötigten Registerzahl von 9 bei der Addition und 4 bei der Verdopplung.

4.3 Aufwandsreduzierung durch Konfiguration

Je nach *Konfiguration* ist die Anzahl der Operationen für die Punktverdopplung unterschiedlich. Unter Konfiguration sollen die **Systemparameter** verstanden werden. Zu den Systemparametern gehören:

- Kurvenparameter der elliptischen Kurve: a,b bzw. a und c aus (vgl. 4.2 u. 4.6),
- der Körper GF(2^k) und das verwendete irreduzible Polynom,
- und die gewählte Basis des Körpers (hier polynomielle Basis).

Für diese Arbeit wurde die Wahl auf elliptischen Kurven in projektive Koordinaten über dem Körper GF(2^k) mit polynomieller Basis getroffen. Der mit dieser Wahl behaftete Aufwand ist in Tabelle 4.1 als Übersicht wiedergegeben.

Durch die Festlegung der Kurvenparameter können die Algorithmen der Punktverdopplung und -addition weiter vereinfacht werden:

Die obere Grenze für die Verdopplung kann durch 20 Multiplikationen und sieben Additionen abgeschätzt werden. Diese Grenze wird durch den DDG in Abb. 4.3 visualisiert.

Ist der Kurvenparameter a jedoch null, so entfallen zwei Multiplikationen und eine Addition. Ist die Z-Komponente des ersten Punktes null, so entfallen fünf Multiplikationen. Die beiden Konfigurationen können auch zusammentreffen, die Einsparungen addieren sich in diesem Fall. Diese Konfigurationen sind in der Übersichtstabelle (Tab. 4.1) als Spezialfälle aufgenommen.

4.4 Skalarmultiplikation

Die Skalarmultiplikation wurde bereits in Kapitel 2.7 vorgestellt, in diesem Abschnitt soll nun etwas tiefer auf unterschiedliche Realisierungsmöglichkeiten eingegangen werden. Insbesondere die Frage, wie die Skalarmultiplikation auf eine Folge von Punktadditionen und Verdopplungen abgebildet wird, bildet den Schwerpunkt dieses Abschnitts.

$$k \cdot B \text{ mit } k \in \mathbb{N},\ B \in E \text{ (Punkt auf der Kurve)}$$

Die Skalarmultiplikation auf elliptischen Kurven bietet eine Vielzahl von Realisierungsmöglichkeiten. Hierbei wird die Skalarmultiplikation entweder nur auf die Punktaddition oder auf die Punktsubtraktion und -addition angewendet. Neu im Vergleich zur Darstellung im Kapitel 2.7 ist die Punktsubtraktion als Hilfsoperation für die Skalarmultiplikation.
Für den tatsächlichen Berechnungsaufwand der Skalarmultiplikation ist zudem noch die verwendete Basis ausschlaggebend. Auf die Verwendung einer Basis wird nur kurz am Ende dieses Abschnitts eingegangen, da dieses Verfahren eine Menge vorberechneter Elemente voraussetzt. Diese Voraussetzung ist aber aufgrund des Speicherbedarfs für diese Menge hier nicht realisierbar. Doch es sei vorweg bereits gesagt, dass dieses Verfahren bei weitem das schnellste ist.

Die Problematik der Skalarmultiplikation bei elliptischen Kurven entspricht dem Exponentieren mit natürlichen Zahlen. In beiden Fällen soll eine Berechnung aus mehreren Teilen

zusammengesetzt werden, wobei die Anzahl der Operationen insgesamt möglichst gering sein soll. Bei der Skalarmultiplikation können Teilergebnisse addiert werden und bei der Exponentiation werden Teilergebnisse miteinander multipliziert. Eine gute Übersicht der zur Verfügung stehenden Verfahren mit Bezug auf elliptische Kurven findet man in [GOR98].

Die Frage kann mit dem Begriff der Additionsketten (*addition chains*) wie folgt formalisiert werden: Wie wenig Operationen benötigt man um $k \cdot B$ zu berechnen, wenn nur die Addition erlaubt ist? Dies ist äquivalent zu der Frage, wie lang die kürzeste Additionskette für k ist.

Eine **Additionskette** für k ist eine Liste von natürlichen Zahlen:

$$a_1=1,\ a_2,\ \ldots,\ a_n = k \tag{4.10}$$

mit der Eigenschaft $\forall i > 1 : \exists j, m : 1 \le j \le m < i : a_i = a_j + a_m$

Eine Additionskette berechnet nun sukzessiv alle $a_i \cdot B$, je kürzer die Kette, desto geringer der Aufwand. Mit *l(k)* sei die **kürzeste Additionskette** für k bezeichnet. Der Wert für *l(k)* ist nur für relativ kleine Argumente bekannt, für große k existiert folgende untere Grenze (vgl. [GOR98], S. 131):

$$l(k) = \log k + (1 + o(1))\frac{\log k}{\log \log k} \tag{4.11}$$

Additionsketten können generell verkürzt werden, wenn man neben der Addition auch Subtraktionen zulässt. Die Verwendung der Subtraktion bei der Exponentiation ist meist nicht sinnvoll, da eine Subtraktion einer Division entspricht und diese viel aufwendiger als die Multiplikation ist. Bei elliptischen Kurven ist die Subtraktion jedoch nur geringfügig aufwendiger und daher sind auch **Additions-Subtraktionsketten** realisierbar.

Die Unterschiede sieht man an dem Beispiel für k = 31, wobei hier die kürzesten Ketten angegeben sind:

Additionskette:	1, 2, 3, 5, 10, 11, 21, 31	*l(k)*= 8
Additions-Subtraktionskette:	1, 2, 4, 8, 16, 32, 31	*l(k)*= 7

Nun stellt sich die Frage, wie man geeignete Additionsketten berechnen kann.

Die grundlegenden Verfahren sind:

1. Als einfachste Möglichkeit bietet sich die wiederholte Addition eines Punktes.
2. Verwendung der binären Methode (*square and multiply*), dies entspricht der Schulmethode für die Multiplikation. (s.u. *Methode* 1)
3. Verwendung einer verbesserten Methode mit Additions-Subtraktionsketten. (s.u. *Methode* 2)

Es sei gleich angemerkt, dass diese Aufzählung nicht vollständig ist, aber für diese Realisierung durchaus ausreichend. Doch nun sollen die Verfahren vorgestellt werden. Da die erste Variante keiner weiteren Erläuterung bedarf, soll sogleich die zweite vorgestellt werden:

Methode 1: Binäre Methode (*Square and Multiply*) bzw. Schulmethode der Multiplikation

Berechnung von: $k \cdot B,\ k < 2^n,\ k \in \mathbb{N}, B \in E$ über $GF(2^n),\ k = \sum_{i=0}^{l} k_i \cdot 2^i$

mit $k_l = 1,\ k_i \in GF(2)$

1. E ← 0
2. **if** $k = 0$ **stop with** E
3. E ← B
4. **for** $i \leftarrow l$-1 **downto** 0
5. E ←2 E (*Punktverdopplung auf elliptischer Kurve*)
6. **if** $k_i = 1$ **then** E ←E+B (*Punktaddition auf elliptischer Kurve*)
7. **next** *i*
8. **stop with** E

Nach genauerer Betrachtung erkennt man, dass hier die Schulmethode der Multiplikation vorliegt: Es wird stellenweise (hier bitweise) verschoben. Dies entspricht der Punktverdopplung. Und falls eine eins beim Multiplikand vorliegt, so wird der Multiplikator auf das Ergebnis addiert. Eine eingehende Betrachtung dieser Methode, die auch als *binary-exponentation* bekannt ist, findet man nebst Aufwandsbetrachtungen in [MO90].

Methode 2: Verbesserte Methode der Multiplikation (*addition-subtraction chains*)

Berechnung von: $k \cdot B,\ k < 2^n,\ k \in \mathbb{N}, B \in E$ über $GF(2^n),\ k = \sum_{i=0}^{l} k_i \cdot 2^i$

mit $k_l = 1,\ k_i \in GF(2)$

Als Hilfsvariable wird *h* benötigt:

$$h = 3k,\quad h = \sum_{i=0}^{m} h_i \cdot 2^i \text{ mit } h_m = 1,\ h_i \in GF(2)$$

1. E ← 0
2. **if** $k = 0$ **stop with** E
3. E ← B
4. **for** $i \leftarrow m$-1 **downto** 1
5. E ←2 E (*Punktverdopplung auf elliptischer Kurve*)
6. **if** ($h_i = 1$) and ($k_i = 0$) **then** E ←E+B (*Punktaddition auf ell. Kurve*)
7. **if** ($h_i = 0$) and ($k_i = 1$) **then** E ←E-B (*Punktaddition auf ell. Kurve*)
8. **next** *i*
9. **stop with** E

Bei dieser Methode ist es nicht sofort ersichtlich, wie das Verfahren funktioniert. Zur Veranschaulichung soll nun der Skalar k =15 dienen:

Beispiel für Methode 2 mit Skalar $k = 15$:

Bei $k = 15$ lautet die binäre Notation $k = (1111)_2$ (hochwertigste Stelle links), für h ergibt sich $h = 3k = (101101)_2$. Für l und m ergeben sich: $l = 3$ und $m = 5$.
In der folgenden Tabelle (4.2) ist das temporäre Register E jeweils zu Beginn und Ende eines Schleifendurchlaufs angegeben. Ferner ist der Schleifenzähler (i), sowie die betrachteten Bits von h und k und die ausgeführte Aktion (Addition oder Subtraktion) angegeben. Die Verdopplung findet stets statt und ist aus diesem Grund nicht aufgeführt.

E zu Beginn	i	h_i	k_i	**Aktion**	**neues E**
B	4	0	0	nichts	2 B
2 B	3	1	1	nichts	4 B
4 B	2	1	1	nichts	8 B
8 B	1	0	1	Subtraktion	15 B

Tabelle: 4.2: Beispiel für Methode 2 mit dem Skalar $k = 15$

Der Algorithmus von Methode 2 konnte hier nur kurz in seiner Funktion skizziert werden. Für eine detailliertere Beschreibung muss auf die Literatur [KNU98,GOR98] verwiesen werden.

Die Verwendung von h kann auch ohne explizites Ausrechnen direkt von k abgeleitet werden, die Methode wird im folgenden Abschnitt beschrieben.

Ternäre Darstellung und Methode 2

Durch die ternäre Darstellung von k mit Hilfe der Variablen h, ist es möglich, nicht nur von aufeinander folgenden Nullen zu profitieren, sondern auch von aufeinander folgenden Einsen:

Eine Eins gefolgt von k Nullen entspricht der Zahl 2^k, nun stelle man sich einen Block von k aufeinander folgenden Einsen vor:

$$\underbrace{11\ldots11}_{k} \triangleq \sum_{i=0}^{k-1} 2^i = 2^k - 1 \qquad (4.12)$$

Dies entspricht der Zahl 2^k -1. Diese Zahl kann durch k Verdopplungen und einer Subtraktion, bei der Verwendung der Methode 2, berechnet werden.
Die binäre Methode 1 benötigt hierfür k Verdopplungen und k Additionen, die Einsparungen sind beträchtlich.
Diese Methode hat auch einen Nachteil und zwar ist eine alternierende Folge von Einsen und Nullen der ungünstigste Fall: Das liegt daran, dass bei der Methode 2 nun abwechselnd addiert und subtrahiert wird, während bei der Methode 1 lediglich alle zwei Stellen eine Addition ausgeführt wird.

Insgesamt betrachtet überwiegt der Gewinn bei Methode 2 die entstehenden Verluste bei alternierenden Folgen (z.B. 1010...) jedoch deutlich. Die Methode demonstriert ein einfaches Verfahren, bei dem Additions-Subtraktionsketten eingesetzt werden.
Die Beschleunigung wird durch einem Mehraufwand erkauft: Denn es muss zunächst die ternäre Darstellung von *k* berechnet werden. Hinzu kommt, dass die Optimierung neben der Addition auch die Subtraktion benötigt. Die Subtraktion wurde jedoch zuvor nicht benötigt.

Für die Implementierung ist nicht notwendig, dass der Skalar doppelt kodiert vorliegen muss, wie es in Methode 1, durch *k* und *h*, verlangt wurde. Es ist möglich aus jeweils zwei benachbarten Stellen der binären Repräsentation von *k*, die gleichen Informationen zu erhalten. Hierdurch kann auf ein zusätzliches Register verzichtet werden. Diese Möglichkeit der Skalarmultiplikation, wird von **Methode 3** ausgenutzt.
Die Ermittlung erfolgt mit Hilfe von Tab. 4.3. Eingabe sind 2 Bits von *k* und Ausgabe ist die jeweilig Auszuführende Operation:

k_i k_{i-1}	**Operation**	**Beschreibung**
0 0	Verdoppeln	0-Folge
0 1	Verdoppeln und Addieren	Ende 1-Folge
1 0	Verdoppeln und Subtrahieren	Ende 1-Folge
1 1	Verdoppeln	1-Folge

Tabelle 4.3: Operationsbeschreibung bei ternärer Zahlendarstellung

Für eine Implementierung muss zusätzlich beachtet werden, dass auf das letzte Bit eine 0 folgt, d.h. $k_{-1} = 0$ gilt. Hier sei die Umformulierung der Methode 2 explizit angegeben:

Methode 3: Methode 2 ohne Umkodierung (*addition-subtraction chains*)

Berechnung von: $k \cdot B,\ k < 2^n,\ k \in \mathbb{N}, B \in E$ über $GF(2^n),\ k = \sum_{i=0}^{l} k_i \cdot 2^i$

mit $k_l = 1,\ k_i \in GF(2)$

1. E ← 0
2. **if** $k = 0$ **stop with** E
3. E ← B
4. **for** $i \leftarrow l$-1 **downto** 1
5. E ←2 E (*Punktverdopplung auf elliptischer Kurve*)
6. **if** ($k_i = 0$) and ($k_{i-1} = 1$) **then** E ←E+B (*Punktaddition auf ell. Kurve*)
7. **if** ($k_i = 1$) and ($k_{i-1} = 0$) **then** E ←E-B (*Punktaddition auf ell. Kurve*)
8. **next** *i*
9. E ←2 E
10. **if** ($k_0 = 1$) **then** E ←E-B
11. **stop with** E

Zur Veranschaulichung kann obiges Beispiel (Tab. 4.2) zur Methode 2 verwendet werden.

An dieser Stelle sei noch einmal darauf hingewiesen, dass die Subtraktion bei elliptischen Kurven ohne besonderen Mehraufwand im Vergleich zur Addition erfolgen kann. Der Mehraufwand für eine Subtraktion statt einer Addition beträgt exakt eine Addition im $GF(2^k)$ (vgl. (2.40)).
Ein analoger Fall, bei dem auch Additionsketten verwendet werden, liegt bei modularer Exponentiation vor. Verwendet man die modulare Exponentiation über Restklassenringe, so entspricht der Addition eine Multiplikation und der Subtraktion eine Division.
Nun ist aber die Division in Restklassenringen sehr viel (um Größenordnungen) aufwendiger als die Multiplikation.
Dies ist ein klarer Vorteil von elliptischen Kurven gegenüber anderen klassischen Verfahren wie beispielsweise RSA.

Weitere Akzelerationen der Skalarmultiplikation

Eine weitere Möglichkeit die Skalarmultiplikation zu beschleunigen, besteht in der Verallgemeinerung der genannten Verfahren. Man kann statt der Komposition zur Basis $1 \cdot B$ auch die Basis $2^i \cdot B$ verwenden. Die Basis besteht dann aus den einzelnen 2-er-Potenzen von B:

$$\{B \cdot 2^0, B \cdot 2^1, B \cdot 2^2, \ldots, B \cdot 2^m\} \tag{4.13}$$

wobei die Anzahl der Basiselemente durch die Ordnung des Basispunktes $\#(B)$ (zur Ordnung eines Punktes s. (3.2)) begrenzt ist:

$$\sum_{i=0}^{m} 2^i = 2^{m+1} - 1 < \#(B) \tag{4.14}$$

Die Ordnung eines Punktes auf einer elliptischen Kurve gibt die Anzahl der Punkte in der Untergruppe an. Insbesondere gilt folgende Skalarmultiplikation:

$$\#(B) \cdot B = O \quad \text{mit } \#(B) \in \mathrm{N} \tag{4.15}$$

(4.14) gibt den größtmöglich darstellbaren Skalarwert, bei Verwendung der Basis (4.13), wieder. Der größtmögliche Skalar muss jedoch kleiner als die Ordnung des Basispunktes sein, weshalb die Anzahl der Basiselemente durch m+1 begrenzt ist.

Der Gebrauch dieser Basis setzt voraus, dass man sehr oft Vielfache eines festen Punktes benötigt. Dies ist bei kryptographischen Verfahren, die u. U. längere Zeit mit einem Basispunkt B arbeiten, der Fall.
Die Skalarmultiplikation gestaltet sich nun mit dieser Basis äußerst einfach: Man berechnet alle 2-er Potenzen im Voraus und addiert diese bei der eigentlichen Berechnung entsprechend der gesetzten Bits des Skalars auf:

$$B \cdot 2^j = P_j \quad 0 \le j < m$$

$$k \cdot B = \sum_{i=0}^{l} k_i \cdot P_i : k_l = 1,\ k_i \in GF(2) \tag{4.16}$$

Methode 3: Skalarmultiplikation mit vorberechneten Additions-Subtraktionsketten

Berechnung von: $k \cdot B,\ k < 2^n,\ k \in \mathbb{N}, B \in E$ über $GF(2^n),\ k = \sum_{i=0}^{l} k_i \cdot 2^i$

mit $k_l = 1,\ k_i \in GF(2)$

1. E ← 0
2. Vorberechnung der P_i (s. o.)
3. **if** $k = 0$ **then stop with** E
4. **for** $i \leftarrow\ l$ **downto** 0
5. **if** $k_i = 1$ **then** E ←E+P_i (*Punktaddition auf ell. Kurve*)
6. **next** i
7. **stop with** E

Zusammenfassend bleibt festzuhalten: Soll die optimierte Skalarmultiplikation (Methode 2) verwendet werden, bedarf es eines weiteren Registers für die 3k-Komponente sowie der Subtraktionsanweisung für Punkte. Für die zuletzt vorgestellte Methode wird enormer Speicherplatz benötigt, da hier entsprechend der Körpergröße n zugleich n Punkte der Kurve gespeichert und zuvor berechnet werden müssen. Für Anwendungen mit geringem Speicherplatz scheidet diese Methode somit fast automatisch aus. Lediglich für Anwendungen mit sehr kleinen Körpergrößen oder für Anwendungen, bei denen ein großer Speicher zur Verfügung steht, ist diese Methode anwendbar.

4.5 Festlegung der Wortbreite

Bei Verwendung von Elementen des Körpers GF(2^k) in Rechenregistern, werden deutlich größere Wortbreiten als bei herkömmlichen Mikroprozessoren, wie beispielsweise 8, 16, 32 oder 64 Bit benötigt. In dem Akzelerator müssen derart große Zahlen, mit der Wortbreite von k Bit, gespeichert und verarbeitet werden. Die Speicherung und Verarbeitung erfolgt je nach Implementierung in Registern oder einer Speicherbank.
Der benötigte Platz für Register mit hoher Wortbreite und insbesondere deren Verdrahtungsplatz ist u. U. zu hoch. Um dies zu vermeiden, können kleinere Register mit geringerer Wortbreite verwendet werden. Allerdings müssen hierdurch mehr Operationen vorgenommen werden. Dies sei am Beispiel der Addition für natürliche Zahlen verdeutlicht:

Bei der Aufteilung einer (vollen) Addition (k Bits) in Additionen halber oder geringerer Wortbreite, müssen zwei Additionen der Teile mit Berücksichtigung des Übertrags vorgenommen werden.

Es ist zwar prinzipiell möglich, Register mit k Bit Wortbreite für alle Operationen bereitzuhalten und das Rechenwerk entsprechend anzupassen, jedoch sind die Kosten für den Mehraufwand an Fläche nicht angemessen. Um dennoch mit diesen Wortbreiten rechnen zu können, müssen die Zahlen in mehrere Einheiten aufgeteilt werden, dieser Vorgang sei im Folgenden **Blockung** genannt.

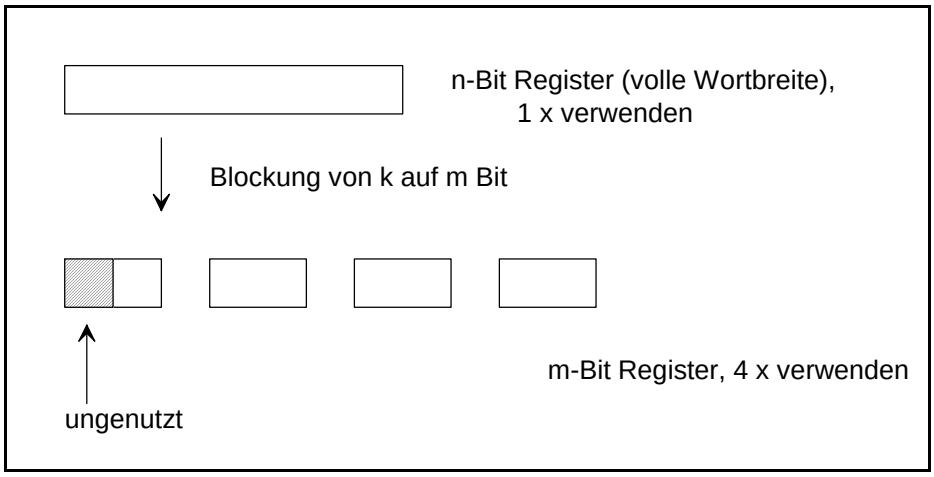

Abbildung 4.3: Veranschaulichung der Blockung von n Bit auf $m = \left\lceil \frac{n}{k} \right\rceil$ mit $k < n$

Die Auswirkungen der Verdrahtung wurden bereits angerissen, sie spiegeln jedoch nur einen Teil der Problematik wieder.
Genau entgegengesetzt verhält es sich mit dem Aufwand für die Teiloperationen. Bei geringerer Wortbreite muss das Teilregister (k/n) mehrfach verwendet werden (vgl. Abb. 4.3), um alle Teile des ehemaligen Gesamtregisters zu bearbeiten. Angenommen, eine Addition von 30 Bit-Zahlen könnte mit Registern der Wortbreite in einem Schritt ausgeführt werden. Bei einer Blockung auf k = 4, hätten die Register eine Wortbreite von m = 8 Bit. Mit Registern der Wortbreite 8 Bit benötigt man 4 Schritte a 8 Bit um zwei 30 Bit-Zahlen zu addieren. Beim letzten Schritt benötigt man statt der 8 Bit jedoch nur 6 Bit. Dieser ungenutzte Bereich ist in Abb. 4.3 schraffiert dargestellt.

Diese sich gegenseitig ausschließenden Faktoren der schnellen Ausführung und der geringen Fläche, müssen für den Einzelfall festgelegt werden. Nicht zuletzt ergeben sich auch aus den Randparametern der Praxis, wie der Leistungsaufnahme, der maximalen Taktfrequenz und der maximalen Layoutfläche und der gesamten Ausführungszeit, entsprechende Festlegungen.

Die Auswirkung von Wortbreiten auf das fertige Chiplayout sind nur schwer analytisch bestimmbar. Dies liegt daran, dass eine Vielzahl von theoretisch schwierigen Problemen bei dem Vorgang der Designerstellung bewältigt werden müssen: Hierzu gehört die Transformation von der Hardwarebeschreibung auf die Logikebene, die Abbildung auf Standardzellen, die Platzierung, die Global- und Kanalverdrahtung.
Aussagen über die Auswirkungen können hier nur noch in Form von Abschätzungen und anhand von Messungen an mehreren Designs mit unterschiedlichen Wortbreiten vorgenommen werden.
Anders verhält es sich bei der Betrachtung der Berechnungsaufteilung: Eine Aufteilung der Berechnung anhand der Wortgröße ist auch analytisch möglich.
Das Problem besteht darin, eine Berechnung für eine große Zahl, mit n Bits, auf mehrere Teilberechnungen mit einer Wortbreite von k Bits (dabei gelte $k < n$), zu verteilen. Man kann es auch wie folgt formulieren: Ein großes Problem in mehrere kleinere zu zerteilen und die Lösungen der Teilprobleme anschließend zur Lösung des Gesamtproblems zusammensetzen.

Im Folgenden soll das Problem der kleineren Wortbreite veranschaulicht werden:

Für die Betrachtung im Folgenden sei n die Registerbreite der Körperelemente und k die Größe der Hardwareregister bzw. der Wortbreite. Dann benötigt man m Einzeloperationen:

$$m = \left\lceil \frac{n}{k} \right\rceil \quad \text{mit} \quad k < n \tag{4.17}$$

Je nachdem, ob n ein Vielfaches von k ist, entsteht ein Overhead von nicht benötigten Operationen und ungenutzten Platz.
Der Overhead wird durch die Differenz zum nächsten Vielfachen erzeugt, sei beispielsweise

$$n = qk + r \quad \text{und } r > 0, \quad m \text{ ergibt sich zu} \quad m = q + 1 \tag{4.18}$$

Dann können q Bearbeitungsschritte mit voller Auslastung, da alle Bits Informationen enthalten, behandelt werden. Hiernach folgt ein weiterer Schritt, in dem lediglich r Bits benötigt werden. Die Wortbreite beträgt jedoch auch hier n Bits, weshalb der Aufwand für die n-r Bits als Overhead betrachtet werden kann.
Genau genommen kommt noch ein weiterer Overhead durch zu behandelnde Überträge und zusätzliche Steuerleitungen hinzu.

Der Sachverhalt soll an dem Beispiel der ganzzahligen Multiplikation verdeutlicht werden: Hier liege ein quadratischer Aufwand vor, d. h. n^2 Operationen werden für die Ausführung benötigt. Der Aufwand bei der Blockung von Registern ändert sich nach obiger Formel auf

$$n \cdot \left\lceil \frac{n}{k} \right\rceil \tag{4.19}$$

Abstrahiert man nun von der zugrunde liegenden absoluten Größe n, und betrachtet nur eine Annäherung der Wortbreite bis hin zu 100%, dann wird dieses Verhalten offensichtlich durch eine Hyperbel mit Treppen beschrieben.
Um die Formel zu veranschaulichen ist am Beispiel für n = 200 die Anzahl der Operationen in folgender Abbildung 4.4 dargestellt.

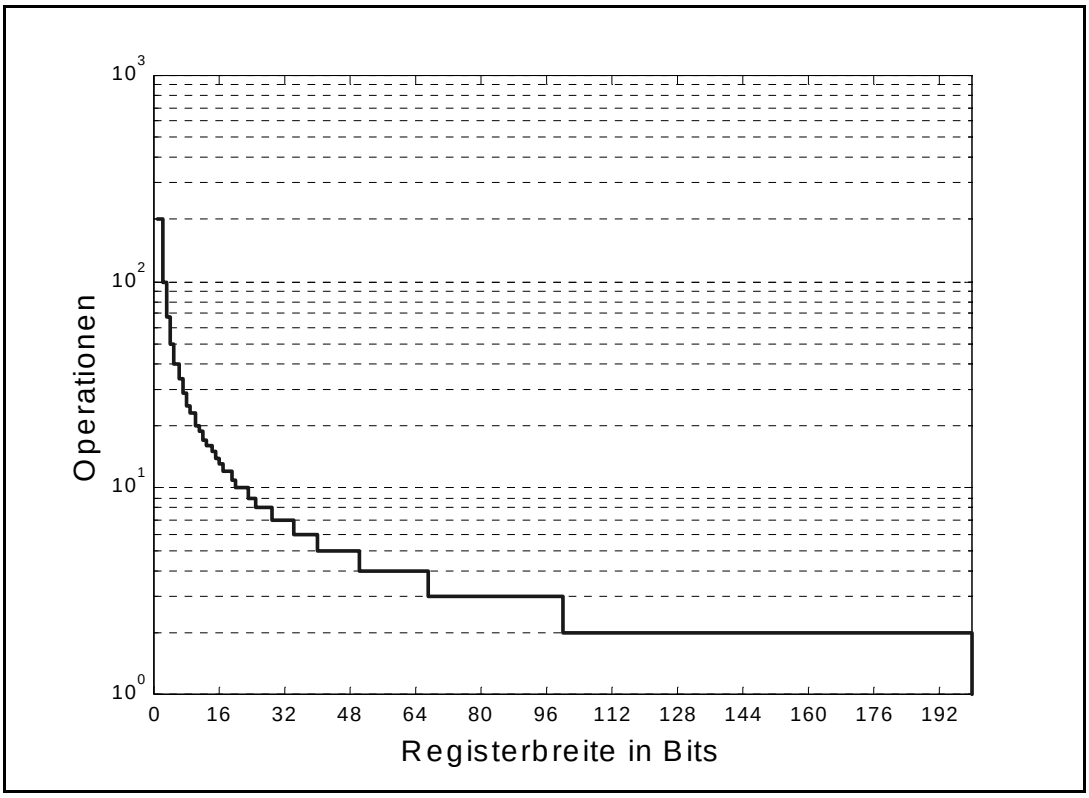

Abbildung 4.4: Operationszahl bei der Blockung

Diese Grafik verdeutlicht die praktische Relevanz der Wortbreite. Vergrößerungen der Wortbreite im unteren Bereich bringen sehr große Verbesserungen. Steigerungen bzw. Vergrößerungen der Wortbreite nach dem „Knick“ (bei ca. 48 Bit Registerbreite) haben wenig Einfluss auf die Effizienz. Genau genommen sinkt der Grad der Verbesserung mit einer Erhöhung der Wortbreite stetig. Dies bedeutet, dass im Einzelfall sehr genau geprüft werden muss, wo die Grenze liegt bzw. ein Kompromiss eingegangen werden muss.

4.6 Körper-Addition

Die Addition wird durch stellenweise XOR-Verknüpfung realisiert. Es sei noch einmal ausdrücklich darauf hingewiesen, dass die Addition ohne Überläufe für alle Stellen der Summanden erfolgt. Die Addition ist also schnell, in einem Schritt, möglich.
Hieraus und mit der bereits oben angesprochenen Möglichkeit der Blockung ergeben sich prinzipiell drei verschiedene Varianten:

1. Addition **bitsequentiell**, d. h. Stellenweise
2. Addition **geblockt**, d. h. Register für Register
3. und **parallele** Addition, d. h. mit Registern der Breite *n*.

Der Aufwand beträgt:

Art der Addition	Schritte	Registerbreite
sequentiell	n	1
geblockt	$\lceil n/k \rceil$	k mit $k < n$
parallel	1	n

Tabelle 4.4: Aufwandsvergleich der Addition bei unterschiedlicher Wortbreite

4.7 Körper-Multiplikation

Die Multiplikation ist komplizierter als die Addition und es existiert eine Vielzahl von Multiplikationsalgorithmen. Hier kann kein Überblick über diese Vielfalt gegeben werden, da dies den Rahmen dieser Arbeit bei weitem sprengen würde. Eine gute Übersicht bietet das Kapitel „How Fast Can We Multiply?“ in [KNU98]:
Neben den hier beschriebenen Methoden existieren unter anderen Ansätze über Restklassenmodule nach dem Chinesischem Restklassensatz und der schnellen Fouriertransformation. Beide Ansätze flossen in einer Verfeinerung zu dem derzeit schnellsten bekannten Multiplikationsalgorithmus ein. Der Algorithmus stammt von Schönhage und Strassen [SS71], ist aber in der Realisierung für den in dieser Arbeit betrachteten Spezialfall viel zu aufwendig und hat erst bei entsprechend großen Zahlen eine bessere Laufzeit.

In dieser Arbeit werden zwei Multiplikationsmethoden miteinander verglichen, beide sind mit vergleichbar geringem Aufwand zu implementieren. Dies ist die Schulmethode und der Ansatz von Karatsuba und Ofman [KO63]. Der Ansatz wird in Kapitel 4.9 skizziert.
Interessanterweise rufen die verschiedenen Optimierungen für die Multiplikation der natürlichen Zahlen bei den endlichen Körpern keine vergleichbaren Verbesserungen hervor. Diese Tatsache wird kurz nach der Behandlung der eigentlichen Multiplikation und Reduktion diskutiert.

Multiplikation und Reduktion

Bei der Multiplikation zweier natürlicher k-Bit-Zahlen entsteht maximal ein $2k$-Bit Ergebnis, bei endlichen Körpern gibt es keine Überträge, das Ergebnis hat demnach maximal $2k$-1 Bits:

Beispiel für natürliche Zahlen: $7 \cdot 7 = 49$, Bitbreiten: [3]+[3] = [6]
Beispiel für Polynome: $(x^2+x+1)\cdot(x^2+x+1) = x^4+2x^3+3x^2+2x+1$, Bitbreiten [3]+[3]=[5]

Nun kommt hinzu, dass der endliche Körper begrenzt ist. Nimmt man n als maximale Stellenzahl bei $GF(2^n)$, so muss das Ergebnis auf diese maximale Größe reduziert werden.
Die Multiplikation ist durch einfache Ausmultiplizierung möglich. Im Folgenden ist die Formel für die Koeffizienten der Berechnung $c = a \times b$ beschrieben:

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} \text{ mit } 0 \le i \le \deg(a) + \deg(b) \tag{4.20}$$

$deg(p)$ ist der Grad des Polynoms p

Die Formel (4.7.1) beschreibt die Konvolution bei der Multiplikation zweier Polynome, bzw. es werden dort die Koeffizienten des Ergebnispolynoms angegeben. Zur Veranschaulichung sei die Multiplikation anhand eines Beispiels demonstriert:

$a = x^3 + x + 1,$ mit den Koeffizienten: $a_0 = 1\ a_1 = 1\ a_2 = 0\ a_3 = 1\ a_4 = 0\ a_5 = 0$
$b = x^2 + 1,$ mit den Koeffizienten: $b_0 = 1\ b_1 = 0\ b_2 = 1\ b_3 = 0\ b_4 = 0\ b_5 = 0$

4.7.1 ergibt nun:

$$
\begin{aligned}
c_0 &= a_0 b_0 = 1 \\
c_1 &= a_0 b_1 + a_1 b_0 = 1 \\
c_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0 = 1 \\
c_3 &= a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0 = 1 + 1 = 0 \\
c_4 &= a_0 b_4 + a_1 b_3 + a_2 b_2 + a_3 b_1 + a_4 b_0 = 0 \\
c_5 &= a_0 b_5 + a_1 b_4 + a_2 b_3 + a_3 b_2 + a_4 b_1 + a_5 b_0 = 1
\end{aligned}
$$

Das Resultat lautet: $c = x^5 + x^2 + x + 1$

Nach der Berechnung des Produktes muss dieses reduziert werden, d. h. der Rest modulo dem irreduziblen Polynom ermittelt werden. Eine einfache Realisierung besteht darin, durch das irreduzible Polynom zu dividieren und den Rest zu verwenden, hierbei kann man sich der Schulmethode bedienen.
Man darf nicht vergessen, dass bei diesem Ansatz zunächst ein doppelt so großes Zwischenergebnis im Vergleich zum endgültigen Ergebnis ermittelt wird. Um dies zu vermeiden, wird bereits vor dem vorliegen des endgültigen Ergebnisses reduziert. So können die Zwischenergebnisse durch eine gewisse Länge begrenzt werden.

Kombination der Multiplikation und Reduktion

Es ist wünschenswert und üblich, ohne das fast doppelt so große Zwischenergebnis nach der Multiplikation auszukommen, schließlich wird nur das reduzierte Ergebnis benötigt. Man kann die beiden Vorgänge der Multiplikation und Reduktion auch kombinieren und erhält ein effizienteres Verfahren. Diese Vorgehensweise wird auch bei der modularen Exponentiation verwendet.

Algorithmus 1: Kombinierte Multiplikation

I' bezeichnet das irreduzible Polynom ohne den höchsten Koeffizienten. Der höchste Koeffizient ist nicht mehr notwendig, da das höchste Bit stets gesetzt ist und das höchste Bit des Zwischenregisters (*t*) sich bei der Reduktion bereits im Übertragsregister (*CarryFlag*) befindet.
t bezeichnet das Ergebnis- und Zwischenregister
deg() bezeichnet den Grad des Polynoms
Shift() ist die Verschiebeoperation nach links (höhere Indices) und als niedrigstes Bit wird eine null eingefügt. Als Rückgabewert dient das zuvor höchste Bit des Arguments.

1. $t \leftarrow 0$
2. for $i \leftarrow deg(I)$-1 **downto** 0
3. *CarryFlag* ← *Shift(t)* Multiplikation mit *x*, Carry ist der Index *deg(I)*
4. **if** *CarryFlag* = 1 **then** $t \leftarrow t+I'$ Reduktionsschritt
5. **if** $b_i=1$ **then** $t \leftarrow t+a$ Addition des Multiplikanden
6. next *i*
7. stop with *t*

Die Verschiebung der Koeffizienten von *t* erfolgt nur bis zum höchsten Indexfeld für den Körper, welches gleich dem Grad des irreduziblen Polynoms ist. Das höchste Indexfeld wird als CarryFlag verwendet (vgl. Abb. 4.5).
Das Vorhandensein eines Carry veranlasst dann in der folgenden Zeile zur Reduktion, wobei hier *I'*, also *I* ohne den höchsten Koeffizienten verwendet wird, der sich ja nun bereits im Carry-Flag befindet. Im letzten Schritt erfolgt die bekannte stellenweise Addition.
Bei dieser Variante wird kein Platz für das Multiplikationszwischenergebnis benötigt.

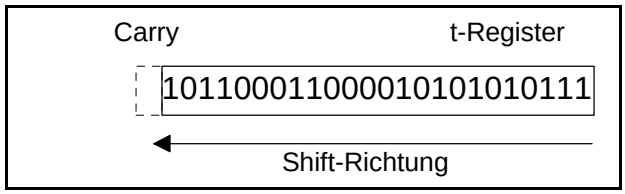

Abbildung 4.5: Übergang vom t-Register in das CarryFlag

Beispiel zum Algorithmus "Kombinierte Multiplikation":

Es sollen $a = (x^3+x+1)$ und $b = (x^2+1)$ miteinander, im GF(2^4), multipliziert werden. Das irreduzible Polynom *I* sei x^4+x+1, das Ergebnis ist 1:

***t*-Register**	***i***	**nach Shift**	**Carry**	**nach Reduktion**	b_i	**nach Addition**
(0000)	3	(0000)	0	-	0	(0000)
(0000)	2	(0000)	0	-	1	(1011)
(1011)	1	(0110)	1	(0101)	0	(0101)
(0101)	0	(1010)	0	-	1	(0001)

Tabelle 4.5: Beispiel für die Multiplikation, höchste Koeffizienten links

Besonderheiten bei speziellen irreduziblen Polynomen

Die Verwendung bestimmter irreduzibler Polynome kann den Berechnungsaufwand verringern. Zeitliche Beschleunigungen sind von größerem Interesse als Platzverringerungen. Zur Platzverringerung zählt die bereits zuvor besprochene Variante der kombinierten Multiplikation und Reduktion in einem rückgekoppelten Schieberegister.
In der Literatur werden z. T. irreduzible Polynome mit dünner Koeffizienten-Besetzung favorisiert. Es handelt sich hierbei um Polynome, bei denen nur sehr wenig Koeffizienten mit '1' und die restlichen mit '0' besetzt sind:

Es handelt sich um so genannte **Tri-** und **Pentanome**, sie haben die Formen:

$$x^k + x^l + 1 \qquad x^k + x^{l_1} + x^{l_2} + x^{l_3} + 1 \qquad k > l,\ l_1,\ l_2,\ l_3 > 1$$

Trinome existieren nicht für jeden Erweiterungsgrad des Körpers. Bei Pentanomen ist dies nicht problematisch, es existiert für jeden Grad mindestens ein entsprechendes irreduzibles Polynom.
Die Vorteile dieser dünn besetzten irreduziblen Polynome sind:

1. Es wird deutlich weniger Platz für die Kodierung derartiger Polynome benötigt. Hierbei kann berücksichtigt werden, dass die ‘1’ stets gesetzt ist.
2. Die Reduktionsoperationen können vereinfacht werden. Hierauf wird im Folgenden noch kurz eingegangen.

Nachteile wurden bis jetzt noch nicht fundiert begründet - allerdings soll hier nicht verschwiegen werden, dass bereits mehrfach Sicherheitsbedenken gegen diese irreduziblen Polynome geäußert wurden.
Zum Abschluss dieses Exkurses soll ein Sonderfall der Trinome betrachtet werden:

Für Körper der Form:

$$\mathrm{GF}(2^k) \text{ mit } k = 2 \cdot 3^l \text{ für } l \geqslant 0, l \in \mathbb{N}$$

sind Polynome der Form

$$x^k + x^{k/2} + 1 \tag{4.21}$$

stets irreduzibel. Das Reduktions-Verfahren wird in [CS97] und [vL91] beschrieben. Diese Form kann genutzt werden, um die Reduktion auf drei Additionen zurückzuführen:

Für zwei Zahlen $a,\ b \in \mathrm{GF}(2^k)$ sei das Produkt $c = a \cdot b$ mit $c \in \mathrm{GF}(2^{2k-1})$, die Koeffizienten von c seien mit c_i wie folgt gegeben:

$$c = \sum_{i=0}^{2k-1} c_i x^i \tag{4.22}$$

Die Reduktion kann durch Addition der Bitfolgen S_1, S_2, S_3 und S_4 erfolgen:

$$c' = c \mod (x^k + x^{k/2} + 1) = S_1 + (1 + x^{k/2})S_2 + S_3 \tag{4.23}$$

mit folgender Bitanordnung:

$$S_1 = \sum_{i=0}^{k-1} c_i x^i$$
$$S_2 = \sum_{i=0}^{k/2-1} c_{i+k} x^i$$
$$S_3 = \sum_{i=0}^{k/2-2} c_{i+\frac{3k}{2}} x^i$$

(4.24)

Die Verschiebungen sind in der folgenden Skizze grafisch als Bitdarstellung aufbereitet:

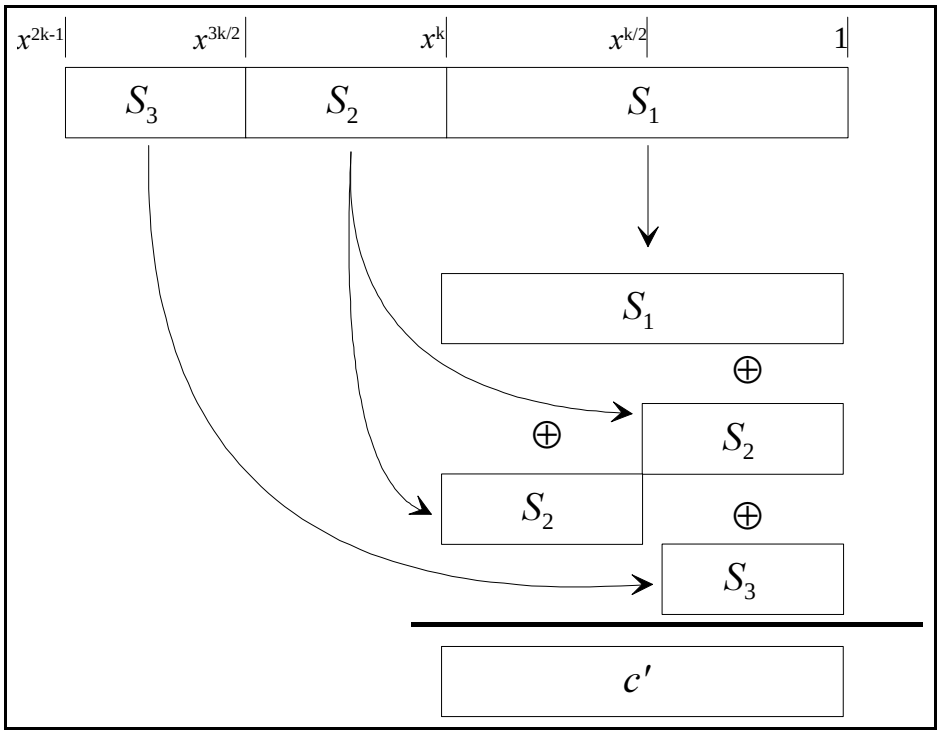

Abbildung 4.6: Reduktion durch 3 Additionen

Die S_i entsprechen Bitblöcken von *c'*.

Einbindung des Reduktions-Verfahrens

Obiges Verfahren benötigt zunächst das unreduzierte Multiplikationsergebnis. Dieses kann parallel mit optimierten Multiplikationsansätzen berechnet werden. Ein paralleles Verfahren wird weiter unten (Kap. 4.9) vorgestellt. Für die Reduktion im allgemeinen Fall, d. h. ohne die besondere Struktur des irreduziblen Polynoms, sind *m*-1 Schritte notwendig. Mit der Struktur des irreduziblen Polynoms reduziert sich der Vorgang auf drei Additionen, nach dem oben gezeigten Schema (Abb. 4.6).
Sieht man von dem zusätzlichen benötigten Platz für das Multiplikationsergebnis ab, so liegt eine Reduktion der Berechnungszeit von $O(k)$ auf $O(1)$ vor. Diese Aussagen sind unabhängig davon, ob eine Hardware- oder Softwarerealisierung vorgenommen wird.

Diese Vereinfachung erlaubt eine wesentliche Beschleunigung der Reduktion, jedoch sei noch einmal daran erinnert, dass es nur mit den irreduziblen Polynomen der oben genannten Form und nur für bestimmte Körpergrade realisierbar ist.

4.8 Aufwand der Multiplikation

Ausgehend von den Möglichkeiten der Additionsimplementierung, ergeben sich nun drei Resultate für die kombinierte Multiplikation und Reduktion:

Art der Addition	Schritte	Laden/ Schreiben	Registerbreite
a) sequentiell	$3n^2$	n	1
b) geblockt	$3n \cdot \left\lceil \frac{n}{k} \right\rceil$	$\left\lceil \frac{n}{k} \right\rceil$	k mit $k < n$
c) parallel	$3n$	1	n

Tabelle 4.6: Aufwandsübersicht der Multiplikation in Abhängigkeit der Wortbreite

Erläuterung zur Tabelle:

Die angegebene Schrittzahl beschreibt den Aufwand der Multiplikation in Abhängigkeit der verwendeten Addition. Hierbei werden Transporte aus Zwischenspeichern *nicht* berücksichtigt. Der Aufwand für das Laden und Schreiben jeweils eines Registers ist in der folgenden Spalte angegeben.

a) Bei der sequentiellen Addition müssen jeweils alle Operanden bitsequentiell geladen werden, hierfür sind jedes Mal n-Schritte notwendig. Für die Berechnung wird quadratischer Aufwand benötigt, müssen die Operanden jedoch vor und nach jeder Operation sequentiell geladen und gespeichert werden, so erhöht sich die Konstante vor dem quadratischen Term entsprechend.
Dieser Fall ist für die Praxis jedoch unrealistisch, da die Verwendung von sequentiellem Speicher oftmals nicht notwendig ist, viel wahrscheinlicher ist die Verwendung von Speicher mit Wort-Zugriff. Liegt dieser Fall vor, so wird man aber auch Rechenoperationen auf Wörtern auszuführen können. Hier liegt dann bereits der Fall b vor.

b) Bei der geblockten Verarbeitung liegt immer noch ein quadratischer Aufwand für die Berechnung vor. Dieser Fall dürfte in der Praxis oftmals vorliegen, insbesondere bei Universalrechnern. Die tatsächlich benötigten Lade- und Schreiboperationen hängen hier von den zur Verfügung stehenden Registern ab, eine Lade- bzw. Schreiboperation benötigt $\left\lceil \frac{n}{k} \right\rceil$ Schritte.

c) Parallele Verarbeitung, zeigt erwartungsgemäß eine drastische Senkung der Verarbeitungsschritte. Die Transportschritte sind jedoch für die Praxis zum Teil unrealistisch, da hier davon ausgegangen wird, das die Variablen in einem Schritt vorliegen. Realistischer ist es davon auszugehen, dass der Speicher eine Wort-Organisation hat, dann ist der Aufwand für Fall b zu verwenden.

Die Tabelle zeigt ein erstaunlich günstiges Resultat bei der parallelen Ausführung und zwar wird die Multiplikation in linearer Zeit berechnet. Das liegt natürlich an der konstanten Zeit für eine Addition, die als Grundlage für die Multiplikation herangezogen wurde. Dieses gute Resultat, wird durch drei n-Bit-Register erkauft. Bei der Implementierung in Hardware wird sich dieses Merkmal deutlich im Platzbedarf von anderen Lösungen hervorheben.

4.9 Optimierte Multiplikationsansätze

Wie bereits angekündigt, wird nun die Karatsuba-Ofman-Multiplikationsmethode für endliche Körper adaptiert. Die Methode kann mit dem *divide-et-impera*-Prinzip verwendet werden, um das Problem der Multiplikation für *2n*-Bit-Zahlen auf mehrere Multiplikationen für *n*-Bit-Zahlen umzusetzen. Diese Teilergebnisse müssen dann zu einem Gesamtergebnis zusammengesetzt werden, hierbei kommt man mit weniger Operationen, als bei dem normalen Berechnung aus.

Die Multiplikation kann für Teile von Zahlen aufgeteilt werden. Die Variablen U und V sollen jeweils einen Multiplikanden repräsentieren, U_0 bzw. V_0 die untere Hälfte (low word) und U_1 bzw. V_1 die obere Hälfte (high word):

$$U = 2^n U_1 + U_0, V = 2^n V_1 + V_0 \tag{4.25}$$

Hierbei seien die indizierten Teile jeweils gleich groß und mögen je *n* Bits enthalten. Somit hat U 2*n*-Bits. Die direkte Ausmultiplizierung ergibt vier Teilmultiplikationen:

$$UV = 2^{2n} U_1 V_1 + 2^n (U_1 V_0 + U_0 V_1) + U_0 V_0 \tag{4.26}$$

Der Ansatz von Karatsuba-Ofman besteht nun darin, mit lediglich drei Multiplikationen auszukommen. Die hier präsentierte Fassung von Karatsuba und Ofman ist nach D. E. Knuth vereinfacht worden, beschreibt jedoch den gleichen Ansatz:

$$UV = (2^{2n} + 2^n) U_1 V_1 + 2^n (U_1 - U_0)(V_0 - V_1) + (2^n + 1) U_0 V_0 \tag{4.27}$$

Vergleich zu (4.26):

$$\begin{aligned} &\Leftrightarrow 2^{2n} U_1 V_1 + 2^n (U_1 V_1 + (U_1 - U_0)(V_0 - V_1) + U_0 V_0) + U_0 V_0 \\ &\Leftrightarrow 2^{2n} U_1 V_1 + 2^n (U_1 V_0 + U_0 V_1) + U_0 V_0 \end{aligned} \tag{4.28}$$

Die Multiplikation wurde in **drei** Multiplikationen **halber Wortbreite**, **vier** Additionen, zwei Additionen **halber Wortbreite**, sowie **zwei** Stellenverschiebungen (2er-Potenzmultiplikationen) transformiert.

Die Laufzeit der Multiplikation beträgt nach einer Abschätzung in [KNU98], $O(n^{\log 3})$.

Die Aufwandsreduktion lässt sich einfach für die Teilmultiplikationen durchführen, problematisch bleibt jedoch die anschließende Reduktion des Ergebnisses. Allein die Reduktion benötigt jedoch so viel Zeit wie die vorige Multiplikation (vgl. Algorithmus 1, Kap. 4.7). Hieraus folgt, dass das Konzept sich nicht für die Multiplikation mit endlichen Körpern verwenden lässt. Die vielversprechenste Optimierung besteht also in der Verwendung von Registern voller Breite, um somit die Multiplikation in linearer Zeit bewerkstelligen zu können.

5 Bewertung der Realisierungsalternativen

In diesem Kapitel werden die bisher vorgestellten Berechnungsalternativen bewertet. Bei der Behandlung der einzelnen Alternativen im vorigen Kapitel wurden bereits ihre Vorteile angesprochen, eine Übersicht und auch eine Abwägung ihrer Vor- und Nachteile kann jedoch erst nach der Vorstellung sämtlicher Alternativen erfolgen.
Im Folgenden werden:

- die Vorteile der GF(2^k) Körper gegenüber den GF(p) besprochen,
- die Effizienz des projektiven Modells gegenüber dem affinen und dem rationalen Modell,
- sowie die Vorteile einzelner Skalarmultiplikationsberechnungen.

5.1 GF(2^k) versus GF(p)

In der Diplomarbeit von Bohnsack [BOH97], welche elliptische Kurven über GF(p) behandelte, wurde festgestellt, dass die Reduktion modulo der Körpercharakteristik p sehr viel Berechnungsresourcen benötigt.
Die Reduktion wurde bereits oben beschrieben. Die Problematik sei hier nochmals verkürzt wiedergegeben:

GF(p):

> Um eine Reduktion vornehmen zu können, muss zunächst mit einem Vergleich geprüft werden, ob überhaupt reduziert werden muss, d. h. ein Test ob die Zahl größer als die Körpercharakteristik p ist. Wenn dies der Fall ist, so muss der ganzzahlige Rest bei einer Division (**mod**-Operation) berechnet werden.

GF(2^k):

> Der Vergleich reduziert sich darauf, ob das Bit an höchster Stelle, dem Grad des irreduziblen Polynoms, gesetzt ist. Der Reduktionsvorgang kann durch die komponentenweise Addition (bitweises XOR) durchgeführt werden. Beide Operationen lassen sich in Hardware einfach realisieren und sind sehr schnell in der Ausführung.

Bei der Addition haben die GF(2^k) Körper ebenfalls einen Vorteil, da die Addition komponentenweise parallel ausgeführt wird, d. h. insbesondere ohne Überträge auskommt.

5.2 Projektive versus affine Koordinaten

Es hat sich herausgestellt, dass die Ermittlung des multiplikativen Inversen den zeitintensivsten Engpass verursacht. Dies ist ein wesentliches Resultat aus der Diplomarbeit von Bohnsack [BOH97]. Gezeigt werden kann dies anhand von Aufwandsabschätzungen für die Berechnung des Inversen.
Wie bereits oben (vgl. Kap. 2.1) beschrieben, gibt es im Wesentlichen zwei Algorithmen hierfür, wobei der erweiterte euklidische Algorithmus der effizientere ist.

Der erweiterte euklidische Algorithmus wird maximal mit der Gradzahl des Erweiterungskörpers aufgerufen und bei jedem Aufruf wird eine Division mit Rest (nicht zu verwechseln mit dem multiplikativen Inversen, bei dem es keinen Rest gibt), eine Multiplikation und eine Addition benötigt. Diese Operationen benötigen wieder die gleiche Zahl an Bearbeitungsschritten, ihr Aufwand entspricht also der logarithmischen Zahlengröße. Eine ausführliche Analyse des euklidischen Algorithmus findet man beispielsweise in [KNU98], Kap. 4.5.3, eine algorithmische Fassung befindet sich im Anhang dieser Arbeit.

Nimmt man jedoch diese Basisaussage, so bedeutet dies bei einem Körper $GF(2^k)$, dass der erweiterte euklidische Algorithmus k-mal iteriert wird und in jedem Schritt:

- eine Multiplikation,
- eine Division mit Rest (eine ganzzahlige Division),
- sowie eine Addition benötigt werden.

Mit der Festlegung, dass die Multiplikation und Division mit Rest jeweils $O(k)$ Schritte benötigen und die Addition $O(1)$, gelangt man zu folgender Komplexitätsabschätzung für die Division:

$$O(\text{„Division in } GF(2^k)\text{“}) = k\,(2\,O(k) + O(1)) = k\,O(k) \tag{5.1}$$

Bei dem projektiven Modell wurde die Anzahl der Multiplikationen und Additionen lediglich um eine Konstante erhöht. Dies zeigt, dass das projektive Modell solange geeigneter ist, wie die Konstante geringer als der Erweiterungsgrad des Körpers ist.

5.3 Projektive versus rationale Koordinaten

Das rationale Modell vermeidet ähnlich wie das projektive Modell die Ausführung der Division. Der Mehraufwand bei einer Verwendung des rationalen Modells sei hier noch einmal zusammengefasst:

Operation	**Rationales Modell**
1 Addition/ Subtraktion	3 Mult. + 1 Add.
1 Multiplikation	2 Mult.
1 Division	2 Mult.

Tabelle 5.1: Vergleich rationales und projektives Modell

Zusätzlich wird durch die getrennte Darstellung von Zähler und Nenner doppelter Speicherplatz benötigt.

Leider ist dieser einfache Ansatz jedoch weitaus ineffizienter als das projektive Modell, was die folgende Übersicht zeigt:

Operation	**projektiv**	**rational**
Punktverdopplung	10 Mult. + 4 Add.	24 Mult. + 6 Add.
Punktaddition	20 Mult. + 7 Add.	33 Mult. + 9 Add.

Tabelle 5.2: Aufwandsvergleich projektiv vs. rational für elliptische Kurven

Vorteil des projektiven Modells ist die Gesamtoptimierung: Es wird nicht Zahl für Zahl mit getrenntem Zähler und Nenner operiert, sondern die eigentlichen Berechnungsausdrücke werden alle auf **einen** Nenner gebracht. Dieser gemeinsame Nenner wird in der Form der mehrdeutigen projektiven *z*-Koordinate mitgeführt. Dadurch, dass der Zähler für die Operationen Punktverdopplung und Punktaddition nur einmal berechnet wird, entfallen viele Operationen. Zugleich können die Zähler-Operationen einfach zusammengefasst werden.
Dies zeigt, dass die Unterschiede zwischen den beiden Ansätzen in der Art der Verbesserung liegen, also lokal bei dem rationalen Modell und global bei dem projektiven Ansatz.

Entfernt man sich nun von der *einfachen* Anwendung rationaler Zahlen für jede Variable, so können einzelne Ausdrücke effizienter berechnet werden und der Unterschied zwischen dem rationalen und projektiven Modell verringert sich. Dieser Ansatz wird hier mit dem **optimierten rationalen Modell** beschrieben.

Damit der Ansatz des optimierten rationalen Modells anschaulicher wird, ist hier die Berechnung der Addition ungleicher Punkte angegeben. Zugleich dient diese Betrachtung der Nachvollziehbarkeit der Effizienz dieses Ansatzes.

Ausgehend von einer elliptischen Kurve E in Normalform für Körper der Charakteristik zwei:

$$E = \{(x,y) : y^2 + xy = x^3 + ax^2 + b\} \cup \{O\} \quad x,y,a,b \in K\text{=GF}(2^k) \text{ mit b} \neq 0 \qquad (2.39)$$

und zwei Punkten (P_1, P_2) dieser Kurve ergeben sich die Koordinaten des dritten Punktes (P_3) zu (vgl. Kap. 2.3):

$P_3 = P_1 + P_2$ mit den Koordinaten $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ und $P_3 = (x_3, y_3)$

$$x_3 = \lambda^2 + \lambda + b_2 + x_1 + x_2$$

$$y_3 = (x_3 + x_1)\lambda + x_3 + y_1$$

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1} \qquad \text{falls } x_1 \neq x_2$$

$$\lambda = x_1 + \frac{y_1}{x_1} \qquad \text{falls } x_1 = x_2$$

Vorlage sei hier das auch im Akzelerator realisierte **projektive Modell**, welches zuerst betrachtet wird. Grundlage sind die homogenen Koordinaten der projektiven Ebene (vgl. Kap. 2.6):

$$(X, Y, Z) \vdash \left(\frac{X}{Z^2}, \frac{X}{Z^3}\right) \Leftrightarrow (x, y) \quad \text{für Z} \neq 0 \qquad (2.49)$$

Und umgekehrt sind die projektiven Koordinaten eines affinen Punktes:

$$(x,y) \vdash (x,y,1) \quad \text{bzw.} \quad X \leftarrow x,\ Y \leftarrow y,\ Z \leftarrow 1 \tag{2.50}$$

Zu Beginn der Umsetzung von affinen auf projektive Koordinaten (2.50) wird die z-Koordinate gleich 1 gesetzt.

Nun wird zunächst schrittweise die x-Koordinate des dritten Punktes berechnet, sie wird anschließend für die y-Koordinate mitverwendet. Die Umsetzung erfolgt durch die Transformation in das projektive Modell mit den obigen Formeln.
Im Folgenden wird lediglich der Fall ungleicher Punkte P_1 und P_2 betrachtet. Im Fall der Punktverdopplung müssen lediglich die diejenigen Formeln, welche λ enthalten, geändert werden:

$$\frac{x_3}{z_3^2} = \lambda^2 + \lambda + a + \frac{x_1}{z_1^2} + \frac{x_2}{z_2^2}$$

$$\lambda = \frac{\frac{y_2}{z_2^3} + \frac{y_1}{z_1^3}}{\frac{x_2}{z_2^2} + \frac{x_1}{z_1^2}} = \frac{y_2 z_1^3 + y_1 z_2^3}{z_2^3 z_1^3} \cdot \frac{z_1^2 z_2^2}{x_2 z_1^2 + x_1 z_2^2} = \frac{y_2 z_1^3 + y_1 z_2^3}{z_2 z_1 (x_2 z_1^2 + x_1 z_2^2)}$$

damit die Ausdrücke nicht zu groß werden, wird die erste Formel schrittweise berechnet, sowie einzelne Terme zusammengefasst:

Mit $\gamma = y_2 z_1^3 + y_1 z_2^3,\ \beta_1 = z_1 z_2,\ \beta_2 = x_2 z_1^2 + x_1 z_2^2,\ \beta = \beta_1 \beta_2$ wird

$$\lambda = \frac{\gamma}{\beta_1 \beta_2} \qquad \text{und} \qquad \lambda^2 = \frac{\gamma^2}{\beta_1^2 \beta_2^2}$$

$$\frac{x_3}{z_3^2} = \lambda^2 + \lambda + a + \frac{x_1}{z_1^2} + \frac{x_2}{z_2^2} \tag{5.2}$$

Fasst man nun die ersten beiden und die letzten drei Terme zusammen, so ergibt sich:

$$\frac{x_3}{z_3^2} = \frac{\gamma^2 + \gamma\, \beta_1 \beta_2}{\beta_1^2 \beta_2^2} + \frac{a\beta_1^2 + \beta_2}{\beta_1^2} \tag{5.3}$$

Dies lässt sich einfach zusammenfassen:

$$\frac{x_3}{z_3^2} = \frac{\gamma^2 + \gamma\, \beta_1 \beta_2 + a\beta_1^2 \beta_2^2 + \beta_2^3}{\beta^2} = \frac{\gamma(\gamma + \beta) + a\beta^2 + \beta_2^3}{\beta^2} \tag{5.4}$$

Die *y*-Koordinate wird wie folgt berechnet:

$$\frac{y_3}{z_3^3} = \left(\frac{x_3}{z_3^2} + \frac{x_1}{z_1^2} \right) \frac{\gamma}{\beta} + \frac{x_3}{z_3^2} + \frac{y_1}{z_2^3} \tag{5.5}$$

Mit der Zusammenfassung der ersten und letzten Terme:

$$\frac{y_3}{z_3^3} = \frac{(x_3 z_1^2 + x_1 \beta^2)\gamma}{z_1^2 \beta^3} + \frac{x_3 z_2^3 + y_1 \beta^2}{z_2^3 \beta^2}$$

$$\Leftrightarrow \frac{y_3}{z_3^3} = \frac{(x_3 + x_1 z_2^2 \beta_2^2)\gamma}{\beta^3} + \frac{z_2^3 (x_3 \beta + y_1 z_1^3 \beta_2^3)}{z_2^3 \beta^3}$$

$$\Leftrightarrow \frac{y_3}{z_3^3} = \frac{(x_3 + x_1 z_2^2 \beta_2^2)\gamma + x_3 \beta + y_1 z_1^3 \beta_2^3}{\beta^3} \tag{5.6}$$

Wie man bei den beiden obigen Koordinaten sieht, wurde die *z*-Koordinate bereits am Anfang bei der Berechnung der *x*-Koordinate ausgerechnet und braucht daher nicht gesondert ermittelt zu werden.
Es ist möglich, diese Operation (Punktaddition ungleicher Punkte) mit **20** Multiplikationen und **7** Additionen zu bewältigen, dies entspricht dem Algorithmus der projektiven Punktaddition im Kapitel 4.2 (vgl. S. 50 ff.).

Im **rationalen Modell** haben die Zahlen jeweils einen Zähler und Nenner:

$$x_1 = \frac{XZ_1}{XN_1}, x_2 = \frac{XZ_2}{XN_2}, x_3 = \frac{XZ_3}{XN_3}, y_1 = \frac{YZ_1}{YN_1}, y_2 = \frac{YZ_2}{YN_2}, y_3 = \frac{YZ_3}{YN_3} \tag{5.7}$$

Die Berechnung der gleichen Koordinaten für P_3, mit

$P_3 = P_1 + P_2$, $P_1 \neq P_2$ und den Koordinaten $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ und $P_3 = (x_3, y_3)$

sieht wie folgt aus:

$$\frac{XZ_3}{XN_3} = \lambda^2 + \lambda + a + \frac{XZ_1}{XN_1} + \frac{XZ_2}{XN_2}$$

$$\lambda = \frac{(YZ_2 \cdot YN_1 + YZ_1 \cdot YN_2)(XN_1 \cdot XN_2)}{(XZ_2 \cdot XN_1 + XZ_1 \cdot XN_2)(YN_1 \cdot YN_2)} \tag{5.8}$$

Mit der Zusammenfassung von Teilen:

$$\gamma_1 = (YZ_2 \cdot YN_1 + YZ_1 \cdot YN_2),\ \gamma_2 = (XN_1 \cdot XN_2),\ \gamma = \gamma_1 \gamma_2$$

$$\delta_1 = (XZ_2 \cdot XN_1 + XZ_1 \cdot XN_2),\ \delta_2 = (YN_1 \cdot YN_2),\ \delta = \delta_1 \delta_2$$

$$\frac{XZ_3}{XN_3} = \frac{\gamma^2 + \delta\gamma}{\delta^2} + \frac{a(XN_1 \cdot XN_2) + XZ_1 \cdot XN_2 + XZ_2 \cdot XN_1}{XN_1 \cdot XN_2}$$

$$\frac{XZ_3}{XN_3} = \frac{\gamma^2 + \delta\gamma}{\delta^2} + \frac{a\delta_2 + \delta_1}{\gamma_2} = \frac{\gamma_2(\gamma^2 + \delta\gamma) + \delta(\delta_1 + a\delta_2)}{\delta^2 \gamma_2} \tag{5.9}$$

Für die *y*-Koordinate ergibt sich folgender Ausdruck:

$$y_3 = (x_3 + x_1)\lambda + x_3 + y_1 \tag{2.39a}$$

$$\frac{YZ_3}{YN_3} = \left(\frac{XZ_3}{XN_3} + \frac{XZ_1}{XN_1}\right)\frac{\gamma}{\delta} + \frac{XZ_3}{XN_3} + \frac{YZ_1}{YN_1}$$

$$\frac{YZ_3}{YN_3} = \left(\frac{XZ_3 \cdot XN_1 + XZ_1 \cdot XN_3}{XN_1 \cdot XN_3}\right)\frac{\gamma}{\delta} + \frac{XZ_3 \cdot YN_1 + YZ_1 \cdot XN_3}{XN_3 \cdot YN_1} \tag{5.10}$$

Mit der Zusammenfassung von Termen:

$$\Delta_1 = (XZ_3 \cdot XN_1 + XZ_1 \cdot XN_3),\ \Delta_2 = (XN_1 \cdot XN_3)$$

$$\Delta_3 = (XZ_3 \cdot YN_1 + YZ_1 \cdot XN_3),\ \Delta_4 = (XN_3 \cdot YN_1)$$

$$\frac{YZ_3}{YN_3} = \frac{\gamma\Delta_1\Delta_4 + \delta\Delta_2\Delta_3}{\delta\Delta_2\Delta_4} \tag{5.11}$$

Zusammenfassung der Unterschiede:

Die Aufwandsbetrachtung für die Punktberechnung (Addition ungleicher Punkte) bei dem *optimierten rationalen Modell* benötigt **26** Multiplikationen und **8** Additionen, wobei hier keine speziellen Optimierungsuntersuchungen vorgenommen wurden. Die Aufwandsabschätzung entspricht den obigen Formeln unter Hilfenahme von mehrfach berechneten Ausdrücken.

Der Aufwand bei dem optimierten rationalen Modell ist also leicht höher und es wird ein Speicherplatz mehr benötigt. Nach dieser Aufschlüsselung der Berechnung ist deutlich geworden, worin die Unterschiede zwischen den beiden Modellen liegen.
Gezeigt wurden zwei Stufen des rationalen Modells und das projektive Modell:

- Die erste Stufe des rationalen Modells bestand darin, dass für jede Variable Zähler und Nenner getrennt gehalten wurden. Die Divisionsproblematik wurde hierdurch entschärft.

- Die zweite Stufe des rationalen Modells bestand in einer Verbesserung für die gesamte Punktaddition. Die Anzahl der Multiplikationen und Additionen konnte reduziert werden.

Der Unterschied zum projektiven Modell nimmt mit dem Verbesserungsgrad bzw. der Stufe des rationalen Modells ab, Stufe zwei hat nur noch geringe Unterschiede zum projektiven Modell. Das projektive Modell kann als eine verbesserte Fassung des rationalen Modells verstanden werden, bei der der Nenner über **beide Koordinaten** optimiert wurde. Er kann somit als eine Weiterentwicklung des oben vorgestellten optimierten rationalen Modells betrachtet werden.

5.4 Projektive Koordinaten

Eine weitere globale Verbesserung besteht darin, auf die Rücktransformation zu verzichten und das Ergebnis in projektiven Koordinaten (x,y,z) auszugeben. Auf den ersten Blick scheint dies nicht zu den globalen Verbesserungen zu gehören, jedoch führt auch dieser Verzicht zu einer Reduzierung der Additionsaufrufe:

Und zwar wird für die Rücktransformation die Division benötigt und diese wiederum ruft mehrfach die Additionsroutinen auf.
Es sei an dieser Stelle noch einmal daraufhingewiesen, dass aus diesem Grund auf die Division bei der Gruppenoperation verzichtet und die Berechnung über projektive Koordinaten eingeführt wurde.

5.5 Skalarmultiplikation

Berechnet werden soll die Skalarmultiplikation:

$$k \cdot B \text{ mit } k \in \mathbb{N},\ B \in E \text{ (Punkt auf der Kurve)}$$

Nach dem oben angegebenen Algorithmus (vgl. Binäre Methode, Kapitel 4.4, S. 60) für die Skalarmultiplikation und einer angegebenen Bitbreite von

$$n = \lceil \log_2 k \rceil \tag{5.12}$$

n Bits und einer angenommen Gleichverteilung des Skalars, sind im Mittel gleich viele Einsen und Nullen in der binären Repräsentation vorhanden.
Pro verwendetem Bit wird eine Verdopplung benötigt, die Addition wird, zumindest bei der binären Methode (s. S. 60), nur bei gesetzten Bits gebraucht.

Bei beispielsweise 5 Bits sind dies im Mittel 5 Verdopplungen und 2,5 Additionen. Die erste Verdopplung wird meist übersprungen da hierbei die Null verdoppelt würde. Genau genommen sind dann nur *n*-1 bzw. 4 Verdopplungen erforderlich.

Aus diesen Angaben kann nun analytisch der Aufwand für den Algorithmus berechnet werden. Es ergibt sich folgender durchschnittlicher Aufwand:

$$n \times \textit{Punktverdopplung} + n/2 \times \textit{Punktaddition}$$

oder

$$n \times (20\ \textit{Multiplikationen} + 7{,}5\ \textit{Additionen}) \tag{5.13}$$

Die Zahlen für die Multiplikationen und Additionen ergeben sich durch Einsetzen von Tabelle 4.1 (s. S. 49). Die analytischen Aufwandsverhältnisse zwischen Multiplikation und Addition stehen bei ca.

$$\tfrac{20}{7{,}5} = 2 + \tfrac{2}{3} \approx 2,67 \tag{5.14}$$

d.h. auf 2,67 Multiplikationen kommt eine Addition.

Dies ist ein erster Hinweis, dass Verbesserungen der Multiplikation größere Auswirkungen haben werden, als dies bei der Addition zu erwarten ist.

6 Implementierung

Nach dem die Berechnungen und Alternativen diskutiert wurden, erfolgt nun eine Beschreibung der Realisierung in Hardware. Zunächst muss hierfür festgelegt werden, welche Teile der Akzelerator realisiert und welche Aufgaben von der Umgebung des Akzelerators übernommen werden.

Der Akzelerator soll die Skalarmultiplikation durchführen bzw. beschleunigen, diese wiederum benötigt die Punktaddition und Verdopplung und diese Operationen greifen auf Multiplikationen und Additionen für Elemente des GF(2^k) zurück. Es entsteht eine dreistufige Berechnungshierarchie, der Sachverhalt ist in folgender Abbildung veranschaulicht:

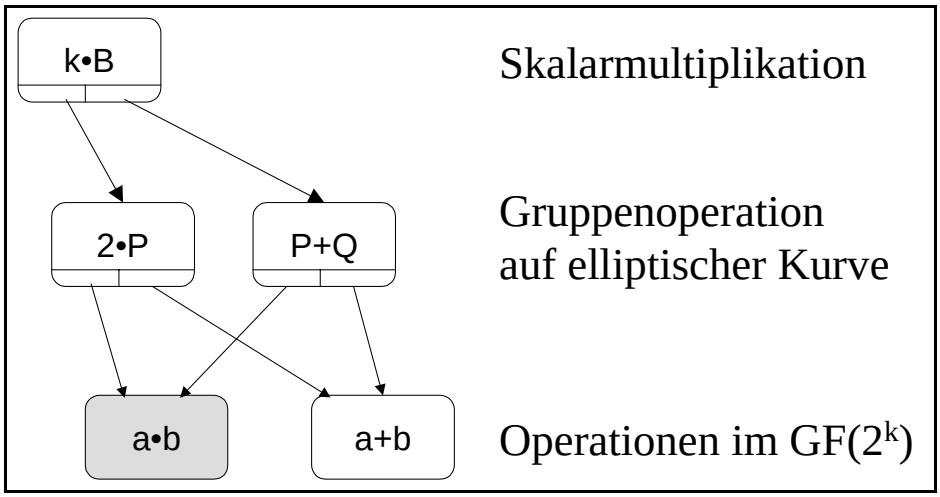

Abbildung 6.1: Berechnungshierarchie der Skalarmultiplikation innerhalb des Akzelerators

Die Funktionalität des Akzelerators ist in obiger Abbildung hervorgehoben. Der Akzelerator dient primär der Multiplikation im GF(2^k). Eingebettet berechnet der Akzelerator die **komplette Skalarmultiplikation**.

An dieser Stelle soll noch einmal der Ansatz eines Akzelerators aufgezeigt werden. Der Akzelerator soll spezielle Funktionen durch dedizierte Hardware unterstützen.

Eine Einbindung der oberen Ebenen aus Abb. 6.1 wäre nur für komplette Systeme vertretbar, in dem Ausgangsfall wird danach nur die unterste Ebene, für Operationen im GF(2^k), unterstützt. Und in dieser Ebene ebenfalls nur die Multiplikation, da die Operation für die Addition in Form einer komponentenweisen XOR-Verknüpfung ebenfalls von Universalrechnern effizient ausgeführt werden kann.
Der Akzelerator dient also primär der Multiplikation. Die GF(2^k)-Addition benötigt keine gesonderte Betrachtung, da sie durch einzelne XOR-Gatter realisiert wird. Diese nehmen keinen nennenswerten Flächen auf dem Chip-Layout ein.

6.1 Besonderheiten der Implementierungspartitionierung

Eine Teilaufgabe dieser Arbeit bestand in der Analyse der Berechnungsaufgabe, deren mathematische Grundlage im Kapitel zwei behandelt wurde. Die schrittweise Verfeinerung der Hauptberechnung (nämlich der Skalarmultiplikation) in kleinere Berechnungseinheiten lieferte den dreistufigen Berechnungsaufbau.
Nach dieser Partitionierung der Berechnung konnte geprüft werden, wie die einzelnen Berechnungsstufen: Skalarmultiplikation, Punktaddition und -multiplikation sowie Addition und Multiplikation im $GF(2^k)$ miteinander verbunden werden können.
Hierzu wurde das gesamte Berechnungsmodell zunächst in Software umgesetzt und Berechnungsengpässe ausgewertet. Die Grundlage hierfür bildeten die Berechnungsalternativen des dritten und vierten Kapitels dieser Arbeit.
Resultat dieser Analysen war, dass das projektive Modell die beste Realisierungsform bezüglich der gestellten Anforderungen darstellt.

Nach dieser Analyse konnte der Weg zu einem hardwarenahen Modell in VHDL (**V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage) beschritten werden. VHDL ist eine formale Hardwarebeschreibungssprache. Ausgang hierfür war eine Beschreibung der Berechnungen auf der Register-Transfer-Ebene (RT-Ebene). Ziel dieses Schrittes ist, eine Hardwarebeschreibung des Akzelerators zu erhalten, die dann als Grundlage für den Standardzellenentwurf eines ASICs (Application Specific Integrated Circuit) dient.
Der Schritt von dem Software-Modell zum synthetisierbaren VHDL-Modell beinhaltet viel Arbeit, unter anderem, weil sich die Entwurfsmethoden für sequentielle Software stark von denen für parallele Hardware unterscheiden.

Die Ergebnisse des ASICs sind in Form von Kennzahlen im Kapitel 7 beschrieben. In diesem Kapitel (6) wird nicht auf alle Details des Entwurfsvorganges sowie seiner Evaluation eingegangen, vielmehr sollen in diesem Kapitel wichtige Elemente und aufgetretene Probleme dieses Aufgabengebietes betrachtet werden.

6.2 Design Entwicklung

In diesem Abschnitt soll ein exemplarischer Einblick in die verschiedenen Stationen des Design-Entwurfs eines ASICs gegeben werden. Anschließend kann dann auf ausgewählte Phasen im Detail eingegangen und auf den Akzelerator angewandt werden.

Die Entwicklung beginnt auf der Systemebene und geht mit zunehmender Spezifizierung der Details von Ebene zu Ebene über die algorithmische Ebene, die Register-Transfer-Ebene (RT-Ebene), zur Logik-Ebene bis zur Schaltungsebene hinunter:

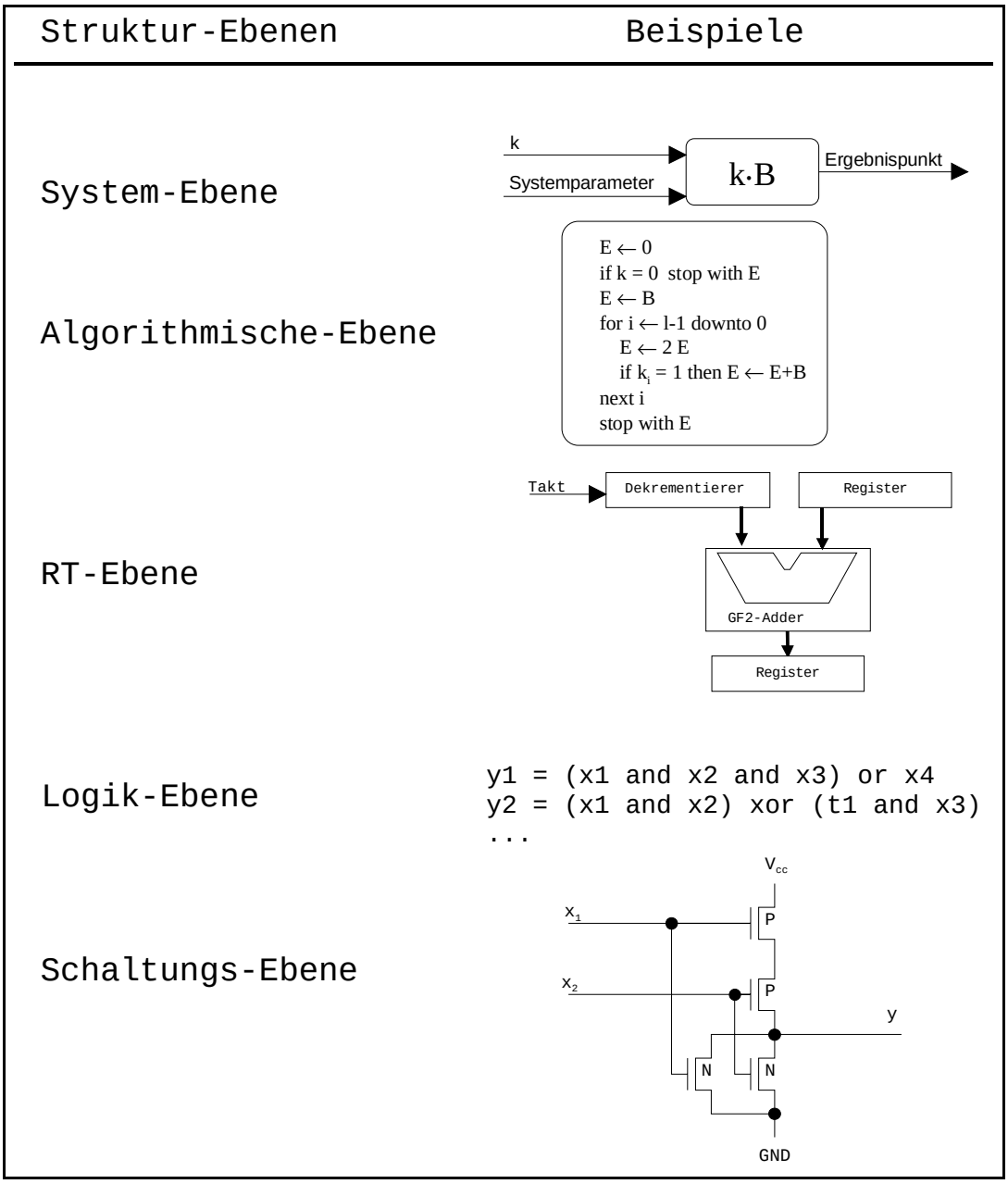

Abbildung 6.2: Strukturübersicht eines Entwurfs

Die Aufgabenstellung wird meist auf der **System-Ebene** formuliert, hieraus wird anschließend eine **algorithmische Spezifikation** gebildet. Die Auswahl an Algorithmus-Alternativen ist Gegenstand der Kapitel drei und vier dieser Arbeit.

Ausgehend von einer algorithmischen Fassung wird das **Register-Transfer-Modell** dieses Algorithmus gebildet. Hierbei werden weitere Detaillierungen über die Struktur getroffen. Festlegungen dieser Ebene betreffen den Datenfluss und die Art- und Weise, wie Funktionen umgesetzt werden.
Für arithmetische Einheiten bedeutet dies festzulegen, wie groß die Wortbreiten der einzelnen arithmetischen Einheiten sind: Ob ein Addierer beispielsweise mit 16 Bit oder zwei mal 8 Bit oder durch einen 8 Bit Addierer mit sequentieller Behandlung der Teilergebnisse eingesetzt wird.

Die **Logikebene** sowie die darunter liegende **Schaltungsebene** wird in der Regel nicht mehr vom Entwerfer bearbeitet. Sie wird durch Tools automatisiert von der höher liegenden RT-Ebene modelliert.
Die Standardzellenbibliothek des Herstellers beinhalten mehrere komplexe Zellen, die die Funktionalität der Logikebene und zum Teil auch direkt Abbildungen auf die RT-Ebene ermöglichen.
Dies bedeutet, dass alle Modellierungen unterhalb der RT-Ebene nur noch im Hinblick auf die verwendeten Standardzellen des späteren Herstellers zu treffen sind.

Das VHDL-Modell wurde in dieser Arbeit direkt von der RT-Ebene umgesetzt bzw. kodiert.

Das VHDL-Modell bzw. die Quelle wird mit Hilfe von Tools bis zur Schaltungsebene umgesetzt. Im ersten Schritt wird die Syntax mit einem Parser getestet. Der compilierte Code kann mit einem Debugger geprüft werden. Dieser Teil eines Entwurfs ist zentral: Hier können erste Realisierungen simuliert werden.
Ein wichtiges Tool zur Visualisierung der Signalvorgänge ist der integrierte Waveform-Viewer, auf dem eine Auswahl von Signalen auf der Zeitachse aufgetragen angezeigt wird.
Die zur Verfügung stehenden Tools und der Vorgang bis zum Layout sind in folgender Übersicht angegeben:

Bei der Hardware-Modellierung für ASICs stehen folgende Tools zur Verfügung:

VHDL-Compiler
: Verifiziert die Syntax und erzeugt ein Zwischenformat, welches sich für die Simulation in einem Debugger eignet.

VHDL-Debugger/ Simulator
: Hier können Wave-Formen (zeitl. Darstellung von Signalen) und Signale des Modells überprüft werden. Für die Verwendung des Debuggers ist die Einrichtung einer Testumgebung notwendig.

Design-Compiler für Standardzellenentwurf (Synthese-Tool)
: Eingabe ist ebenfalls der VHDL-Quellcode, Ausgabe ist eine Netzliste mit konkreter Abbildung auf Komponenten eines Herstellers

Placement- und Routingtools (Synthese-Tool)
: Diese Tools erzeugen die konkrete Platzierung und Verdrahtung der einzelnen Standardzellen.

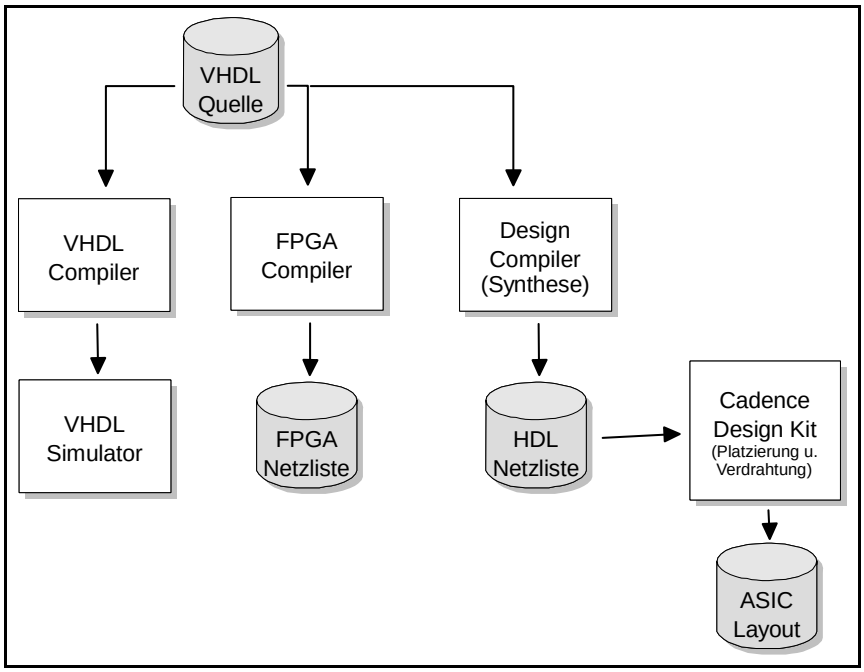

Abbildung 6.3: Tools beim ASIC-Entwurf

Die in Abb. 6.3 gezeigte Übersicht stellt eine starke Vereinfachung des Layout-Entwicklungsvorganges dar, sie zeigt jedoch die wesentlichen Schritte. In ihr fehlt insbesondere die Verifikation des Layouts. Dies geschieht mit Tools, die aus dem geometrischen Layout eine Modell auf Logikebene erstellen und dieses mit dem Logik-Modell des Entwurfs vergleichen. Auch die Verifikation von Timing-Verhalten sowie die Gewährleistung der Testbarkeit auf höherer Ebene (Modellierungsebene) ist in der Abbildung aus Übersichtlichkeitsgründen nicht aufgezeigt.

6.3 Grob-Struktur des ASICs

Beim Entwurf des ASICs wurden generell zwei Varianten verfolgt:

1. Entwurf ohne Speicher, mit Registern in der Größe des verwendeten Körpers. Diese Variante wird im Folgenden mit **DIST** für *distributed* (verteilt) bezeichnet.

2. Entwurf mit Speicher und einer Wortbreite, die kleiner als die Körpergröße ist. Bei diesem Entwurf gruppieren sich alle Einheiten um einen zentralen Datenbus und die ALU (Arithmetic Logic Unit), weshalb diese Variante im Folgenden mit **CENT** für *centralized* (zentriert) bezeichnet wird.

Die DIST-Variante hat Vorteile bei der Platzierung und Verdrahtung und sie erlaubt eine schnelle Berechnung, da die Wortbreite der Körpergröße entspricht wodurch keine größeren Wartezeiten für das Laden eines Registers benötigt wird (vgl. Kap. 3.5). Die Vorteile bei der Platzierung und Verdrahtung werden im Anschluss an die kurze Vorstellung beider Varianten aufgezeigt.

Variante CENT verwendet ein Bus-System, mit dem Vorteil, dass der Bus nur an wenige verarbeitende Einheiten geführt werden muss. Probleme treten bei CENT bei größeren Busbreiten auf. In diesem Fall wird beim Standardzellen-Entwurf die Platzierung und Verdrahtung zunehmend schwerer. Auf die spezifischen Vor- und Nachteile der beiden Varianten wird nach einer schematischen Darstellung und einer kurzen Beschreibung ihrer Elemente zusammenfassend eingegangen.

Zuerst soll die DIST-Variante vorgestellt werden.

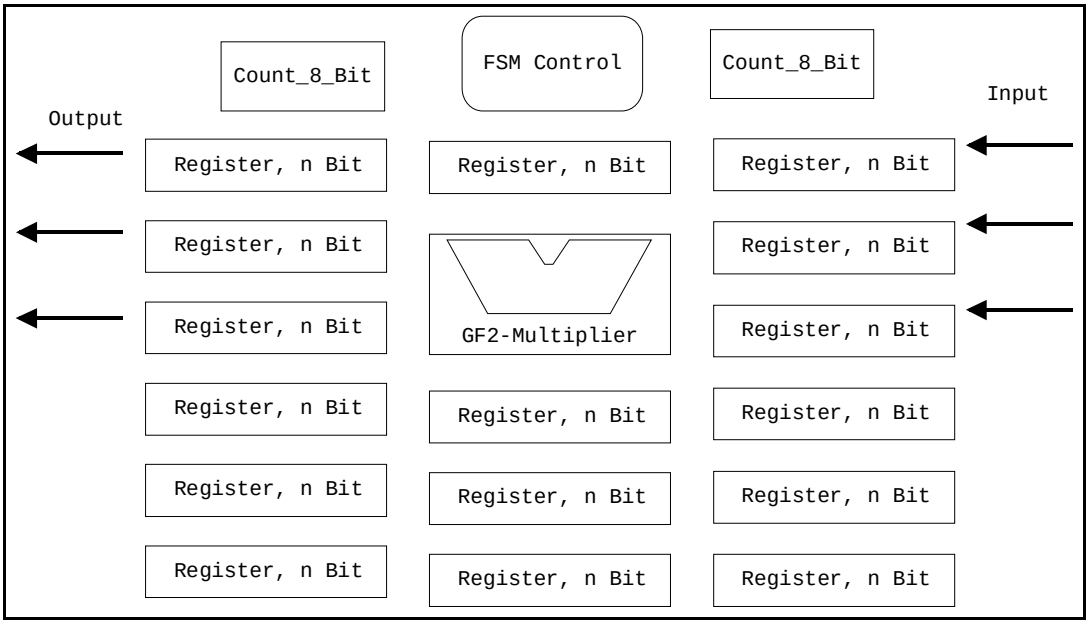

Abbildung 6.4: Strukturübersicht DIST-Variante (ohne Datenpfade)

Die Datenpfade sind aus Gründen der Übersichtlichkeit nicht aufgenommen. Die Addierer (XOR) zwischen einzelnen Registern sind ebenfalls aus Übersichtlichkeitsgründen nicht in Abb. 6.4 aufgenommen worden.
Wesentliches Merkmal der DIST-Variante ist, dass die Register inklusive der ALU sehr gut verteilt werden können. Als Beispiel hierfür ist im folgenden Bild ein Ausschnitt aus der ALU zu sehen:

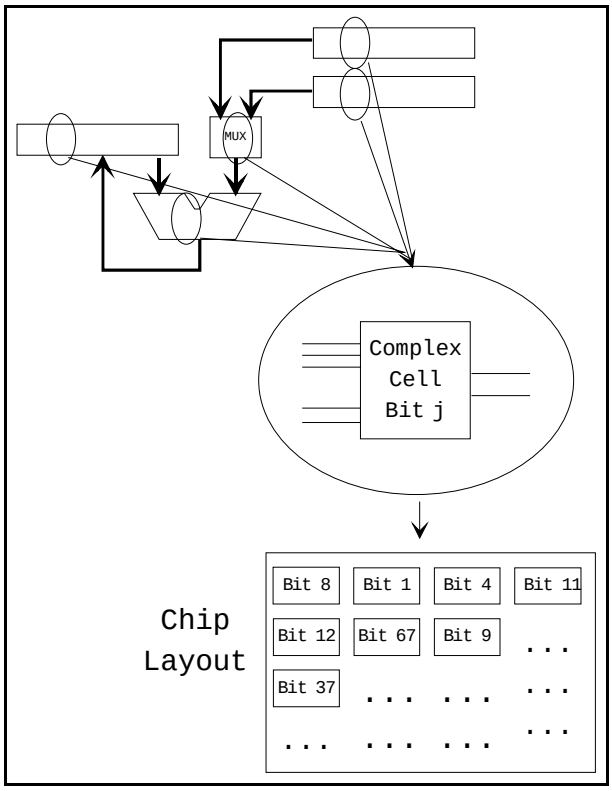

Abbildung 6.5: Umsetzung der verteilten Logik

In obiger Abbildung sieht man wie gut einzelne Teile der Register und ALU miteinander verschmolzen werden können und kleine reguläre Einheiten entstehen. Diese kleinen regulären Einheiten können sehr einfach über die ganze Layoutfläche verteilt werden, weshalb dieser Entwurfsstil auch *distributed logic design style* genannt wird.

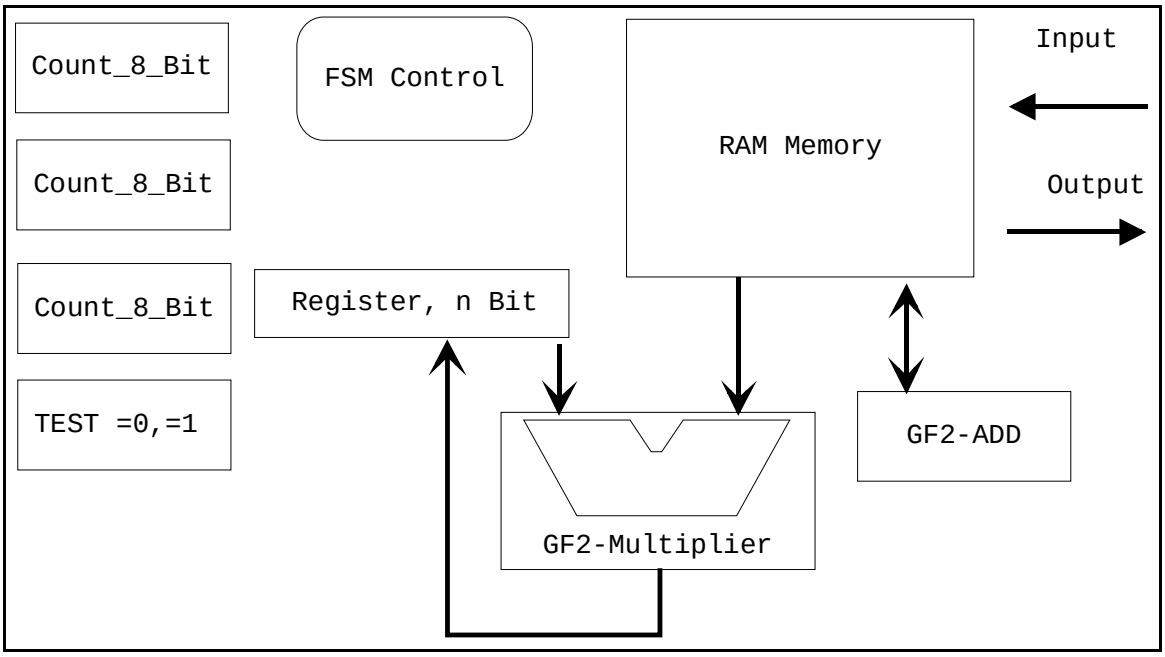

Abbildung 6.6: Strukturübersicht CENT-Variante

Die CENT-Variante wird durch das RAM dominiert.

GF2-Multiplizierer

Bei beiden Varianten wird die GF2-Multiplikation verwendet, ihre Strukturübersicht ist in der folgenden Abbildung zu sehen.

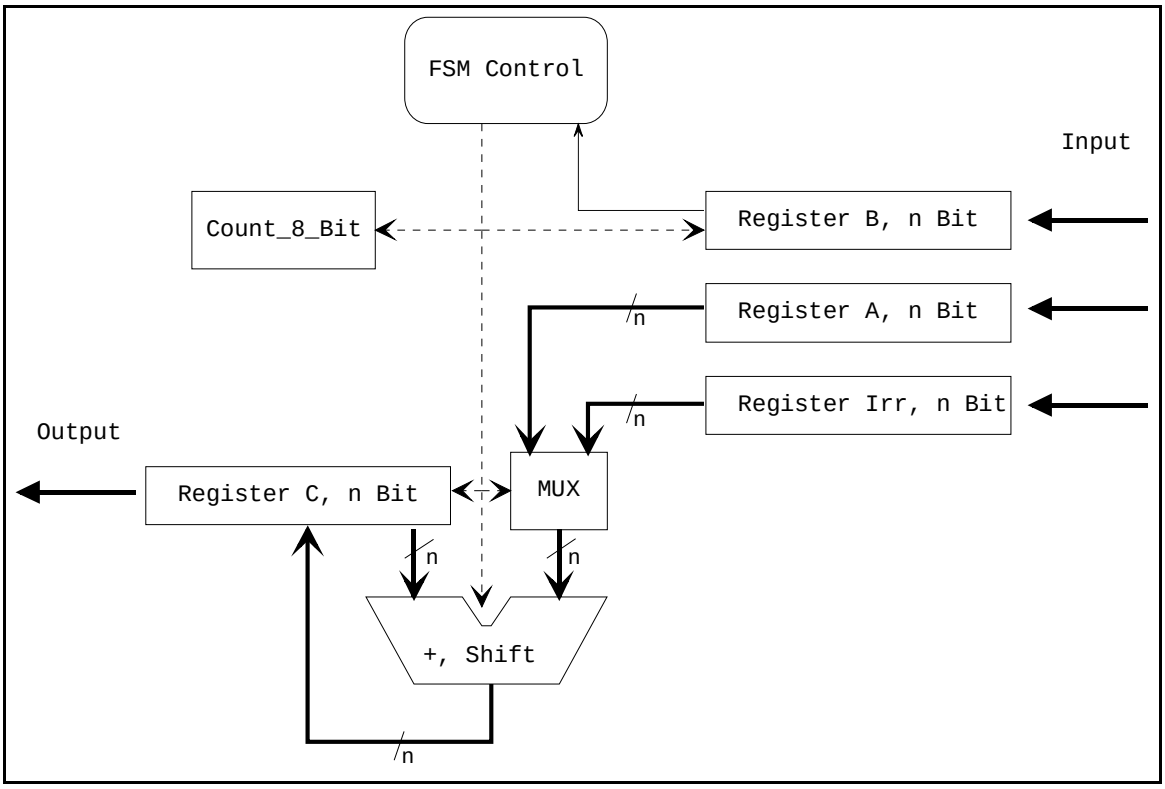

Abbildung 6.7: Strukturübersicht des GF(2^k)-Multiplizierers

Vor- und Nachteile der Varianten

DIST:

Vorteile:

- Mehr Freiheitsgrade bei der Platzierung und Verdrahtung, wodurch weniger Layoutfläche beim Standardzellenentwurf ungenutzt bleibt.
- Höhere Geschwindigkeit ist möglich, da viele kleinere Busse statt eines zentralen Busses vorliegen. Somit ist die kritische Leitungslänge (Critical Path), die den maximalen Takt des Systems bestimmt, kürzer als bei der CENT-Variante.
- Keine Zugriffszeiten für ein RAM-Speicher, kein Flaschenhals.

Nachteile:

- Großer Suchraum bei Platzierung und Verdrahtung, bedingt durch mehr Komponenten und die Freiheitsgrade, insbesondere bei einer hohen Bitbreite bzw. Körpergröße.
- Die Fläche für die Register (Flip-Flops) ist größer als für ein RAM mit entsprechender Speicherkapazität.
- Großer Stromverbrauch

CENT:

Vorteile:

- Geringere Fläche (ohne Verdrahtung), da hoch optimierte RAM-Zellen eingebunden werden. Dem stehen Flip-Flops der DIST-Variante gegenüber.

- Geringer Stromverbrauch

Nachteile:

- Platzierungs- und Verdrahtungsengpässe sind vorprogrammiert, sofern die Busbreiten größer werden.
- Speziell hier sind einzelne schlecht ausgenutzte Verdrahtungskanäle zu erwarten.
- Niedrigere maximale Taktrate aufgrund langer Verbindungsleitungen.

Man sieht bei dieser Gegenüberstellung, dass viele Vorteile Nachteile der anderen Variante sind. Dies zeigt, wie gegensätzlich der Realisierungsstil beider Alternativen ist.

6.4 Probleme mit den Tools

Probleme ergaben sich wie vorauszusehen war, bei den Synthese-Tools. Zwei Punkte kristallisierten sich als besonders schwierig heraus:

1. Das Zeitverhalten bei hohen Wortbreiten von ca. 200 Bit.
2. Das Fehlen von vordefinierten Makrozellen für linear rückgekoppelte Schieberegister.

Die Präsenz von größeren Wortbreiten, die wiederum größere Busse nach sich ziehen, hat erheblichen Einfluss auf alle Tools. Insbesondere bei der Synthese war eine Arbeit mit integriertem Steuerungsautomat auf gleicher Ebene nicht möglich. Die Ursache liegt darin, dass die Tools für derartige Entwürfe offensichtlich nicht ausgelegt sind.

Dies liegt an verschiedenen NP-vollständigen Problemen, die diese Tools zu bewältigen haben. Dies bedeutet, dass in der Praxis die Berechnungszeit mindestens exponentiell mit der Problemgröße (Anzahl der zu platzierenden und verdrahtenden Elemente) ansteigt.

An dieser Stelle sollte man den Hintergrund und die Anforderungen an die Tools beim Chipentwurf nicht außer acht lassen, diese sind:

- Garantie einer zulässigen Lösung
- und kurze Berechnungszeit.

Um diesen Anforderungen gerecht werden zu können, werden z.B. die Probleme der Platzierung und Verdrahtung **nacheinander** gelöst. Als zweites Mittel bedienen sich die Tools bestimmter Heuristiken, um ihre Probleme in polynomieller Zeit zu lösen.
Kritisch ist hierbei, dass die Heuristiken nicht allgemein für alle Aufgaben zugeschnitten sind, sondern für so genannte *typische Anwendungen* gute Resultate liefern.

Leider gehören Systeme mit großen Datenbussen nicht zu diesen *typischen Anwendungen*, weshalb die Tools hierbei mitunter extrem unzureichende Lösungen produzieren.
Charakteristisches Merkmal dieser unzureichenden und teilweise nicht zulässigen Lösungen ist die Existenz einer lokal sehr hohen Auslastung bei der Kanalverdrahtung, wobei in den anderen Kanälen und außerhalb dieser lokalen Auslastung deutlich weniger Fläche benötigt wird. Dies hat zur Folge, dass ein wesentlicher Teil der Layoutfläche ungenutzt bleibt.

Ein Folgeproblem dieser lokalen Häufung und der zuvor festgelegten Platzierung tritt auf, wenn die maximale Kanalkapazität überschritten wird:
Tritt dieser Fall ein, bevor alle Zellen verdrahtet wurden, so können Verbindungen nicht mehr durch diesen Kanal geführt werden. Der einzige Ausweg besteht darin, die Verdrahtung außen, über einen anderen Kanal zu führen.
Hierdurch entstehen Leitungslängen, die ein Vielfaches der direkten Weglänge ausmachen. Resultat dieser Bahnverlängerung ist eine Reduzierung der maximalen Taktrate und eventuell eine resultierende erforderliche Erhöhung der Treiberleistung für diese Netze.

Ein weiterer Punkt liegt insbesondere an dem Umfang des Übertragungsprozesses aus dem VHDL-Modell auf die Herstellerbibliotheken.
Optimal in diesem Zusammenhang wäre ein Rückgriff auf Makrozellen für Rückgekoppelte Schieberegister, da in diesem Fall der Hersteller für eine optimale Abbildung auf seine Standardzellen sorgen könnte. Sind derartige Makrozellen nicht vorhanden, so müssen sie erst vom Anwender modelliert werden.

6.5 Einbindung des Akzelerators in komplexe Systeme

Der Akzelerator ist so konzipiert, dass er eine spezielle Aufgabe wahrnimmt, insbesondere ist er aber ohne entsprechenden Support seiner Umgebung nicht in der Lage selbständig zu agieren.
Er ist somit ein Hardware-Modul auf einem Chip, das Szenario ist in Abb. 6.8 skizziert:

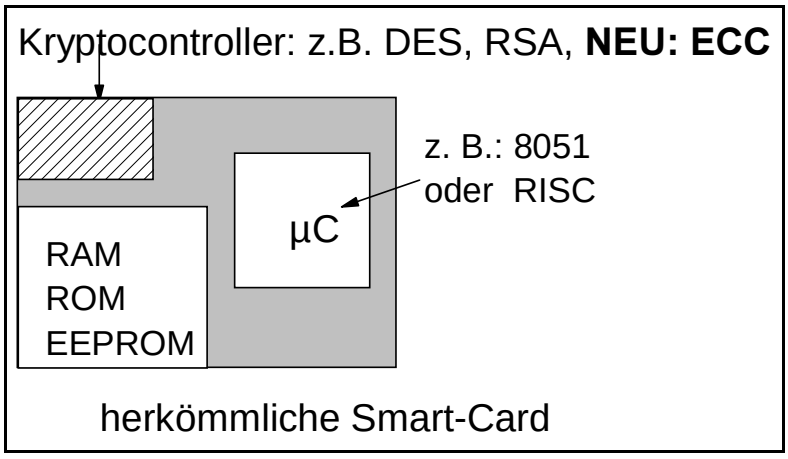

Abbildung 6.8: Beispiel für die Systemeinbindung des Akzelerators

Für derartige Systeme sind neue Entwurfsmethoden und Testverfahren notwendig, da die Gatterzahl im Vergleich zu einzelnen, kleineren Designs, um Größenordnungen höher liegt. Es sind spezielle Entwurfsstrategien für die Testbarkeit der Einzelmodule und des Gesamtsystems notwendig. Eine gute Übersicht über diese Thematik findet man in [REIF98].

Es ist erforderlich, dass die Teststrategie für derartige Entwürfe integriert wird. Generell sind bei Controllern neben den funktionalen Blöcken zusätzliche Einheiten für die Testbarkeit einzubinden.

Bei der Einbindung des Akzelerators in ein SoC (**S**ystem **o**n a **C**hip) sollten folgende Punkte berücksichtigt werden (vgl. [REIF98], S. 26, 63, 67 f.):

- Keine Verwendung gemischter Flanken (entweder steigende oder fallende Flanke) für die Clock .
- Keine internen Clock-Signale erzeugen.
- Verwendung von Signalen anstatt Variablen in der VHDL Quelle.
- Vermeidung von Latches.
- Integration der In-Circuit-Teststrategie in den Entwurf.

Die Verwendung unterschiedlicher Flanken für Speicherelemente in einem Design ist oftmals aufgrund von Bibliotheksbeschränkungen nicht synthetisierbar. Auch die Verwendung von Variablen kann in heterogenen Umgebungen zu Problemen führen. Die Vermeidung von Latches wird auch in anderen Designs vermieden.

Bei dem hier beschriebenen Akzelerator ist die Testbarkeit bereits durch die verwendete Struktur gegeben. Bei einer Skalarmultiplikation mit großem Körpergrad werden nahezu alle Berechnungspfade beschritten. Dies entspricht zugleich dem besten Test, der durch die reale Anwendungsumgebung gegeben ist.
Die Bedeutung des Praxistest zeigt sich auch an einer aktuellen Statistik aus [REIF98], wonach 90 % der ASICs bei der ersten Silikonfertigung funktionieren, jedoch nur 50 % hiervon auch den Praxistest fehlerfrei bestehen. Häufigste Ursache ist eine fehlende System-Level-Simulation. Deren Machbarkeit hängt jedoch von der verwendeten Gatterzahl ab.

6.6 Anforderungen an innovative Tools

Neben dem eigentlichen Ziel, nämlich der Erstellung eines Chip-Layouts, wurden nach der Implementierung die einzelnen Entwurfsschritte als solche ausgewertet. An dieser Stelle sollen die Erfahrungen, die hierbei gemacht wurden, kurz wiedergegeben werden.

Es hat sich gezeigt, dass eine Software-Simulation in einer nicht hardware-nahen Sprache, deutliche Probleme bei der späteren Umsetzung mit sich bringt. Dies trifft auch auf die hardwarenahe Sprache VHDL zu. Diese Tatsache mag zunächst verwundern, erklärt sich jedoch dadurch, dass die Synthese-Tools nur bestimmte Ausdrücke umsetzen können.
Hierbei darf man den Hintergrund von VHDL nicht außer acht lassen, denn VHDL wurde eigens für die Simulation und Dokumentation entworfen.

Ausschlaggebend für eine leicht zu übernehmende Software-Simulation ist ihre Verwandheit mit hardwareähnlichen Mechanismen. Hier sollten Objekte wie Zähler, Vergleicher und Handshake-Signale verwendet werden. Prinzipiell sind diese Aufgaben objektorientierte Sprachen besonders geeignet.
Verwendet man jedoch neben diesen hardwarenahen Objekten andere Elemente der jeweiligen Sprache, so erhöht sich der Aufwand bei einer späteren Umsetzung in Hardware. Kleine

Konstrukte, die bei dem Software-Modell als nebensächlich eingestuft wurden, können zu komplexen Realisierungsproblemen im Hardware-Modell führen.
In dieser Arbeit entstand ein gewisser Modellierungsoverhead durch die Beschreibung des Interfaces für das Ein- und Auslesen der Daten, sowie der Handhabung diverser Zähler und Adresszeiger. Wie beachtlich dieser Overhead ist und welcher Fläche er entspricht, wird im folgenden Kapitel veranschaulicht.

Ein zweiter Problempunkt bestand in der Verifikation der Modelle durch Testvektoren. Da die Anzahl der internen Zustände, bedingt durch die Registergröße und -zahl, sehr hoch ist, müssen entsprechend viele Testvektoren geprüft werden.
Die Verifikation mit Testvektoren beanspruchte einen nicht unerheblichen Zeitaufwand, der zuvor unterschätzt wurde. Hierbei wurde deutlich, wie wichtig es ist, entsprechende Automatisationen (in Form von Scripten) in der Überprüfung zu verwenden. Eine Testumgebung mit automatisierten Elementen bedarf jedoch auch einer gewissen Flexibilität, damit sie auch nach kleineren Änderungen am Modell nicht neu entworfen werden muss.

6.7 Unterstützung von Änderungen der Wortbreite

In dem Hardware-Modell wurden alle Parameter, die die Wortbreite und den maximalen Körpergrad betreffen in einer gesonderten Einheit (*package*) untergebracht. Hierdurch sind Analysen für verschiedene Wortbreiten sehr einfach einstellbar und Ergebnisse können ohne Änderungen am Hardware-Modell durchgeführt werden.
Diese Parametrisierung dient auch der Gewinnung von asymptotischen Aussagen für Körpergrade, die nicht synthetisierbar sind. Abschätzungen befinden sich im folgenden Kapitel, das als Datenblatt des ASICs dient.

7 Kenndaten des ASICs

7.1 Angaben zu den GF(2^k)-Multiplizieren

Der verwendete GF(2^k)-Multiplizierer wird hier in verschiedenen Realisierungsformen betrachtet. Generell wurde entsprechend des Designstils entweder ein Multiplizierer in voller Wortbreite, d.h. Wortbreite gleich Körpergröße, oder eine geringere Wortbreite verwendet. Bei der geringeren Wortbreite muss der Multiplizierer entsprechend öfter aufgerufen werden, bis das Resultat für die volle Körpergröße vorliegt.

Die Klassifizierung erfolgt in folgender Tabelle durch den Typ:
- N = Wortbreite gleich Körpergröße,
- M = Wortbreite wie angegeben und Anzahl der benötigten Worte in der Spalte Wortzahl.

Typ	Körpergröße	Wortbreite	Wortanzahl	Fläche [mm²]
N	192			3,92
N	96			2,05
N	48			1,12
N	32			0,75
M	192	32	6	4,72
M	192	16	12	4,61
M	96	32	3	2,66
M	96	16	6	2,55
M	48	16	3	1,54
M	32	16	2	1,17

Tabelle 7.1: Flächenangaben zu GF(2^k)-Multiplizierern

Die Flächenangaben beziehen sich auf Abschätzungen des Synthese-Tools, d.h. eine reale Platzierung und Verdrahtung wurde noch nicht berücksichtigt.

Der Verlauf dieser Messpunkte kann in folgender Grafik betrachtet werden, wobei zu Berücksichtigen ist, dass für die Wortbreite 32 nur die Messpunkte 96 und 192 vorliegen:

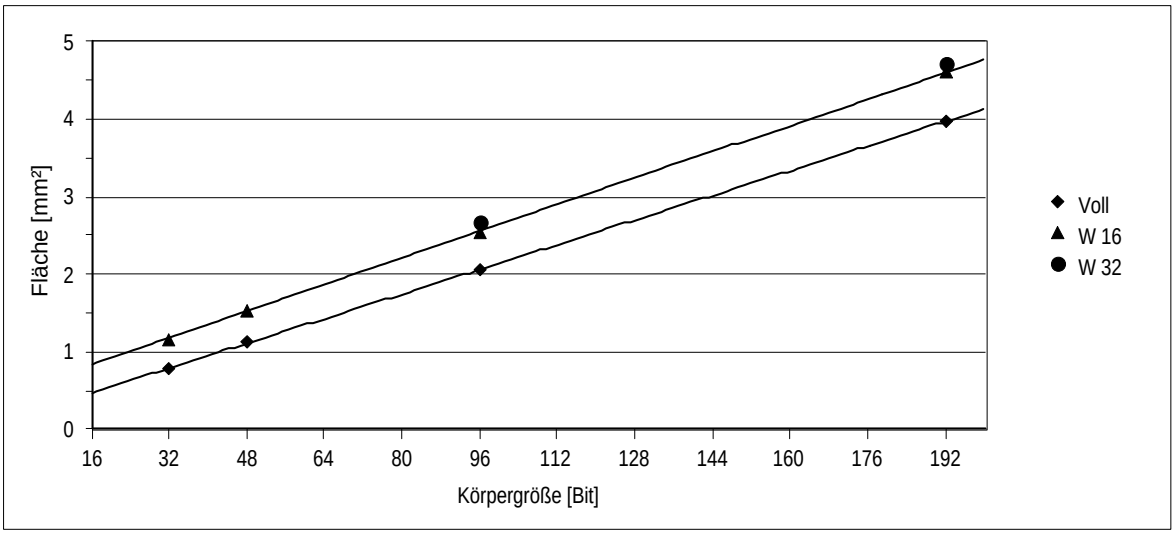

Abbildung 7.1: Flächenbedarf des GF(2^k)-Multiplizierers nach Typ und Wortbreite

Aus obiger Abbildung (7.1) geht deutlich hervor, dass die **volle Wortbreite** stets **besser** abschneidet, als die Aufteilung in Worte kleinerer Größe. In der linearen Regression schneidet die volle Wortbreite auch deutlich besser ab. Dieser Trend kann jedoch aufgrund der Abschätzungen des Synthese-Tools nicht bestätigt werden, da die Unsicherheit für die *untypischen Anwendungen* (vgl. Anm. in Kap. 5.3) mit der Designfläche steigen dürften.
Die zweite Aussage obiger Messung ist, dass die doppelte Wortbreite nur geringfügig mehr Designfläche benötigt. Dies ist besonders positiv, da somit eine doppelt so schnelle Bearbeitung nur geringfügig mehr Fläche benötigt. Dies bedeutet, dass der Overhead an Zählern u. ä. Komponenten von der Fläche her einen wesentlich größeren Anteil, als die eigentlichen Register ausmachen. Dies erklärt auch, warum im Extremfall, also bei voller Wortbreite, weniger Fläche benötigt wird.

Um diese Abschätzungen in einem festen Bezug zur Realität setzen zu können, wurden einzelne Entwürfe nicht nur synthetisiert, sondern auch platziert und verdrahtet. Diese Layout-Angaben bieten im Vergleich zu den Abschätzungen realitätsnahe Ergebnisse.

LAYOUT FÜR GF(2^K)-MULTIPLIZIERER

Prozess:	0,6 µm, 2-Lagen-Verdrahtung
Standardzellen-Bibliothek:	AMS
Multiplizierer-Typ:	**32 Bit**, Typ N (volle Wortbreite)
Kantenlängen:	692 x 725 µm
Fläche:	**~ 0,5 mm²**
Hinweise zur Fläche:	Design ohne PAD-Kranz

Abbildung 7.2: Layoutdaten des GF(2^k)-Multiziplierers

Das Layout hat keine Besonderheiten in seiner Struktur. Die Kanaldichten sind annähernd gleich verteilt und weist keine lokale Konzentrationen auf. Das rückgekoppelte Schieberegister des GF(2^k)-Multiplizierers konnte offensichtlich gut über die Layoutfläche verteilt werden.

7.2 Angaben zum Skalarmultiplizierer

Der Skalarmultiplizierer beinhaltet als Multiplizierer den GF(2^k)-Multiplizierer in voller Wortbreite (Typ N). Die maximale Taktrate und Flächenabschätzungen sind in Tabelle 7.2 angegeben.

Körpergröße *k*	**max. Taktrate [MHz]**	**Fläche [mm²]**	**Fläche pro Bit [mm²]**
5	131,2	2,08	0,42
16	129,7	4,39	0,27
24	130,1	5,98	0,25
32	133,7	7,76	0,24
40	132,3	9,48	0,24
48	130,7	12,48	0,26
56	122,5	14,27	0,25

Tabelle 7.2: Leistungen und Flächen verschiedener Skalarmultiplizierer

Als Vergleichsgröße enthält Spalte 4 die Fläche pro Bit der Körpergröße.

Die Flächenangaben beziehen sich auf Abschätzungen des Synthese-Tools, d.h. eine Platzierung und Verdrahtung wurde noch nicht berücksichtigt. Der Verlauf dieser ermittelten Daten kann in folgender Grafik betrachtet werden:

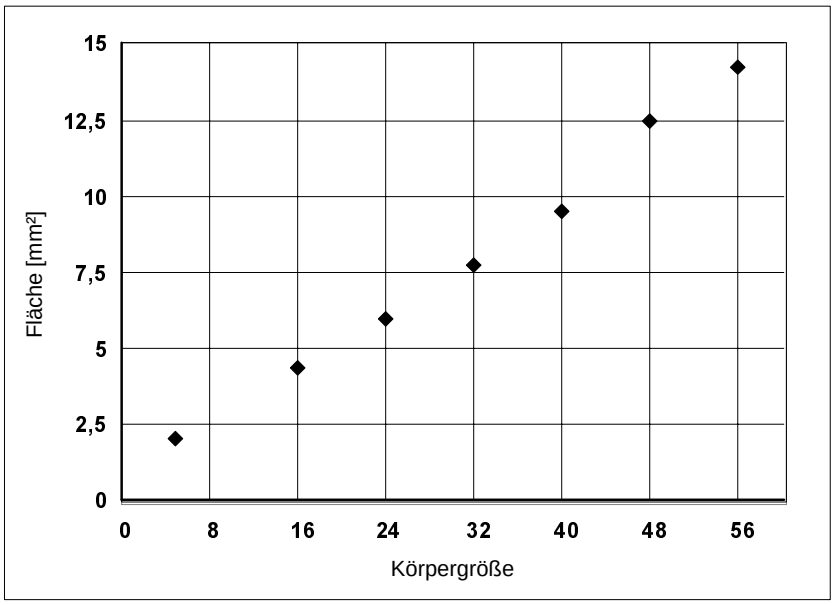

Abbildung 7.3: Fläche des Skalarmultiplizierers in Abhängigkeit von der Körpergröße

Abb. 7.3 lässt die Vermutung zu, dass für größere Werte der Körpergröße, ein linearer Anstieg der Kurve vorliegen wird. Interessant im Vergleich zu den absoluten Flächenangaben in Abb. 7.3 sind die relativen Flächenangaben des Skalarmultiplizierers pro Bit der Körpergröße (Abb. 7.4):

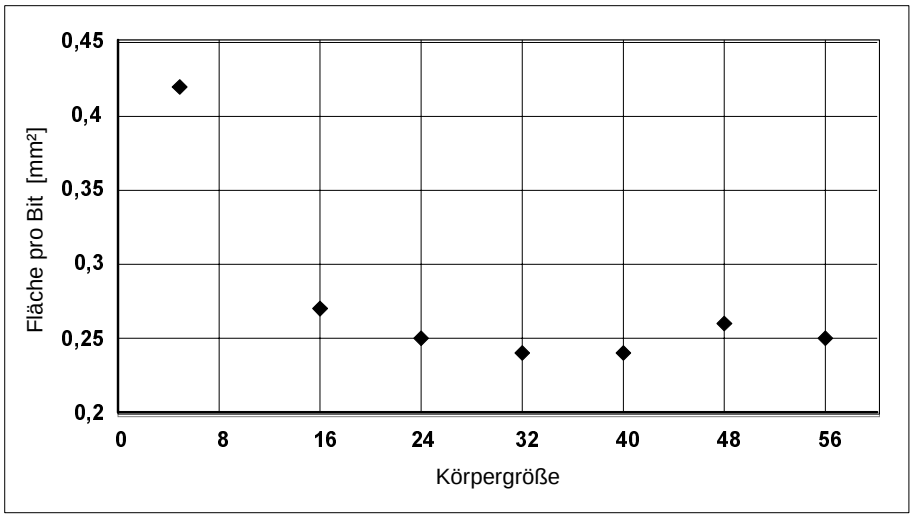

Abbildung 7.4: Fläche des Skalarmultiplizierers pro Bit der Körpergröße

In Abb. 7.4 erkennt man deutlich eine Konvergenznäherung zu einem bestimmten Wert, bei größeren Bitbreiten. Dieser Effekt kann folgendermaßen erklärt werden: Die Fläche für die endliche Kontrolle (State-Machine) ist konstant und die seriellen Ein- und Ausgabeeinheiten steigen nur linear mit der Körpergröße an. Ab einer gewissen Körpergröße treten diese Effekte in den Hintergrund und der relative Flächenanteil pro Bit konvergiert (vermutlich) gegen einen Wert, in der Größenordnung von 0,25 mm².

Eine kompakte Übersicht bietet die Kombination der Abbildungen 7.3 und 7.4:

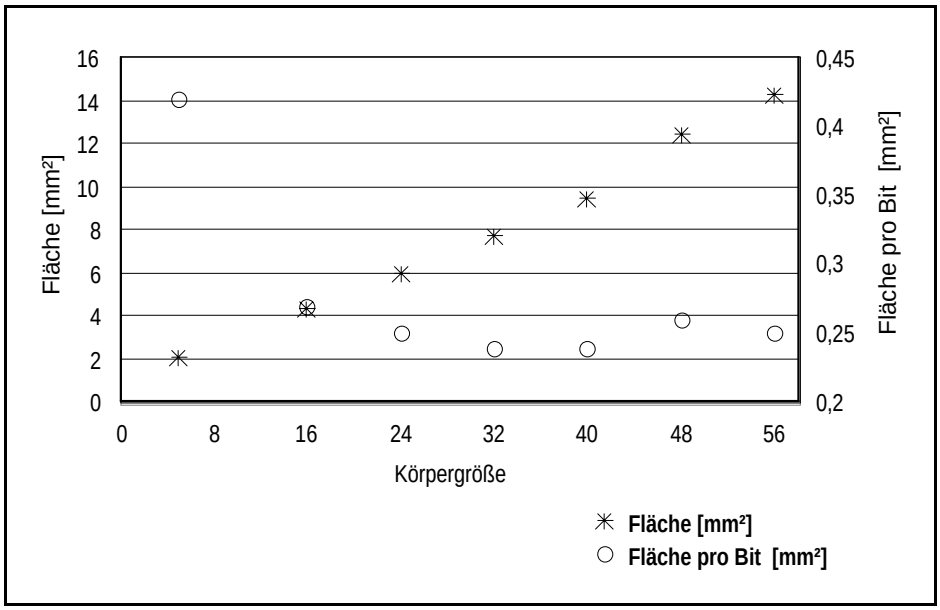

Abbildung 7.5: Fläche und Fläche pro Bit des Skalarmultiplizierers

Die Abweichungen von dem vermuteten Grenzwert bei der Kurve Fläche pro Bit, deutet auf unterschiedliche Layoutoptimierungen des Synthese-Tools hin. Hier sollte noch einmal auf den komplexen Prozess der Platzierung und Verdrahtung hingewiesen werden: Die Synthese-Ergebnisse hängen auch von verschiedenen Heuristiken ab, so dass zwei Synthese-Vorgänge ein- und derselben Beschreibung zu unterschiedlichen Ergebnissen führen kann. Die Heuristiken erklären zudem, warum kleinere Schwankungen durchaus erwartet werden müssen.

Durchsatz und Taktrate

Tabelle 7.2 gibt die maximalen Taktraten an, dort wird ein Bereich zwischen ca. 130 und 133 MHz ausgegeben. Um aus diesen Werten den Durchsatz des Skalarmultiplizierers berechnen zu können, müssen die theoretischen und simulierten Durchlaufzeiten mit der Taktrate aufgeschlüsselt werden.
Der Durchsatz bestimmt sich wie folgt:

$$Durchsatz(n, f_t) = \frac{n \cdot f_t}{Zyklen(n)} \tag{7.1}$$

f_t	ist die Taktrate
$Zyklen(n)$	ist eine Funktion, welche die Zahl von Taktzyklen für die Bitbreite n liefert

Der Durchsatz ergibt sich aus der Zeit, die der Akzelerator für eine Berechnung ($k \cdot B$) benötigt, multipliziert mit der Informationsmenge, die dabei verwendet werden kann in Bits. Die Berechnungszeit ist:

$$Berechnungszeit(n, f_t) = \frac{Zyklen(n)}{f_t} \tag{7.2}$$

Die logarithmische Zahlengröße des Skalars (seine Bitbreite) entspricht der aufnehmbaren Informationsmenge. So gelangt man zur Formel 7.1.

Zum theoretischen Durchsatz gelangt man anhand des VHDL-Modells. Aus diesem Modell müssen die benötigten Zyklen abgeschätzt werden. Als grobe Näherung können folgende Werte dienen:

Funktion	**Zyklen**
GF(2^k)-Multiplikation	$4n+1$
GF(2^k)-Addition	1
Punktaddition (projektiv)	$15\cdot$(GF2-Mul.)$+7\cdot$(GF2-Add.)$+1 = 60n + 23$
Punktverdopplung (projektiv)	$10\cdot$(GF2-Mul.)$+5\cdot$(GF2-Add.)$+1 = 40n + 16$
Restliche Berechnungen	$2n^2 + 18n + 11$

Tabelle 7.3: Zyklenabschätzungen für Bitbreite *n*

Diese Werte gehen in die Abschätzungen für den mittleren Aufwand der Skalarmultiplikation (vgl. Tab. 4.1 und Kap. 5.5) ein. Hierbei wurde die Addition von Punkten mit ½ im Vergleich zur Punktverdopplung gewichtet (s.o. (5.13)):

$$Zyklen(n) = 72n^2 + 45,5n + 11 \tag{7.3}$$

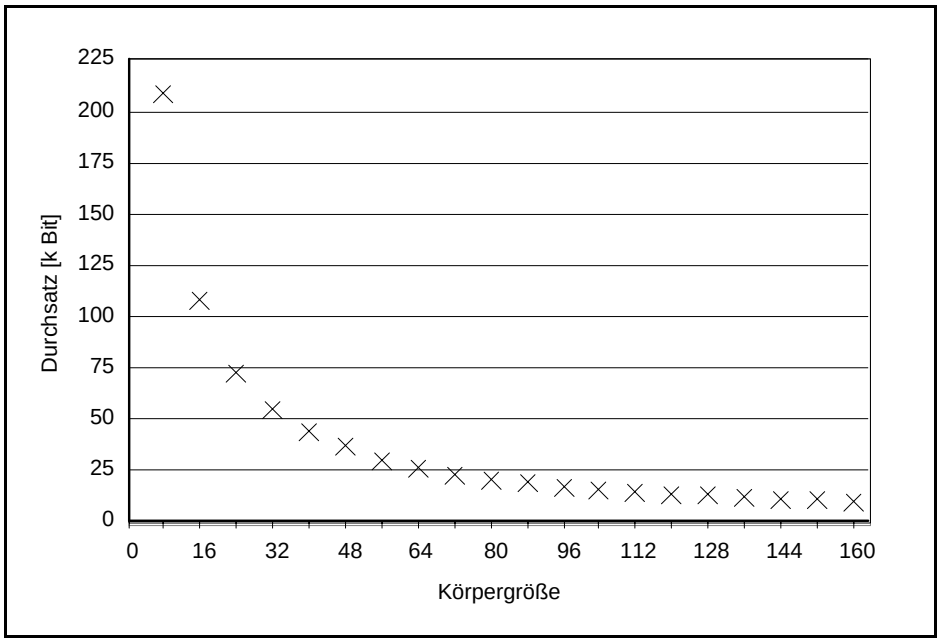

Abbildung 7.6: Theoretischer Durchsatz des Skalarmultiplizierers, Funktion 7.3 mit Frequenzreduzierung von 130 MHz (für 8 Bit) bis 122 MHz (für 160 Bit)

Der theoretische Durchsatz ist für einige Körpergrößen in Abb. 7.6 dargestellt.

LAYOUT FÜR SKALARMULTIPLIZIERER

Prozess:	0,6 µm, 2-Lagen-Verdrahtung
Standardzellen-Bibliothek:	AMS
Multiplizierer-Typ:	**40 Bit**, Typ N (volle Wortbreite)
Kantenlängen:	2350 µm
Fläche:	**~ 5,5 mm²**
Hinweise zur Fläche:	Design ohne PAD-Kranz, **39 Pins**

Abbildung 7.7: Layoutdaten für den Skalarmultiplizierer

Die zentrale Bedeutung des GF(2^k)-Multiplizierers für den Durchsatz des Gesamtsystems wird durch (7.3) deutlich:

$$Zyklen(n) = 72n^2 + 45,5n + 11 \tag{7.3}$$

Eine Halbierung der Arbeitsschritte des GF(2^k)-Multiplizierers, bringt nahezu eine Verdopplung des Durchsatzes (7.4) und (7.5). Bei der Betrachtung wurde folgende Gewichtung der Punktverdopplungen und Punktadditionen für die Ausführung der Skalarmultiplikation verwendet:

$$n \times Punktverdopplung + n/2 \times Punktaddition$$

oder

$$n \times (20\ Multiplikationen + 7{,}5\ Additionen) \tag{5.13}$$

Aus diesen Angaben kann nun analytisch der Aufwand für den Algorithmus berechnet werden. Es ergibt sich folgender durchschnittlicher Aufwand, wenn die GF(2^k)-Multiplikation doppelt so schnell wäre:

$$Zyklen\_GF(\tfrac{1}{2}) = 37n^2 + 45,5n + 11 \tag{7.4}$$

$$\frac{37}{72} \approx 51,4\% \tag{7.5}$$

Wobei (7.5) lediglich den größten Term von (7.4) und (7.3) miteinander vergleicht.

Dies unterstreicht, weshalb eine Verbesserung des Multiplizierers die größten Performanceverbesserungen verspricht.

8 Ausblick und Zusammenfassung

8.1 Zusammenfassung

Das Thema dieser Arbeit lautet: *Konzeption, Evaluierung und Implementation eines Akzelerators für elliptische Kurven.*
Der Fokus auf einen Hardware-Akzelerator mit den Rahmenbedingungen: geringer Speicher, geringe Fläche etc., hat gezeigt, dass viele Ansätze der Literatur hierauf nicht passen und demnach nicht übernommen werden können. Die überwiegende Mehrzahl der Literatur konzentriert sich auf Berechnungsmodelle mit deutlich mehr Speicher, als er bei einem Akzelerator vorhanden ist. Es werden Vergleiche mit Implementierungen auf Intel-Pentium Prozessoren und SPARC-Workstations durchgeführt, hierbei steht ein RAM-Speicher von einigen Megabytes zur Verfügung, während der Akzelerator mit wenigen Kilobits (ca. 2) auskommen muss.
Derartig unterschiedliche Vorgaben bzw. praktische Randbedingungen erfordern entsprechende Differenzierungen bei dem Vergleich einzelner Realisierungsmöglichkeiten.

Die **Realisierungsmöglichkeiten** bei elliptischen Kurven sind sehr groß, in folgender Übersicht (Abb. 8.1) sind die, für diese Arbeit wesentlichen Realisierungsmöglichkeiten, aufgezeigt:

- Körper:
  - GF(p) Problem: Modulo, Vergleich und Invertierung
  - GF(p^m) Problem: wie GF(p), zusätzlich: mehr Speicher
  - GF(2^m) Problem: Invertierung
- Wahl der Basis bei GF(p^m), GF(2^m):
  - polynomiell $B = \{1, x, x^2, x^3, ..., x^m\}$
  - normal $B = \{\alpha, \alpha^2, \alpha^{2^2}, \alpha^{2^4}, ..., \alpha^{2^{m-1}}\} : \alpha \in GF(2^m)$

    einfache Quadrierung, komplizierte Multiplikation
- Koordinatensystem:
  - affine Koordinaten (x, y)
  - projektive Koordinaten (x, y, z) (keine parallelen Geraden)

    Problem: komplizierte Gruppenoperation

Abbildung 8.1: Realisierungsmöglichkeiten elliptischer Kurven mit wesentlichen Merkmalen

Die Realisierungsmöglichkeiten elliptischer Kurven lassen sich in **drei Stufen** einteilen: Körper, Basis bei Erweiterungskörpern und Koordinatensystem.
Anhand dieser Realisierungsmöglichkeiten, wurde nach den Spezifika für einen Hardware-Akzelerator, pro Stufe eine Auswahl getroffen. Abbildung 8.1 zeigt neben den Realisierungsmöglichkeiten die wesentlichen Merkmale eines jeden Kriteriums auf. Die getroffene Auswahl ist in Abbildung 8.2 wiedergegeben:

- Körper:
  - GF(p) Problem: Modulo, Vergleich und Invertierung
  - GF(p^m) Problem: wie GF(p), zusätzlich: mehr Speicher
  - ✓ GF(2^m) Problem: Invertierung
    Vorteile: einfache Addition, schnelle Multiplikation möglich,
    - hardwarefreundlich -
- Wahl der Basis bei GF(p^m), GF(2^m):
  - ✓ polynomiell Vorteil: wenig Speicher
  - normal
- Koordinatensystem:
  - affine Koordinaten (x, y)
  - ✓ projektive Koordinaten (x, y, z)
    Vorteil: Invertierung lässt sich reduzieren

Abbildung 8.2: Realisierungsentscheidungen für den Hardware-Akzelerator

Die Wahl des **Körpers** fiel auf den GF(2^k), da die Probleme der Modulo-Operationen einfach in Hardware realisierbar sind, als bei den anderen Körpern.
Die Wahl der **Basis** fiel auf die polynomielle, da normale Basen einen wesentlich höheren Speicheraufwand benötigen.
Anhand dieser Auswahlstufe werden die Beschränkungen des Akzelerators besonders deutlich: Den der Speicheraufwand für die GF(2^k)-Multiplikation bzw. für die Koeffizienten der Multiplikationsmatrix würde bereits den Speicherrahmen von 2 Kilobits sprengen (vgl. Kap. 2.1 auf S. 17 ff).
Die Wahl des **Koordinatensystems** fiel auf homogene Koordinaten in der projektiven Ebene. Bei diesen Koordinaten ist es möglich, die sehr aufwendige Division weitestgehend zu entschärfen und auf eine Division in GF(2^k) pro Skalarmultiplikation zu reduzieren.
Ein **Zwischenschritt** hierbei ist das **rationale Modell**, welches eine getrennte Zähler-Nenner-Darstellung verwendet um so die Division zu vermeiden. Der rationale Ansatz hat jedoch neben seinen positiven auch negative Effekte: Hierzu zählen der erhöhte Speicherbedarf für die Zwischenergebnisse und der Mehraufwand (vgl. Tab. 2.3 auf S. 37) verursacht durch die Erweiterung des Zählers und Nenners bei Additionen und Subtraktionen.
Um diesen Aufwand zu reduzieren, wurde die bereits genannte projektive Ebene eingeführt (vgl. Kap. 2.6 auf S. 34 ff.). Die Unterschiede wurden in Kapitel 5.2 und 5.3 dargestellt.

Dies waren die getroffenen Realisierungsentscheidungen. An dieser Stelle sei noch einmal explizit erwähnt, dass diese Auswahl nur für die Rahmenbedingungen des Akzelerators Gültigkeit besitzen. Effektive Verbesserungen der Skalarmultiplikation, wie etwa durch vorberechnete Punkte auf der Kurve oder die Verwendung bestimmter Verfahren für Additions- und Subtraktionsketten, wie sie im Kapitel 4.4 (vgl. S. 58 ff.) beschrieben wurden, scheiden ebenfalls wegen Speichermangels aus.

Ein weiteres Ergebnis dieser Arbeit ist die Problematisierung, welche Teile von einem Akzelerator realisiert werden sollen und welche Funktionen die umgebende Hardware ausführen

soll. Zur Erinnerung sei hier noch einmal die Struktur des Akzelerators aus Abb. 6.1 Kapitel 6 aufgezeigt:

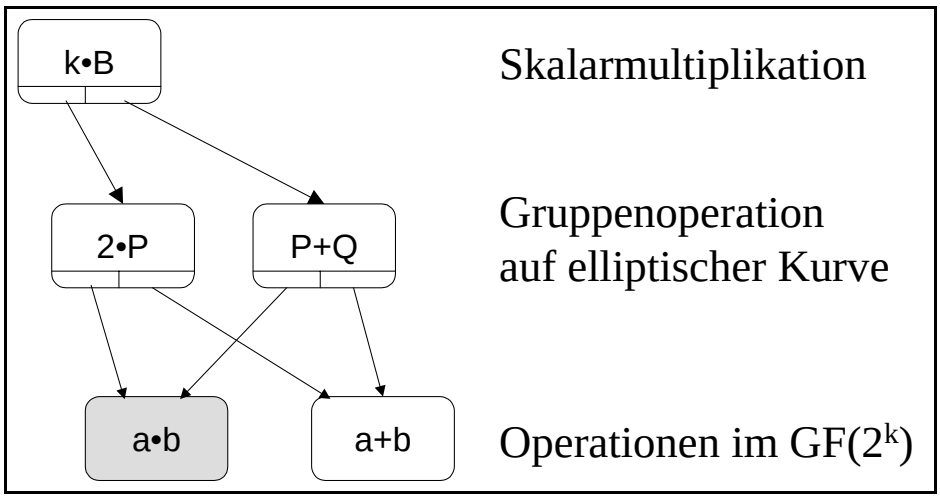

Abbildung 6.1: Berechnungshierarchie der Skalarmultiplikation

Abb. 6.1 zeigt die drei wesentlichen Stufen der Skalarmultiplikation: Auf der obersten Stufe ist die Skalarmultiplikation, sie benötigt die Addition und Verdopplung von Punkten auf der elliptischen Kurve. Diese Operationen benötigen ihrerseits die Multiplikation and Addition von Elementen des GF(2^k).
Es hat sich gezeigt, dass die GF(2^k)-Multiplikation als *Kern* des Akzelerators angesehen werden kann, hierin verbirgt sich die komplizierteste Einheit. Die GF(2^k)-Addition hingegen besteht lediglich aus sehr elementaren XOR-Einheiten und ist in Hardware *praktisch* ohne Kosten realisierbar. Die oberen beiden Stufen in Abb. 6.1 bestehen hauptsächlich aus endlichen Automaten und Speicherregistern.
Der Akzelerator kann nun direkt die Skalarmultiplikation realisieren, dies ist die direkte Umsetzung der Aufgabe. Je nach der umgebenden Hardware des Akzelerators, kann es sinnvoll sein, die Aufgaben der oberen beiden Stufen von der Umgebung realisieren zu lassen und lediglich die GF(2^k)-Multiplikation in dem Akzelerator zu implementieren.
Als Ergebnis sei jedoch festgehalten, dass ein Akzelerator mindestens die GF(2^k)-Multiplikation beinhalten muss.

Struktur des GF(2^k)-Multiplizierers

Nach der Beschreibung der Grob- oder Systemstruktur soll nun kurz die Struktur des GF(2^k)-Multiplizierers rekapituliert werden. Der Multiplizierer muss genau genommen zwei Aufgaben bewältigen:
Die **Multiplikation** und die **Reduktion** des Multiplikationsergebnisses.
Beide Operationen können nacheinander ausgeführt werden. In diesem Fall ist das Zwischenergebnis jedoch fast doppelt so groß, benötigt also entsprechenden Platz.
Neben der sequentiellen Ausführung können die beiden Operationen auch ineinander verwoben werden. Diese Variante benötigt die geringste Layoutfläche und wurde aus diesem Grund implementiert. Ein Nachteil liegt darin, dass die Ausführung der GF(2^k)-Multiplikation

sequentiell erfolgt. Es ist möglich parallele Verfahren auf Kosten der Layoutfläche zu implementieren (vgl. Kap. 4.9 auf S. 74).
Problematisch bei den Ansätzen für die Parallelisierung ist die Kombination der Multiplikation und Reduktion. Während für Multiplikationen mehrere parallele Ansätze existieren, so stand für die Reduktion nur ein brauchbarer Ansatz zur Verfügung. Und dieser Ansatz ist nur mit speziellen irreduziblen Polynomen anwendbar (vgl. Kap. 4.8 S. 73 ff.).

Auf der Implementierungsebene des GF(2^k)-Multiplizierers hat sich das rückgekoppelte Schieberegister als zentraler Bestandteil herausgebildet. Derartige Register sind einfach und besitzen eine hohe Regularität.
Auf der Implemetierungsebene ist eine Annäherung an reguläre Strukturen generell erstrebenswert. Ein derartiger Ansatz lässt sich effizienter auf ein Chip-Layout abbilden. Die optimale Umsetzungsform für hochreguläre Strukturen ist der Full-Custom-Entwurf.

Implementierungsergebnisse des ASICs

Die hier untersuchten Akzelerator-Strukturen zeigen mit welcher Fläche und welcher Geschwindigkeit Operationen auf elliptischen Kurven beschleunigt werden können.
Die Flächenabschätzungen der ASIC-Varianten und deren Leistungsdaten wurden in Kapitel 7 beschrieben. Wichtiges Resultat aus diesem Kapitel ist die mutmaßliche Konvergenz der Fläche pro Bit des Skalarmultiplizierers an einen Wert von ca. 0,25 mm² (vgl. Abb. 7.3 auf S. 100). Hier spiegelt sich die Regularität der Implementierung wieder. Zugleich bedeutet dies, dass Abschätzungen und Hochrechnungen auch für größere Bitbreiten möglich sind und dass deren Fläche nur linear mit der Bitbreite ansteigt.

Ein weiteres erstaunliches Resultat ist die hohe Güte der Synthese-Tools bei der Abschätzung der Chipfläche bereits vor dem Layouten. Dies wurde in einem Vergleich der Abschätzungen des Design-Compilers und des Platzierungs- und Verdrahtungstools (Cadence) deutlich.

8.2 Ausblick

In dieser Arbeit konnten die Auswirkungen der Mehrebenenverdrahtung für das VLSI-Layout nicht getestet werden. Bei den Implementierungen wurde eine Zweilagenverdrahtung zugrunde gelegt, mit Mehrebenenverdrahtungen sind mindestens drei und maximal fünf Ebenen gemeint. Mehr als fünf Ebenen bringen keine wesentlichen Verbesserungen im Vergleich zu den erhöhten Herstellungskosten.
Es sprechen jedoch Gründe für eine erhebliche Flächeneinsparung, bei Verwendung von Mehrebenenverdrahtung. Die Gründe sind im einzelnen:

- Das Modell enthält relativ wenig Logik im Vergleich zur Fläche für die Verdrahtung
- über zwei Drittel der Fläche wurden von der Verdrahtung belegt.

Systolische Arrays

In dieser Arbeit wurden zwar Hinweise für Verwendung von regulären Strukturen gefunden und eingesetzt. Den Untersuchungen lag jedoch kein paralleles Konzept, wie beispielsweise systolischen Arrays, zugrunde.
Eine gezielte Untersuchung auf die Abbildung der Berechnung auf systolische Arrays scheint jedoch sehr vielversprechend zu sein. Die Struktur der systolischen Arrays ist von *Natur* aus hochregulär und erlaubt einen kompakten Full-Custom-Entwurf.
Zudem gibt es in systolischen Arrays, abgesehen von Synchronisationsleitungen, nur direkte Verbindungen benachbarter Zellen. Hierdurch verringert sich die maximale Leitungslänge und die maximale Taktrate würde sich erhöhen.
Zusammenfassend gesagt, bedeutet die Anwendung systolischer Arrays, dass das Design noch regulärer und somit weniger Fläche beanspruchen und ggf. höhere Taktraten erlauben würde.

Verteiltes Berechnungskonzept

Unabhängig von systolischen Arrays können auch Teile des Akzelerators parallelisiert werden. Dies kann die Ausführungszeit ebenfalls erheblich reduzieren. Zur Realisierung ist jedoch die Hinzunahme von weiterer Designfläche notwendig oder es müssen Einschränkungen bezüglich der zu verwendeten Systemparameter gemacht werden. Ein derartiger Ansatz wurde bereits in Kapitel 4.7 (S. 68 ff.) unter Besonderheiten des irreduziblen Polynoms betrachtet.
Eine weitere Möglichkeit besteht in der Verwendung mehrerer GF(2^k)-Multiplizierer. Hierbei wäre die Frage zu klären mit welcher Anzahl von Multiplizierern ein optimales Preis-Leistungsverhältnis erreicht wird.

Die zentrale Bedeutung des GF(2^k)-Multiplizierers für den Durchsatz des Gesamtsystems wird durch seinen Anteil (Tab. 7.3) am Gesamtaufwand der Skalarmultiplikation (7.3) deutlich:

$$Zyklen(n) = 72n^2 + 45,5n + 11 \tag{7.3}$$

Eine Beschleunigung des GF(2^k)-Multiplizierers um den Faktor 2 bringt nahezu eine Verdopplung des Durchsatzes:

$$\frac{37}{72} \approx 51,4\% \tag{7.5}$$

Dies unterstreicht, weshalb eine Verbesserung des Multiplizierers die größten Performanceverbesserungen verspricht.

9 Eidesstattliche 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.

Hamburg, den 1.2.2000

_____

(Siegmund Gorr)

Danksagung

Mein Dank gilt meinen Betreuern Herrn Dr. habil. Reinhard Rauscher (Arbeitsbereich Technische Grundlagen der Informatik, Universität Hamburg) und Herrn Prof. Dr. Valk (Arbeitsbereich Theoretische Grundlagen der Informatik, Universität Hamburg), die mir die Möglichkeit gaben, dieses interessante Thema zu bearbeiten.

Ebenfalls meinem Dank gebührt Herrn Dr. Steffen Drews (Philips Semiconductors GmbH Hamburg), von dem ich Impulse und Reflektionen über die praktische Anforderungen an Kryptoakzeleratoren bekommen habe.

Weiterer Dank gilt dem gesamten Arbeitsbereich Technische Grundlagen der Informatik der Universität Hamburg, insbesondere Herrn Andreas Mäder, für die Unterstützung bei der Verwendung des Synopsys und Cadence Entwicklungssystems.

Ausdrücklich möchte ich mich auch bei Herrn Sebastian Wallner für die kritische Diskussion und Durchsicht dieser Arbeit danken.

Literaturverzeichnis

[ANS98] ANSI. *ANSI X9.63. Public Key Cryptography for the Financial Services Industry: Elliptic Curve Key Agreement and Key Transport Schemes.* Draft 2.0. Nov. 1998.

[BOH97] Bohnsack, Frank. *Untersuchung von Elliptischen Kurven für die Tauglichkeit zur Hardwareakzeleration von Kryptoverfahren.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik: 1997.

[CC87] Chudnovsky, D.V. and G.V. Chudnovsky. "Sequences of Numbers Generated by Addition in Formal Groups and New Primality and Factorizations Tests". *Advances in Applied Mathematics*, 7 (1987): 385 - 434.

[CS97] Cai, Jin-Yi und D. Sivakumar. „Resolution of Hartmanis' Conjecture for NL-hard sparse sets". *3rd Annual International Computing and Combinatoric Conference (COCOON).* Lecture Notes in Computer Science 1276 (1997): 62-71.

[D7-98] IEEE. *IEEE P1363. Standard Specifications for Public Key Cryptography.* Draft Version 7. New-York: IEEE, 1998.

[DTV82] *dtv-Atlas zur Mathematik.Tafeln und Texte. Grundlagen Algebra und Geometrie.* Hg. Fritz Reinhardt und Heinrich Soeder, Bd. 1. 5. Aufl. München: dtv-Verlag, 1982.

[FOR96] Forster, Otto. *Algorithmische Zahlentheorie*. Braunschweig/ Wiesbaden: Vieweg: 1996.

[GOR98] Gordon, Daniel M. „A survey of fast exponentiation methods". *Journal of Algorithms* 27.1 (1998): 129 - 146.

[HAM98] Hamdy, Safuat. *Annotationes de Rationibus Cryptologiae. Anwendungen elliptischer Kurven in der Kryptologie*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik: 1998.

[HUS87] Husemöller, Dale. *Elliptic Curves*. Graduate Texts in Mathematics 111. New York [u.a.]: Springer, 1987.

[JÄN91] Jänich, Klaus. *Lineare Algebra*. 4. Aufl. Berlin [u.a.]: Springer-Verlag, 1991.

[KNU98] Knuth, D. E. *The Art of Computer Programming. Volume 2. Seminumerical Algorithms*. 3. Aufl. Massachusetts [u.a.]: Addison-Wesley, 1998.

[KO63] Karatsuba, A. und Yu Ofman. „Multiplication of multidigit numbers on automata". *Soviet Physics Doklay* 7 (1963): 595-596.

[KOB98] Koblitz, Neal. *Algebraic Aspects of Cryptography. With an Appendix on Hyperelliptic Curves by Alfred J. Menezes, Yi-Hong Wu, and Robert J. Zuccherato.* Algorithms and Computation in Mathematics 3. Berlin [u.a.]: Springer, 1998.

[LEU96] Leutbecher, Armin. *Zahlentheorie. Eine Einführung in die Algebra*. Berlin [u.a.]: Springer, 1996.

[LID82] Lidl, Rudolf. *Angewandte abstrakte Algebra.* Bd. 2. Mannheim [u. a.]: Bibliographisches Institut: 1982.

[LN94] Lidl, Rudolf, und Harald Niederreiter. *Introduction to finite fields and their applications.* Cambridge Press: 1994.

[LV99] Lenstra, Verheul. *Selecting Cryptographic Key Sizes*. November 1999.

[MEN93] Menezes, A. J. *Elliptic Curve Public Key Cryptosystems*. Boston: Kluwer Academic Publishers, 1993.

[MO90] Morain, François und Jorge Olivos. „Speeding up the computations on an elliptic curve using addition-subtraction chains“. *Inform. Theor. Appl.* 24 (1990): 531-543.

[MOV93] Menezes, A. T. Okamoto und S. Vanstone. „Reducing elliptic curve logarithms to logarithms in a finite field“. *IEEE Transactions on Information Theory* 39 (1993): 1639-1646.

[MP98] Müller, V. und S. Paulus. *Elliptische Kurven und Public Key Kryptographie*. Darmstadt: Online, 1998.

[MUE95] Müller, V. *Ein Algorithmus zur Bestimmung der Punktzahl elliptischer Kurven über endlichen Körpern der Charakteristik größer drei*. Diss. Universität Saarbrücken: 1995.

[REIF98] Reifschneider, N. *CAE-gestützte IC-Entwurfsmethoden.* München [u.a.]: Prentice-Hall, 1998.

[SCH96] Schneier, B. *Angewandte Kryptographie. Protokolle, Algorithmen und Sourcecode in C.* Bonn [u.a.]: Addison-Wesley, 1996.

[SIL86] Silverman, Joseph H. *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics 106. New-York [u.a.]: Springer, 1986.

[SIL92] Silverman, Joseph H., und John Tate. *Rational Points on Elliptic Curves.* Undergraduate Texts in Mathematics. New-York [u.a.]: Springer, 1992.

[SS71] Schönhage, A. und V. Strassen. „Schnelle Multiplikation großer Zahlen“. *Computing* 7 (1971): 281-292.

[vL91] van Lint, J. H. *Introduction to Coding Theory.* Springer, 1991.

[WIE98] Wiener, M. J. „Performance Comparison of Public-Key Cryptosystems“. *RSA Laboratories CryptoBytes* 4 (1998). Online-Letter: RSA-Laboratories, 1998.

[WOB97] Wobst, Reinhard. *Abenteuer Kryptologie. Methoden, Risiken und Nutzen der Datenverschlüsselung*. Bonn [u.a.]: Addison-Wesley, 1997.

[WZ98] Wiener, M. J., und R. J. Zuccherato. *Faster Attacks on Elliptic Curve Cryptosystems.* 1998.

[ZEI96] Zeidler, E. *Teubner-Taschenbuch der Mathematik*: *Begr. von I. N. Bronstein und K. A. Semendjajew. Weitergeführt von G. Grosche, V. Ziegler und D. Ziegler.* Hg. E. Zeidler. 1.Bd. Stuttgart [u.a.]: Teubner, 1996.

1 Elliptische Kurven

1.1 Erweiterter Euklidischer Algorithmus

Der erweiterte euklidische Algorithmus dient zur effizienten Berechnung des multiplikativen Inversen in endlichen Körpern.

Berechnet wird: $u\,u_1 + v\,u_2 = u_3 = \gcd(u, v)$

1. $(u_1, u_2, u_3) \leftarrow (1, 0, u)$
   $(v_1, v_2, v_3) \leftarrow (0, 1, v)$
2. **if** $v_3 = 0$ **then stop with** (u_1, u_2, u_3)
3. $q \leftarrow \lfloor u_3 / v_3 \rfloor$ (Division, Rest wird ignoriert)
   $(t_1, t_2, t_3) \leftarrow (u_1, u_2, u_3) - (v_1, v_2, v_3) \cdot q$
   $(u_1, u_2, u_3) \leftarrow (v_1, v_2, v_3)$
   $(v_1, v_2, v_3) \leftarrow (t_1, t_2, t_3)$
4. **goto** 2

Diese lineare Darstellung ist [KNU98] entnommen.

Beispiel für GF(2^4):

Berechnung von $gcd(x^4+x+1, x^2+x)$

1. $(u_1, u_2, u_3) \leftarrow (1, 0, x^4+x+1)$
   $(v_1, v_2, v_3) \leftarrow (0, 1, x^2+x)$
2. $v_3 = x^2+x$
3. $q \leftarrow \lfloor x^4+x+1 \,/\, x^2+x \rfloor = x^2+x+1$
   $(t_1, t_2, t_3) \leftarrow (1, 0, x^4+x+1) - (0, 1, x^2+x) \cdot q = (1, x^2+x+1, 1)$
   $(u_1, u_2, u_3) \leftarrow (0, 1, x^2+x)$
   $(v_1, v_2, v_3) \leftarrow (1, x^2+x+1, 1)$
4. $v_3 = 1$
5. $q \leftarrow \lfloor x^2+x \,/\, 1 \rfloor = x^2+x$
   $(t_1, t_2, t_3) \leftarrow (0, 1, x^2+x) - (1, x^2+x+1, 1) \cdot q = (x^2+x, x^4+x+1, 0)$
   $(u_1, u_2, u_3) \leftarrow (1, x^2+x+1, 1)$
   $(v_1, v_2, v_3) \leftarrow (x^2+x, x^4+x+1, 0)$
6. $v_3 = 0$, stop with $(1, x^2+x+1, 1)$

$gcd(x^4+x+1, x^2+x) = 1 = (x^4+x+1) \cdot (x^2+x+1) + (x^2+x) \cdot 1$

Das multiplikative Inverse von (x^2+x) ist also (x^2+x+1).

1.2 J-Invariante und Diskriminante

Die **J-Invariante** beschreibt einen Teil der Gleichung, die unter allen isomorphen Abbildungen invariant ist. Dies ist jedoch nicht ihre einzige Bedeutung. Sie wird zudem herangezogen, um ein wichtiges Kriterium für die Kryptographie zu beschreiben. Und zwar sind Kurven mit j-Invariante null, so genannte supersinguläre Kurven. Für diese Kurvenart ist die Berechnung des diskreten Logarithmus einfacher, sie sind somit unsicherer und sollten daher nicht verwendet werden. Dies betrifft auch die erste Form für Kurven im GF(2^m).

Neben der J-Invarianten existiert noch die **Diskriminante** als eine weitere Charakterisierungsgröße für elliptische Kurven, sie wird mit Δ bezeichnet. Die Diskriminante darf nicht null werden, in diesem Fall liegt eine singuläre Kurve vor, für die die Gruppenoperation nicht definiert ist.

$$
\begin{aligned}
j &= c_4^3/\Delta : c_4 = b_2^2 - 24b_4 \\
b_2 &= a_1^2 + 4a_2 \\
b_4 &= 2a_4 + a_1a_3 \\
b_6 &= a_3^2 + 4a_6 \\
\Delta &= -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6 \\
b_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2
\end{aligned}
$$

1.3 Substitutionen für Körper der Charakteristik zwei

Für Körper der Charakteristik 2 können, in Abhängigkeit von *j(E)*, zwei verschiedene Varianten angegeben werden. (j, die so genannte j-Invariante, bezeichnet eine Invariante bezüglich aller möglichen Substitutionen, sie wird weiter unten definiert. Neben der Invarianten existiert noch die Diskriminante, sie ist mit Δ bezeichnet.) Dazu muss zunächst die J-Invariante für Körper der Charakteristik 2 berechnet werden, sie lautet: $j(E) = a_1^{12}/\Delta$.

Fall	Substitution $(x,y) \vdash$	Ergebnis
$j = 0$	$(\tilde{x} + a_2, \tilde{y})$	$y^2 + b_3y = x^3 + b_4x + b_6$
$j \neq 0$	$(a_1^2\tilde{x} - a_3/a_1, a_1^3\tilde{y} + (3a_3^2 + a_1^2a_4)/a_1^3)$	$y^2 + xy = x^3 + b_2x^2 + b_6$

Die Zusammenfassung der neuen Koeffizienten vor den Variablen erfolgte mit gleicher Index-Systematik, wie bei der Ausgangsgleichung. Um auf die Abweichung gegenüber der allgemeinen Form hinzuweisen wurde der Buchstabe *b* statt *a* verwendet. Die Konstanten sind für jeden der obigen Fälle unterschiedlich. Sie können am Ende der jeweiligen Substitution entnommen werden.

Für $j = 0$ lautet die Substitution $(x,y) \vdash (\tilde{x} + a_2, \tilde{y})$ mit $a_1 = 0$, für $j \neq 0$ ist die Substitution

$$(x,y) \vdash (a_1^2\tilde{x} - a_3/a_1, a_1^3\tilde{y} + (3a_3^2 + a_1^2a_4)/a_1^3)$$

und $a_1 \neq 0$.
Die Umformungsergebnisse aus der allgemeinen Gleichung lauten:

$j = 0$: $y^2 + a_3y = x^3 + (a_2^2 + a_4)x + (a_2a_4 + a_6)$
$j \neq 0$: $y^2 + xy = x^3 + b_2x^2 + b_6$

Umformung $j = 0$ Substitution $(x,y) \vdash (\tilde{x} + a_2, \tilde{y})$

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$
$$\vdash y^2 + a_3y = x^3 + 3a_2x^2 + 3xa_2^2 + a_2^3 + a_2x^2 + 2a_2^2x + a_2^3 + a_4x + a_2a_4 + a_6$$
$$\vdash y^2 + a_3y = x^3 + 4a_2x^2 + (5a_2^2 + a_4)x + 2a_2^3 + a_2a_4 + a_6$$
$$\vdash y^2 + a_3y = x^3 + (a_2^2 + a_4)x + a_2a_4 + a_6$$
$$\vdash y^2 + b_3y = x^3 + b_4x + b_6$$

Bei der Substitution soll der quadratische x-Term verschwinden. Bei einem Körper der Charakteristik 2, entsteht ein Faktor vor diesem Term, der ein Vielfaches von 2 ist, wodurch der Term verschwindet.

Umformung $j \neq 0$ Substitution $(x,y) \vdash (a_1^2\tilde{x} - a_3/a_1, a_1^3\tilde{y} + (3a_3^2 + a_1^2a_4)/a_1^3)$

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$
$$\vdash^1 y^2 + a_1^3xy = a_1^6x^3 + (a_1^4a_2 - 3a_1^3a_3)x^2 + (3a_3^2 + a_1^2a_4)x + \varepsilon$$
$$\varepsilon = a_2\frac{a_3^2}{a_1^2} - \frac{a_3^3}{a_1^3} - \frac{a_3a_4}{a_1} + a_6$$
$$\vdash y^2 + a_1^3xy = a_1^6x^3 + (a_1^4a_2 - 3a_1^3a_3)x^2 + (3a_3^2 + a_1^2a_4)x + \varepsilon$$
$$\vdash^2 a_1^6y^2 + a_1^6xy = a_1^6x^3 + (a_1^4a_2 - 3a_1^3a_3)x^2 + \varepsilon - (a_1^4a_4^2 + 3^2a_3^4)/a_1^6$$
$$\vdash y^2 + xy = x^3 + x^2(a_1^4a_2 - 3a_1^3a_3)/a_1^6 + \varepsilon/a_1^6 - (a_1^4a_4^2 + 3^2a_3^4)/a_1^{12}$$
$$\vdash y^2 + xy = x^3 + b_2x^2 + b_6$$

Die Substitution erfolgt in zwei Schritten, zuerst die *x*-Variable und anschließend die *y*-Variable. Vereinfachungen ergeben sich hier durch Eleminierung des *y*-Terms und anschließend durch Elemination des *x*-Terms sowie durch Wegfälle im Körper. So werden zwei Konstanten zu null und eine zu eins.

1.4 Normalformen für Körper der Charakteristik 3

Hier gibt es in Abhängigkeit von der *J*-Invarianten zwei mögliche Normalformen:

J	Normalform	Diskriminante
$j = 0$	$y^2 = x^3 + b_4 x + b_6$	$\Delta = -b_4^3$
$j \neq 0$	$y^2 = x^3 + b_2 x^2 + b_6$	$\Delta = -b_2^3 b_6 = j$

Die Substitutionsschritte sind weggelassen, da sie hier nicht interessieren. Sie entsprechen aber der quadratischen Auflösung der *y*-Variablen und einer linearen Substitution für den Fall, dass die J-Invariante ungleich null ist.