Reconfigurable Computing am Beispiel ECDSA

Philipp Kewisch

Fachhochschule Wedel
Fachrichtung Technische Informatik

Referent: Prof. Dr. Sergei Sawitzki
Abgabedatum: 18. November 2016

Inhaltsverzeichnis

1. Einleitung

Zur Problemlösung in einem Gesamtsystem stehen vielseitige Vorgehensweisen zur Verfügung. Durch die Softwareentwicklung lässt sich eine Lösung entwerfen, die stark auf das Problemfeld zugeschnitten ist, sich aber dennoch flexibel gestalten lässt. Der Aufwand für Anpassungen ist gering und neue Anforderungen lassen sich simpel umsetzen.

Bei leistungskritischen Problemfeldern ist jedoch der Einsatz einer Hardwarelösung zu betrachten. Neben dem Einsatz von spezialisierten Prozessoren, die bestimmte mathematische Operationen beschleunigen, können mit Hilfe von ASICs maßgeschneiderte Lösungen erstellt werden, die einen großen Leistungsvorteil gegenüber einer allgemeinen Softwarelösung ermöglichen.

Im Themengebiet *Reconfigurable Computing* wird eine Zwischenlösung gesucht, die die Leistungsvorteile aus der Hardwareentwicklung mit sich bringt, aber dennoch die Flexibilität einer Softwarelösung bereitstellt (Compton 2002).

In dieser Arbeit wird das Themengebiet an Hand eines Beispiels aus der Verschlüsselungstechnik betrachtet. Es wird für den ECDSA Algorithmus eine Softwarelösung vorgestellt, die sich mit einfachen Mitteln in Hardware umsetzen lässt. Die Einschränkung auf Koblitzkurven begünstigt dabei die Umsetzung. Der Focus dieser Arbeit liegt bei der Softwarelösung, es werden jedoch Algorithmen und Verfahren vorgestellt, die eine Optimierung auf der Hardwareebene ermöglichen.

Die Abkürzung ECDSA steht für „Elliptic Curve Digital Signature Algorithm“. Es handelt sich dabei um ein Verfahren zur digitalen Signatur von Nachrichten, bei dem der Absender der Nachricht zweifelsfrei bestätigt werden kann.

Das Verfahren basiert auf den mathematischen Eigenschaften von elliptischen Kurven und gewinnt gegenüber klassischen Verfahren wie RSA und DSA immer mehr an Popularität. Die Schlüssellänge ist bei gleichen Sicherheitsanforderungen deutlich geringer als bei RSA und DSA.

Nachfolgend wird zunächst in die Theorie von elliptischen Kurven eingeführt und die mathematischen Grundlagen betrachtet. Es wird zudem in die Algorithmen eingeführt, die für die Berechnung notwendig sind. Im Anschluß wird die im Rahmen dieser Arbeit entwickelte Softwarelösung vorgestellt und ein Ausblick darauf gegeben, welche weitere Arbeit folgen kann.

2. Grundlagen

In diesem Kapitel werden die notwendigen Grundlagen vorgestellt, die für den Einsatz von elliptischen Kurven zum Erstellen von digitalen Signaturen notwendig sind.

Nach einer allgemeinen Einführung in asymmetrische Kryptosysteme werden finite Felder und elliptische Kurven besprochen, die zusammen einen Ausgangspunkt für die Umsetzung eines Kryptosystems basierend auf elliptischen Kurven darstellen.

2.1. Asymmetrische Kryptosysteme

Beim ECDSA Verfahren werden sogenannte Private und Public Keys eingesetzt. Der Private Key ist geheim und nur dem Versender der Nachricht bekannt, während der Public Key durch Veröffentlichung dem Empfänger bekannt gemacht werden kann.

Die Schlüssel stehen derart in einer Beziehung zueinander, dass der Empfänger der Nachricht mit Hilfe des öffentlichen Schlüssels verifizieren kann, ob sie vom Versender mit dem geheimen Schlüssel signiert wurde. Das Verfahren lässt sich auch für die Verschlüsselung von Daten verwenden.

Die Beziehung zwischen den Schlüsseln wird über eine *Falltürfunktion* beschrieben. Es handelt sich um eine spezielle Art von Einwegfunktion. Die Besonderheit einer Einwegfunktion besteht darin, dass die Funktion einfach zu berechnen ist, das Lösen der Umkehrfunktion jedoch um ein Vielfaches schwieriger ist. Darüberhinaus lässt sich eine Falltürfunktion mit der Kenntnis einer einzigartigen Informationsmenge leicht umkehren (Krawczyk 1998, S.284).

Im Bezug auf die Verschlüsslung kann mit Hilfe des Public Keys des Empfängers eine Nachricht kodiert werden. Die Berechnung des verschlüsselten Wertes ist leicht durchzuführen. Die Entschlüsselung ist jedoch schwer zu erlangen, es sei denn der Private Key – die Falltür-Information – ist bekannt.

2.2. Finite Felder

Zum Einsatz der Kurven in der Kryptographie werden die elliptischen Kurven über finite Felder eingesetzt. Finite Felder sind auch als *Galois-Felder* bekannt und werden mit GF(q) oder $\mathbb{F}_q$ abgekürzt, wobei q für die Anzahl der Elemente im Feld steht. Es handelt sich um eine Abstraktion bekannter Zahlensysteme,

bestehend aus einer finiten Menge $\mathbb{F}$, sowie zwei mathematischen Operationen – der Addition und der Multiplikation. Das neutrale Element der Addition ist 0 und das neutrale Element der Multiplikation ist 1. Es muss weiterhin das Distributivgesetz gelten.

Die Operationen Subtraktion und Division werden durch die Addition bzw. Multiplikation definiert, in dem das zweite Element jeweils umgekehrt wird. Bei der Umkehrung bzgl. der Addition muss gelten $a+(-a)=0$ und bei der Multiplikation ermöglicht die Gleichung $b*b^{-1}=1$ das Finden eines inversen Elements (Hankerson 2003, S.25f).

Die Ordnung q eines Galois-Felds muss eine Primzahl sein und wird auch als *Charakteristik* von $\mathbb{F}$ bezeichnet. Es ist möglich die Ordnung q um eine Potenz m zu erweitern, bei $\mathbb{F}_{2^m}$ handelt es sich dann um ein Erweiterungsfeld.

2.2.1. Binäre Felder und polynomiale Darstellung

Finiten Feldern der Charakteristik 2 werden außerdem als binäre Felder bezeichnet. Eine besondere Art das Feld $\mathbb{F}_{2^m}$ zu konstruieren ist eine polynomiale Basisrepräsentation. Die Elemente des Feldes sind dabei binäre Polynomiale, also solche dessen Koeffizienten gerade 0 oder 1 sind, mit einem Höchstgrad von $m-1$.

Zur Reduktion der Feldelemente bei Erweiterungsfeldern muss außer dem ein *irreduzibles Polynom* angegeben werden, dessen Terme nicht als Produkt von binären Polynomen eines kleineren Grades faktorisiert werden können.

Die Addition im binären Feld mit polynomialer Darstellung wird durch die Addition der Polynome modulo 2 durchgeführt. Die Multiplikation wird modulo des irreduziblen Polynoms durchgeführt.

2.3. Elliptische Kurven

Eine elliptische Kurve E über ein finites Feld K wird mit Hilfe der Weierstraß-Gleichung definiert:

$$E: y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6$$

Die Parameter $a_1 \ldots a_6$ legen fest, welche Form die elliptische Kurve annimmt und sind Elemente des finiten Feldes K. Sie werden im Rahmen der Domain-Parameter

für den Einsatz der Kurve in der Kryptographie festgelegt. Im weiteren Verlauf dieses Abschnitts werden die Domain-Parameter genauer beschrieben.

Unter der Einschränkung, dass das Feld K die Charakteristik 2 aufweist, lässt sich die Weierstraß-Gleichung stark vereinfachen. Auf Grund der eingeschränkten Möglichkeit der Variablenveränderung lässt sich eine Substitution vornehmen. Es wird unterschieden für den Fall $a_1=0$ und $a_1\neq 0$, im Ergebnis können zwei neue Gleichungen aufgestellt werden (Hankerson 2003, S.76ff):

$$a_1\neq 0: y^2+xy=x^3+ax^2+b \qquad a_1=0: y^2+cy=x^3+ax+b$$

Diagramm 2.1 zeigt die Kurve für $a_1\neq 0$ mit $a=1, b=1$ in affinen Koordinaten.

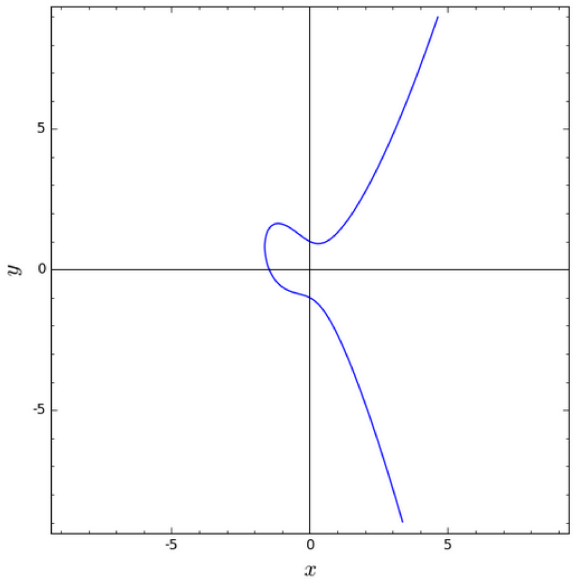

Diagramm 2.1: Kurve mit $y^2+xy=x^3+x^2+1$

Neben den Werten für die x- und y-Koordinaten der Gleichung wird weiterhin ein Punkt im Unendlichen definiert und der Punktmenge auf E zugefügt. Der Punkt ist unter anderem für die Addition eines Punkts P mit dem durch die horizontale Achse gespiegelten Punkt $-P$. Die Addition geschieht durch Verbinden von Punkten auf der Kurve. Verbindet man P mit $-P$, entsteht eine vertikale Linie, die die Kurve kein weiteres Mal schneidet. Der Ergebnispunkt wird in diesem Fall auf ∞ festgelegt.

Weitere Anwendungsfälle, sowie Details zur Punktaddition werden in Abschnitt 3.5.3 ausgeführt.

2.3.1. Domain-Parameter

Die vereinfachen Gleichungen bilden die Basis für die Auswahl einer Kurve für den Einsatz in der Kryptographie. Die Auswahl der Kurve geschieht durch die Festlegung der Variablen a, b und c, sowie weiteren Parametern die nachfolgend aufgeführt werden:

- q: Die Ordnung des Feldes $\mathbb{F}$, z. B. $q=2^m$ für ein Binärfeld
- FR: Die Feldrepräsentation für die Elemente von $\mathbb{F}_q$, z. B. Polynomialdarstellung
- S: Für zufällig generierte Kurven, den Seed-Wert für die Zufallszahlengenerierung.
- a,b: Die Parameter zur Definition der Kurvengleichung, wie im vorherigen Abschnitt beschrieben.
- P: ein Basispunkt auf der Kurve, in affinen Koordinaten. Der Punkt wird auch *Generatorpunkt* genannt, da durch Vervielfachung des Punktes alle anderen Punkte der Kurve erzeugt werden können.
- n: Die Ordnung des Punktes P.
- h: Der Kofaktor $h=\#E\left(\frac{\mathbb{F}_q}{n}\right)$, der einen möglichst kleinen Wert annehmen sollte.

Um zu gewährleisten, dass die Verschlüsselung bzw. Signierung möglichst sicher ist, müssen einige Rahmenbedingungen bei der Auswahl der Domain-Parameter eingehalten werden. Verschiedene Sicherheitsgruppen haben Vorschläge für sichere Parameterwerte veröffentlicht, die weitverbreitet verwendet werden.

Von einer eigenständigen Auswahl von Domain-Paramater wird ohne ein tiefergehendes mathematisches Verständnis des Themengebiets abgeraten, da die Auswahl falscher Parameter die Sicherheit des Algorithmus beeinträchtigen kann.

2.3.2. Koblitzkurven

Eine Sonderkategorie von Kurven sind die sogenannten *Koblitzkurven*, auch bekannt als *anomalous binary curves*. In der Weierstraß-Gleichung wird dafür der Fall

$a_1 \neq 0$ ausgewählt, mit der weiteren Besonderheit, dass $b=1$ und $a=0$ oder 1. Die Kurven werden über das Erweiterungsfeld $\mathbb{F}_{2^m}$ definiert, wodurch eine weitere Vereinfachung der Arithmetik möglich ist (Cohen 2005, S.356).

Folglich bestehen zwei Kategorien von Koblitzkurven:

$$E_0 : y^2 + xy = x^3 + 1$$
$$E_1 : y^2 + xy = x^3 + x^2 + 1$$

Die Ordnung der Kurven ist eine Fastprimzahl, wenn $\#E_a(F_{2^m}) = hn$ gilt, n eine Primzahl ist und h den Wert 4 bei E_0 und 2 bei E_1 annimmt. Die Werte n und h sind somit in den Domain-Parametern festgelegt. Koblitzkurven dieser Art sind besonders förderlich für den Einsatz in der Kryptographie (Hankerson 2003, S.114).

Ein weiterer Vorteil ist eine optimierte Berechnung der Punktmultiplikation. Durch ein angepasstes Verfahren lässt sich ein Geschwindigkeitsvorteil besonders bei der Multiplikation mit Vielfachen von 2 erlangen.

2.3.3. Falltürfunktion für Elliptischen Kurven

Zum Einsatz von elliptischen Kurven in der Kryptographie, ist eine in Abschnitt 2.1 vorgestellte Falltürfunktion notwendig. Die benötigte Funktion ist die Lösung des diskreten Logarithmus Problems einer abelschen Gruppe, die auf der elliptischen Kurve definiert wird.

Betrachtet man zwei Elemente der Gruppe r und q, sowie eine hinreichend große Primzahl p, gilt es die nachfolgende Gleichung nach der Zahl k aufzulösen:

$$r = q\,k \bmod p$$

Bei der Anwendung des diskreten Logarithmus Problems auf elliptische Kurven wird die Punktmultiplikation betrachtet. Bei der Multiplikation einer Zahl k mit einem Punkt P auf der Kurve ergibt sich der Ergebnispunkt Q. Diese Beziehung wird auch als der diskrete Logarithmus von Q zur Basis P bezeichnet.

Bei der Schlüsselerzeugung in Kryptosystemen mit elliptischen Kurven bezeichnet k den privaten Schlüssel, während Q den öffentlichen Schlüssel darstellt. Der Punkt P ist ein spezieller Basispunkt auf der Kurve, der durch die Domain-Parameter der Kurve festgelegt ist.

Die Domain-Parameter sind vorgegeben, so ist der Wert von P bekannt, jedoch ist es nicht mit vertretbarem Aufwand möglich den Wert von P unter Kenntnis von k und Q zu errechnen.

3. Umsetzung

Im Rahmen der Ausarbeitung wurde eine Softwarelösung entworfen, die die wichtigsten Kernoperationen von ECDSA implementiert. Dazu gehören neben den mathematischen Grund- und Punktoperationen auch die ECDSA-Operationen zur Schlüsselerzeugung, zum Signieren und zum Verifizieren von Nachrichten. Es werden weiterhin Testfälle beschrieben, die zur Sicherstellung der korrekten Operation eingeführt worden sind.

Um eine Vergleichsbasis zwischen der im Zuge dieser Arbeit entstandenen Softwarelösung und einer zukünftigen Hardwarelösung zu schaffen wurde darauf geachtet möglichst Algorithmen und Programmkonstrukte zu verwenden, dessen Umsetzung auf maßgeschneiderter Hardware unter geringem Ressourceneinsatz zu bewerkstelligen ist.

Mit vergleichbaren Leistungswerten zwischen Software und Hardware wäre eine Basis geschaffen, um sowohl für Software auf einem bestimmten Mikroprozessor als auch für eine Hardwarelösung mit einem ASIC Optimierungen für die jeweilige Architektur durchzuführen.

Dahingehend werden Optimierungsmöglichkeiten im Anschluss im Ausblick besprochen.

3.1. Grundlegender Ablauf

Nachfolgend wird der Ablauf zum Signieren und Verifizieren von Nachrichten überblicksartig beschrieben und die notwendigen mathematischen Operationen aufgeführt. Im weiteren Verlauf des Kapitels werden die Operationen im Detail beleuchtet.

Als Grundvoraussetzung zum Signieren einer Nachricht wird ein Schlüsselpaar benötigt, wie bereits in Abschnitt 2.1 eingeführt. Es entsteht ein geheimer und ein öffentlicher Schlüssel.

Mit dem entstandenen geheimen Schlüssel kann anschließend eine Nachricht signiert werden. Die Signatur erfolgt nicht auf der gesamten Nachricht, es wird stattdessen zunächst ein Hash-Wert der Nachricht erzeugt, der im Anschluss signiert werden kann. Das Ergebnis der Signatur ist ein Punkt auf der Kurve, der als Signaturwert mit den Parametern r und s bezeichnet wird.

In der Regel wird nun die signierte Nachricht zum Empfänger versendet, der den Absender verifizieren möchte. Der Empfänger errechnet erneut den Hash-Wert

der Nachricht. Zusammen mit dem öffentlichen Schlüssel des Versenders und der Signatur kann nun bestimmt werden, ob die Nachricht zweifelsfrei vom Versender stammt.

3.2. Auswahl der Domain-Parameter

Zur Umsetzung auf der an der Fachhochschule Wedel zur Verfügung stehenden Hardware wurden vorab einige Einschränkungen vorgenommen, die eine effiziente Ausführung begünstigen.

Durch die Wahl besonderer Domain-Parameter und Feld Charakteristiken kann die Arithmetik vereinfacht werden, um insbesondere die Laufzeit der Punktmultiplikation zu verringern.

In Abschnitt 2.3.2 wurden Koblitzkurven eingeführt, die Optimierungen zur Reduzierung der benötigten Hardwarekapazität ermöglichen. Im Zusammenhang mit den Koblitzkurven wurde weiterhin auf einen bestehenden Satz von Domain-Parametern zurückgegriffen. Mit der Auswahl der Kurve SECT163K1 (SECG 2010), auch bekannt unter NIST-K163 (NIST 1999), werden die nachfolgend vorgeschlagenen Domain-Parameter verwendet. Das in Abschnitt 2.2.1 vorgestellte zugehörige binäre Feld ist $F_{2^{163}}$, es wird daher auch ein irreduzibles Polynom angegeben.

```
p(z) = z^163 + z^7 + z^6 + z^3 + 1
q    = 0x80000000000000000000000000000000000000C9
a    = 1
Px   = 0x2fe13c0537bbc11acaa07d793de4e6d5e5c94eee8
Py   = 0x289070fb05d38ff58321f2e800536d538ccdaa3d9
n    = 0x040000000000000000020108A2E0CC0D99F8A5EF
h    = 2
```

Neben den empfohlenen Domain-Parametern wurden des Weiteren Parameter für ein Feld in $\mathbb{F}_{2^9}$ entworfen. Die Parameter werden jedoch ausschließlich in den Funktionstests verwendet und sind nicht für den Einsatz in einer sicheren Verschlüsselung oder Signierung zu verwenden. Die Parameter haben den Vorteil, dass mehr als ein Datenwort verwendet wird, m aber weiterhin Primzahl ist und die Berechnung mit Vergleichsprogrammen in zumutbaren Zeitaufwand zu erreichen ist. Folgende Parameter wurden für die Funktionstests verwendet:

```
p(z) = z^9 + z^1 + 1
q    = 0x302
a    = 1
Px   = 0xEE
Py   = 0xAF
n    = 0x604
h    = 2
```

Das irreduzible Polynom wurde aus einer Polynomtabelle (Seroussi 1998) entnommen und kommt mit einer niedrigen Gewichtung der Polynomterme aus. Zum Erlangen des Basispunkts wurde die Kurvengleichung für alle möglichen Elemente des Feldes errechnet und ein beliebiger Wert ausgewählt. Der Wert für n genügt der Gleichung $n \cdot P = \infty$ unter Berücksichtigung der in Abschnitt 3.5 vorgestellten Punktmultiplikation.

3.3. Darstellung und Architektur

Für die Darstellung der SECT163K1 Kurve wird mit der in Abschnitt 2.2.1 eingeführten Polynomialdarstellung eine Zahlendarstellung mit 163 bits benötigt, sowie einem Zustand für den Punkt im Unendlichen. Die nächstgrößere Zweierpotenz ist 256, für die es in C keinen nativen Datentypen gibt.

Daher wird eine Liste mit mehreren Datenwörtern einer festen Breite verwendet. Es wurde in der Softwarelösung eine geringe Bitbreite von 8 bits pro Wort verwendet, da hier die Testkurve in $GF(2^9)$ bereits zwei Datenwörter verwendet. Die Daten werden in *Little Endian* Reihenfolge gespeichert, das Bit mit dem niedrigsten Stellenwert ist an Position 0.

Der Vorteil zeigt sich bei den Funktionstests. Es kann sichergestellt werden, das die Algorithmen mit mehreren Datenwörtern funktionieren, ohne dass eine aufwendige Gegenrechnung durchzuführen, so wie es z. B. bei $GF(2^{67})$ der Fall wäre.

Durch die Darstellung mit mehreren Datenwörtern ist es notwendig einige grundlegende Operationen neu zu definieren. Dazu gehören unter anderem die Shift-Operationen.

3.4. Finite Feldarithmetik

In diesem Abschnitt werden die arithmetischen Operationen auf finiten Feldern vorgestellt, sowie der umgesetzte Algorithmus besprochen. Wie in Abschnitt 2.2 besprochen ist für ein finites Feld zunächst die Addition und Multiplikation definiert. Darüberhinaus werden hier die Subtraktion und Division besprochen sowie die softwaretechnisch notwendige Reduktion auf den Wertebereich.

Wie im vorherigen Abschnitt erwähnt, wird die Polynomiale Darstellung verwendet. Die Elemente des Feldes sind also Polynome, die softwaretechnisch als Binärzahlen dargestellt werden. In den weiteren Unterabschnitten wird dieses Implementierungsdetail vorausgesetzt.

3.4.1. Addition und Subtraktion

Binärfelder haben den Vorteil, dass bei der Addition kein Übertrag berechnet werden muss, was für sich bereits einen Vorteil für die Implementation in Hardware bringt. Die Addition in einem Feld $\mathbb{F}_{2^m}$ geschieht modulo 2, daher gleicht die Addition einer bitweise exklusiven OR Operation (XOR) (Carlet 2007, S.43). Die Subtraktion wird ebenso über XOR ausgeführt.

Die Umsetzung in der Software ist ohne weiteren Aufwand zu erlangen, es wird die XOR-Operation für jedes Wort ausgeführt und jeweils im Ergebniswort an der gleichen Stelle gespeichert.

3.4.2. Multiplikation

Multiplikation in einem binären finiten Feld kann mit verschiedenen Algorithmen ausgeführt werden.

In der Softwarelösung wurde ein einfacher *shift-and-add* Multiplizierer Verwendet. Das Prinzip gleicht dem Verfahren, wie es auch bei der manuellen Multiplikation auf Papier gelehrt wird. Für die Multiplikation zweier Polynome a(z) und b(z) wird an jeder Stelle in a(z), an der das Bit auf 1 gesetzt ist, eine Kopie von b(z) addiert. Die Kopie wurde zuvor um gerade so viele Bits nach links verschoben, wie die Stelle die gerade betrachtet wurde. Algorithmus 1 wurde als Referenz verwendet und beschreibt das Verfahren auf formale Weise.

INPUT: Binary polynomials $a(z)$ and $b(z)$ of degree at most $m-1$.
OUTPUT: $c(z) = a(z) \cdot b(z) \bmod f(z)$.

1. If $a_0 = 1$ then $c \leftarrow b$; else $c \leftarrow 0$.
2. For i from 1 to $m-1$ do
   2.1 $b \leftarrow b \cdot z \bmod f(z)$.
   2.2 If $a_i = 1$ then $c \leftarrow c + b$.
3. Return(c).

Algorithmus 1: shift-and-add Multiplizierer in $\mathbb{F}_{2^m}$ aus Hankerson 2003, S.48

Das Ergebnispolynom c(z) hat höchstens den Grad $2m-2$ und muss anschließend reduziert werden. Im Gegensatz zum Referenzalgorithmus 1 wird darauf verzichtet in Schritt 2.1 eine Reduktion durchzuführen. Die Reduktion wird in einer getrennten Funktion mit dem Verfahren aus Abschnitt 3.4.3 durchgeführt.

Der Algorithmus ist besonders gut für den Einsatz auf Hardware geeignet, auf der das Verschieben einer Mehrwort-Zahl in einem Takt durchführbar ist. In Software sind allerdings durch die Anzahl der Shift-Operationen deutlich mehr Taktzyklen notwendig (Hankerson 2003, S.48).

Alternative Verfahren zur Multiplikation, die einen Geschwindigkeitsvorteil bringen, werden in Kapitel 4.2.2 beschrieben.

3.4.3. Reduktion

Wie zuvor erwähnt, weisen Ergebnisse aus der Multiplikation einen Grad von $2m-2$ auf und müssen auf eine Breite von m reduziert werden.

Es wird zum einen ein allgemeiner Algorithmus eingesetzt, der eine Reduktion um jedes Polynom durchführen kann. Andererseits werden für die in Abschnitt 3.2 erwähnten Kurven spezielle Algorithmen verwendet, die auf die Domain-Parameter der Kurve angepasst sind und eine schnellere Verarbeitung erlauben.

3.4.3.1. Allgemeine Variante

In der allgemeinen Variante wird zunächst eine Vorverarbeitung durchgeführt, die eine Nachschlagetabelle vorbereitet, die den Modulo-Wert um genauso viele Bits verschiebt, wie der Tabellenindex an dem der berechnete Wert gespeichert wird.

Im Anschuss wird das Eingangspolynom vom Bit $2m-2$ rückwärts durchlaufen. An jeder Stelle die ein Bit Wert 1 aufweist wird ein Wert aus der Tabelle abgerufen

und zum Ergebniswert addiert. Nach Ablauf des Algorithmus sind alle Stellen, die über die Bitposition m hinausgehen auf 0 gesetzt und das Ergebnis ist im Bereich 0 bis m. Folgendes Beispiel verdeutlicht die Reduzierung in $GF(2^9)$:

```
Beispiel für m = 9 f(z) = 00000010 00000011
c(z) = 00000000 00000001 00100000 00000011
d(z) = c(z) mod f(z)

c(z)              00000000 00000001 00100000 00000011
+ f(z) << 7       00000000 00000001 00000001 10000000
                  ----------------------------------
=                 00000000 00000000 00100001 10000011
+ f(z) << 5       00000000 00000000 00100000 00110000
                  ----------------------------------
=                 00000000 00000000 00000001 10110011
d(z) =                              00000001 10110011
```

Der Referenzalgorithmus 2 führt bereits eine Optimierung durch, in dem die Nachschlagetabelle nur für die Bitbreite eines Worts aufgestellt wird und durch geschickte Wortauswahl eine Addition mit Versatz ausführt.

INPUT: A binary polynomial $c(z)$ of degree at most $2m-2$.
OUTPUT: $c(z)$ mod $f(z)$.

1. *Precomputation.* Compute $u_k(z) = z^k r(z), 0 \le k \le W-1$.
2. For i from $2m-2$ downto m do
   2.1 If $c_i = 1$ then
   Let $j = \lfloor (i-m)/W \rfloor$ and $k = (i-m) - Wj$.
   Add $u_k(z)$ to $C\{j\}$.
3. Return$(C[t-1], \ldots, C[1], C[0])$.

Algorithmus 2: Modulare Reduktion aus Hankerson 2003, S.53

Die Vorberechnung der Nachschlagetabelle kann für jede Kurve einmalig erfolgen und muss nicht wiederholt werden.

3.4.3.2. Spezielle Variante für sect163k1

Die Reduktion um das irreduzible Polynom einer Kurve kann beschleunigt werden, da die Bitbreite und Wortbreite im Vorhinein bekannt ist. In der Softwarelösung benötigt sect163k1 einundzwanzig Wörter mit je 8 Bit. Es werden dann mit 168 verfügbaren Bits die 163 benötigten abgedeckt.

Das Verfahren optimiert die Laufzeit, in dem die Schleife ausgerollt wird und einige Zwischenrechnungen auf die Wort- und Polynombreite angepasst werden können. Eine formale Beschreibung ist nachfolgend in Algorithmus 3 zu finden.

1: INPUT: A binary polynomial $c(z)$ of degree at most 324
2: OUTPUT: $c(z)$ mod $f(z)$
3: **for** $i \leftarrow 41$ to 21 **do**
4: $\quad T \leftarrow C[i]$.
5: $\quad C[i-21] \leftarrow C[i-21] \oplus (T \ll 5)$.
6: $\quad C[i-20] \leftarrow C[i-20] \oplus (T \ll 4) \oplus (T \ll 3) \oplus T \oplus (T \gg 3)$.
7: $\quad C[i-19] \leftarrow C[i-19] \oplus (T \gg 4) \oplus (T \gg 5)$.
8: **end for**
9: $T \leftarrow C[20] \gg 3$.
10: $C[0] \leftarrow C[0] \oplus (T \ll 7) \oplus (T \ll 6) \oplus (T \ll 3) \oplus T$.
11: $C[1] \leftarrow C[1] \oplus (T \gg 1) \oplus (T \gg 2)$.
12: $C[20] \leftarrow C[20]$ & $0x07$.
13: Return $C[20], \ldots, C[2], C[1], C[0]$.

Algorithmus 3: Reduktion für sect163k1 aus Seog Chung Seo 2008, S.8

3.4.4. Division und Inversion

Wie in Abschnitt 2.2 eingeführt, wird die Division und somit auch die Inversion mit Hilfe der multiplikativen Inverse durchgeführt. Die Inversion wird als Division $\frac{b}{a}$ betrachtet, in der der Faktor $b=1$ gesetzt ist. In einem Binärfeld muss für eine Division von b durch a die Gleichung $a \cdot b = 1$ gelten. Der Referenzalgorithmus 4 ist an den Erweiterten Euklidischen Algorithmus für binäre Polynome angelehnt, verzichtet aber auf die Bestimmung der Ordnung zweier Zahlen in jedem Durchlauf.

```
Modular_Division _GF(2 m){
   A← x(t), B← M(t), U← y(t), V← 0;
   while ( A not equal B ) do {
         if ( A even )  then {
                  A <– A / t;
                  if  ( U even ) then U <– U / t else U <– (U+M) / t;
         } else if  ( B even ) then {
                  B <– B / t;
                  if  ( V even ) then V <– V / t else V <– (V+M) / t;
         } else if ( A > B ) then {
                  A <– (A+B) / t;  U <– U+V;
                  if  ( U even ) then U <– U / t else U <– (U+M) / t;
         } else {
                  B <– (A+B) / t; V <– U+V;
                  if  ( V even ) then V <– V / t  else V <– (V+ M) / t;
         }
   }
}
```

Algorithmus 4: Modulare Division nach Shantz 2001, S.4

3.5. Punktarithmetik

Die Arithmetik auf finiten Feldern besteht im Kern aus der Addition und Multiplikation, die auf binären Feldern leistungsoptimal zu berechnen ist. In der nächsten Stufe wird nun die Punktarithmetik eingeführt, bezogen auf elliptische Kurven unter Verwendung des Finiten Feld $\mathbb{F}_{2^m}$.

Die Rechenoperationen bestehen wieder im Kern aus der Addition und Multiplikation. Des Weiteren wird die Punktdopplung eingeführt, die vergleichbar mit einer Shift-Operation in der Arithmetik der finiten Felder ist.

Als Koordinatensystem wird das Affine Koordinatensystem beibehalten, die Operationen werden nicht in anderen Koordinaten ausgeführt. In Abschnitt 4.2.1 wird beschrieben, wie mit anderen Koordinatensystemen eine Beschleunigung der Algorithmen durchgeführt werden kann.

3.5.1. Punkt auf der Kurve

Die im Rahmen dieser Ausarbeitung erwähnten Operationen haben gemeinsam, dass im Ergebnis wieder ein Punkt steht, der Teil der Kurve sein muss. Daher wird zunächst eine Funktion eingeführt, die überprüfen kann, ob sich ein Punkt auf der Kurve befindet. Als Eingabe wird der Punkt übergeben, der getestet werden soll sowie die Domain-Parameter der Kurve.

In Abschnitt 2.3 wurde die Punktmenge für eine elliptische Kurve definiert und festgelegt, dass der Punkt im Unendlichen ebenso dazu gehört. Es wird also zunächst geprüft, ob der Eingabepunkt der Punkt im Unendlichen ist und in dem Fall 1 zurückgegeben.

Andernfalls wird die in der Software verwendete Kurvengleichung $y^2+xy=x^3+ax^2+b$ mit $a=1$ und $b=1$ mit den Koordinaten aus dem Eingabepunkt berechnet. Stimmen die Seiten der Gleichung überein, befindet sich der Punkt auf der Kurve.

3.5.2. Punktdopplung

Eine Kernoperation der Punktarithmetik ist die Dopplung eines Punktes, vergleichbar mit einer Links-Shift Operation um eine Stelle. Es handelt sich um den Sprung von P zu $2P$. Da die Operation der Multiplikation auf die Punktaddition und Punktdopplung aufbaut, kann nicht die Multiplikation verwendet werden um $2P$ auszurechnen.

Zur Berechnung des Ergebnisses $2P=(x_3, y_3)$ in affinen Koordinaten werden zwei Formeln angegeben, die mit der Arithmetik des Binärfeldes auskommt (Cohen 2005, S.291):

$$x_3=\lambda^2+\lambda+a \qquad y_3=x_1^2+\lambda x_3+x_3$$

$$\text{mit } \lambda=x_1+\frac{y_1}{x_1}$$

Durch die Division ergibt sich die Rahmenbedingung, dass x_1 nicht 0 sein darf. Tritt der Fall dennoch ein, wird der Punkt im Unendlichen zurückgegeben. Weiterhin gilt auf Grund der Identität $P+\infty=\infty+P=P \;\forall\; P\in E(F_{2^m})$

der Sonderfall $2\cdot P=P+P=\infty$ für $P=\infty$.

Graphisch gesehen wird die Verdopplung eines Punktes durch das Ziehen einer Tangente an der Kurve zum Punkt P eingeleitet. Die Tangente schneidet die

Kurve ein weiteres Mal, woraufhin der Punkt an der x-Achse gespiegelt wird. Der Zielpunkt hat den Wert $2P$. Ein Beispiel wird in Diagramm 3.1 gegeben.

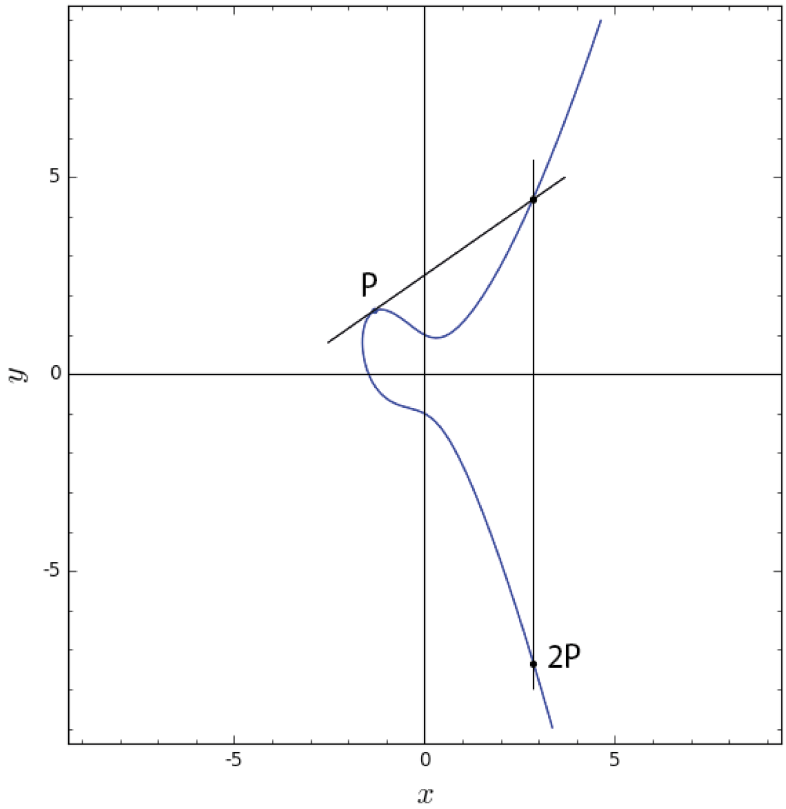

Diagramm 3.1: Punktverdopplung von P

3.5.3. Punktaddition

Hat man die Punktdopplung implementiert, kann darauf die Punktaddition aufbauen. Es handelt sich um die Addition zweier Punkte P und Q. Es sind zunächst einige Rahmenbedingungen zu beachten. Die Bereits in Abschnitt 3.5.2 erwähnte Identität führt bei der Addition zu den folgenden Bedingungen:

$$P+\infty=P \qquad \infty+Q=Q$$

Falls $P=Q$ wird stattdessen die Punktdopplung durchgeführt. Außerdem entsteht bei gleicher x-Koordinate aber unterschiedlicher y-Koordinate eine vertikale Linie, die nicht erneut den Graphen schneidet. Definitionsgemäß wird auch hier der Punkt im Unendlichen zurückgegeben.

Anschließend kann die Berechnung des Ergebnisses unter normalen Bedingungen erfolgen. Es werden dazu ähnliche Formeln wie bei der Punktdopplung verwendet:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \qquad y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

$$\text{mit } \lambda = \frac{y_1 + y_2}{x_1 + x_2}$$

Graphisch wird die Summe R der Punkte P und Q ebenso durch eine Linie eingeleitet, jedoch keine Tangente. Die beiden Punkte werden durch eine Linie verbunden, die eventuell erneut die Kurve schneidet. Durch Spiegelung an der x-Achse wird der Punkt $R = P + Q$ bestimmte.

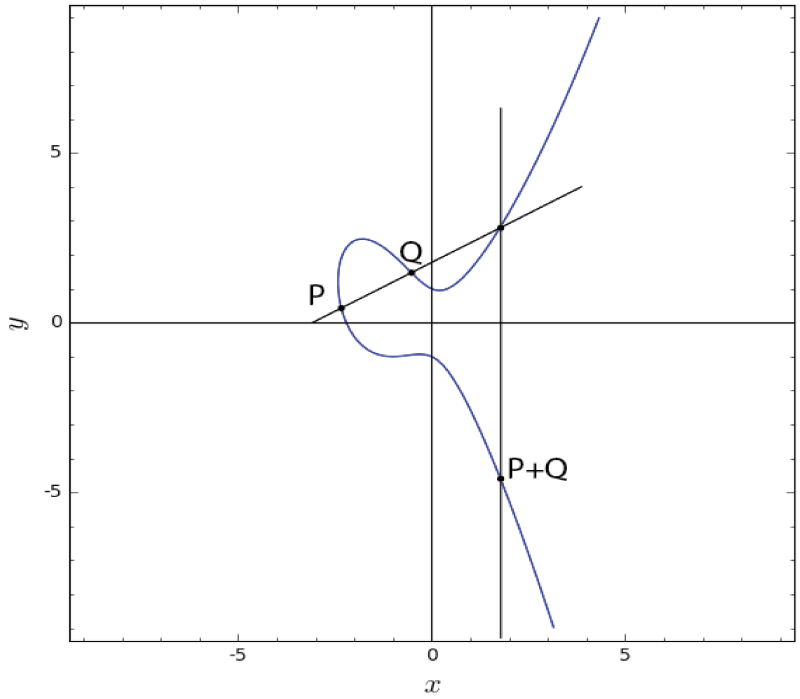

Diagramm 3.2: Punktaddition P + Q

3.5.4. Punktmultiplikation

Die Punktmultiplikation wird ebenso wie die Multiplikation in finiten Feldern über die Kombination der Verschiebeoperation und einer Addition durchgeführt. Die Verschiebeoperation entspricht in der Punktearithmetik der in Abschnitt 3.5.2 eingeführten Punktdopplung.

Es handelt sich um eine Multiplikation eines skalaren Wertes mit einem Punkt auf der Kurve. Der skalare Wert ist Element des finiten Feldes. Der Ergebnispunkt Q wird einführend auf unendlich gesetzt. Nun werden die Bits des skalaren Wertes durchlaufen. In jedem Schritt wird der Ergebniswert Q gedoppelt. Im Anschluss wird eine Punktaddition von P und Q durchgeführt, falls das Bit an der aktuellen Position im Skalar gerade 1 ist. Der Ablauf ist sehr nah an dem in Abschnitt 3.4.2 beschriebenen Verfahren und wird im Referenzalgorithmus 5 formalisiert.

Die Punktverdopplung und Punktaddition weist immer ein Ergebnispunkt auf, der sich auch auf der Kurve befindet. Dadurch kann sichergestellt werden, dass auch das Ergebnis der Multiplikation auf der Kurve liegt.

INPUT: $k = (k_{t-1}, \ldots, k_1, k_0)_2$, $P \in E(\mathbb{F}_q)$.
OUTPUT: kP.
1. $Q \leftarrow \infty$.
2. For i from $t-1$ downto 0 do
   2.1 $Q \leftarrow 2Q$.
   2.2 If $k_i = 1$ then $Q \leftarrow Q + P$.
3. Return(Q).

Algorithmus 5: Punktmultiplikation für Kurven mit Binärfeld, aus Hankerson 2003, S.97

Nun sind alle mathematischen Operationen eingeführt, die für das Kryptosystem mit elliptischen Kurven notwendig sind. Im nächsten Abschnitt können die für die digitale Signatur notwendigen Operationen implementiert werden.

3.6. ECDSA-Algorithmen

Der letzte Teil der Softwareimplementierung beruht auf der Umsetzung der ECDSA-Algorithmen. Im Vergleich zu den arithmetischen Funktionen sind die Funktionen zur Schlüsselgenerierung, Signatur und Verifizierung deutlich einfacher umzusetzen. Dennoch zeigt sich an dieser Stelle, ob alle bisherigen Operationen korrekt umgesetzt worden sind.

Der Domain-Parameter n gewinnt bei der Umsetzung der ECDSA-Funktionen eine besondere Wichtigkeit. Es handelt sich um die Ordnung des Generatorpunktes und wird als Modulo-Wert in allen ECDSA-Funktionen eingesetzt. Wird in diesem Abschnitt eine Reduktion durchgeführt, geschieht diese modulo dem Domain-Parameter n.

3.6.1. Schlüsselgenerierung

Die Schlüsselgenerierung erzeugt ein Schlüsselpaar aus geheimen und öffentlichen Schlüssel, wie sie in Abschnitt 2.1 eingeführt wurden. Die Erzeugung hinsichtlich elliptischer Kurven ist nach der korrekten Implementierung der Punktarithmetik nahezu trivial.

Es muss zunächst ein Private Key erzeugt werden, dazu wird ein zufälliges Element des finiten Feldes ausgesucht. Softwaretechnisch werden Zufallsdaten erzeugt und die Wörter der Zahl gefüllt. Im Anschluss wird mit der aus Abschnitt 3.4.3 bekannten Reduktion sichergestellt, dass sich die Zufallsdaten im Wertebereich befinden.

Im Anschluss muss lediglich eine Punktmultiplikation durchgeführt werden. Der geheime Schlüssel wird mit dem Generatorpunkt multipliziert, um den öffentlichen Schlüssel zu erhalten.

Die Rückrechnung ist wie in Abschnitt 2.3.3 besprochen nicht mit vertretbarem Aufwand zu erlangen.

Der öffentliche Schlüssel kann im Anschluss validiert werden. Zum einen darf der öffentliche Schlüssel nicht der Punkt im Unendlichen sein. Zum anderen müssen die Komponenten des Punktes gültige Elemente aus dem finiten Feld sein. Selbstverständlich muss der öffentliche Schlüssel auch ein Punkt auf der Kurve sein.

3.6.2. Hash-Verfahren

Das ECDSA-Verfahren ist nicht in der Lage, gesamte Texte ohne Umwege zu signieren. Die Anzahl der Bits, die signiert werden, ist begrenzt. Es wird daher zunächst ein Hashverfahren angewandt, dass die Nachrichtenlänge verkürzt.

Ein Hash-Verfahren bildet Daten beliebiger Länge auf eine feste Länge ab. Die Abbildung ist stabil, werden die gleichen Daten zweimal in die Hash-Funktion eingegeben entsteht das gleiche Ergebnis. Diese Eigenschaft wird ausgenutzt um die Nachricht auf eine feste länge zu kürzen, so dass die Daten signiert werden können.

Eine kryptographische Hash-Funktion hat weiterhin den Vorteil, dass die Abbildung sich nicht mit vertretbarem Aufwand umkehren lässt. Als kryptographisch sicher gelten zum aktuellen Zeitpunkt die SHA-2 Familie der Hashverfahren, zum Beispiel SHA-256 oder SHA-512.

Das Verfahren verwendet eine Reihe von Konstanten und bricht die Nachricht in mit 0 Werten aufgefüllten Blöcken herunter. In einer Schleife wird dann durch eine Reihe von Rotationen, sowie AND und XOR Funktionen, die Wortlänge reduziert.

In der Softwarelösung wurde das SHA-256 Verfahren eingesetzt. Da es sich lediglich um die Umsetzung eines überschaubaren standardisierten Verfahrens handelt und eine eigene Implementierung stark vergleichbar mit den existierenden wäre, wurde stattdessen eine vorgefertigte Variante mit offener Lizenz verwendet. Die gewählte Implementierung lässt sich ebenso leichtfertig in Hardware umsetzen, wie die übrige Softwarelösung.

3.6.3. Digitale Signatur

Zur Erzeugung der digitalen Signatur wird der geheime Schlüssel des Versenders benötigt, sowie der Hash-Wert der zu signierenden Nachricht.

Es wird zunächst ein einzigartiger Wert k für die Signatur ausgesucht. Es ist wichtig, dass der Wert nicht erneut für eine andere Nachricht verwendet wird, da ein Angreifer ansonsten den geheimen Schlüssel errechnen könnte. Der Wert k darf nicht größer als der Domain-Parameter n sein.

Als Nächstes wird eine Punktmultiplikation mit dem Wert k und dem Generatorpunkt der Kurve durchgeführt. Die x-Komponente wird erneut auf den Bereich des Domain-Parameters n reduziert und legt den Signaturwert r fest.

Der Signaturwert s wird nun mit folgender Formel aus dem einzigartigen Wert k, dem geheimen Schlüssel d, Hash-Wert $H(m)$ und Signaturwert r errechnet:

$$s = k^{-1} \cdot (H(m) + d * r)$$

Hat der Hash-Wert einen höheren Grad als die Zahlen im Kryptosystem, wird der Hash-Wert abgeschnitten. Es sollte daher im Kryptosystem ein ausreichend großes finites Feld verwendet werden. Es muss zu dem Sichergestellt werden, dass die Werte r und s nicht 0 sind. Die Signatur ist nun über die Kombination der Werte r und s definiert. Formal liegt der Umsetzung durch Referenzalgorithmus 6 zu Grunde.

INPUT: Domain parameters $D = (q, \text{FR}, S, a, b, P, n, h)$, private key d, message m.
OUTPUT: Signature (r, s).

1. Select $k \in_R [1, n-1]$.
2. Compute $kP = (x_1, y_1)$ and convert x_1 to an integer $\bar{x}_1$.
3. Compute $r = \bar{x}_1 \bmod n$. If $r = 0$ then go to step 1.
4. Compute $e = H(m)$.
5. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
6. Return(r, s).

Algorithmus 6: ECDSA Signaturerzeugung, aus Hankerson 2003, S.184

3.6.4. Verifizierung der Signatur

Die Verifikation einer Signatur erfordert nicht den geheimen Schlüssel, sondern kommt auf Grund der Ausführungen in Abschnitt 2.1 und 2.3.3 mit den öffentlichen Schlüssel aus. Weiterhin muss die zu verifizierende Signatur und der Hash-Wert der Nachricht bereitgestellt werden.

Nach einer Prüfung, ob r und s gültige Zahlen modulo des Domain-Parameters n sind, muss folgende Formel berechnet werden:

$$X = (H(m) \cdot w \bmod n) \cdot P + (r \cdot w \bmod n) \cdot Q \quad \text{mit } w = s^{-1} \bmod n$$

Der resultierende Punkt darf nicht der Punkt im Unendlichen sein. Anschließend wird die x-Koordinate des Punktes modulo n mit dem Signaturwert r verglichen. Stimmen die Werte überein, so ist die Signatur gültig. Die Grundlage stellt hier Referenzalgorithmus 7.

INPUT: Domain parameters $D = (q, \text{FR}, S, a, b, P, n, h)$, public key Q, message m, signature (r, s).
OUTPUT: Acceptance or rejection of the signature.

1. Verify that r and s are integers in the interval $[1, n-1]$. If any verification fails then return("Reject the signature").
2. Compute $e = H(m)$.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $X = u_1 P + u_2 Q$.
6. If $X = \infty$ then return("Reject the signature");
7. Convert the x-coordinate x_1 of X to an integer $\bar{x}_1$; compute $v = \bar{x}_1 \bmod n$.
8. If $v = r$ then return("Accept the signature");
   Else return("Reject the signature").

Algorithmus 7: ECDSA Signaturverifikation, aus Hankerson 2003, S.184

3.6.5. Korrektheit der Signierung

Die nachfolgende Gleichung aus Abschnitt 3.6.3 ist ausschlaggebend dafür, dass die signierte Nachricht tatsächlich vom Absender stammt:

$$s = k^{-1} \cdot (H(m) + d * r)$$

Durch Umstellung der Gleichung lässt sich mathematisch zeigen, dass der Algorithmus korrekt ist:

$$k \equiv s^{-1}(H(m) + dr) \equiv s^{-1}e + s^{-1}rd \equiv we + wrd \equiv u_1 + u_2 d \pmod{n}$$

Folglich gilt $X = u_1 P + u_2 Q = (u_1 + u_2 d) P = k P$ und es muss $v = r$ gelten, wie in Algorithmus 7 gefordert (Hankerson 2003, S.185).

3.7. Testfälle

Eine vollständige Besprechung der einzelnen Testfälle wäre für diese Arbeit nicht zielführend, daher wird an dieser Stelle nur bestimmte Aspekte der Testfälle angesprochen.

Es wurde auf eine möglichst große Abdeckung des Quellcodes geachtet. Die mathematischen Operationen hängen alle voneinander ab, besteht also in einer Funktion ein Fehlerfall kann es Auswirkungen auf das gesamte Projekt haben.

Die Testfälle wurden weitgehend mit der Testkurve aus Abschnitt 3.2 durchgeführt. Zur Überprüfung der mathematischen Operationen wurden zwei Ansätze verfolgt. Zum einen wurden manuelle Tests geschrieben, resultierend auf den Ergebnissen einer manuellen Rechnung auf Papier. Der Test überprüft mit zufällig ausgewählten Eingabedaten ein festes Resultat.

Zum anderen wurden Tests geschrieben, die auf Nachschlagetabellen basieren. Die Tabellen wurden mit der bereits bestehenden Software Rye[1] generiert und geben die Ergebnisse für die Addition, Multiplikation und Inversion auf einem finiten Feld $GF(2^9)$. Anschließend wird der Algorithmus aus der eigenen Softwarelösung mit den Werten überprüft.

Für Tests bei denen keine Nachschlagetabellen verfügbar waren, wurde der Algorithmus mit Hilfe von Rye nachprogrammiert und so die Ergebnisse überprüft.

1 http://molefrog.com/rye/ – „Rye is a JavaScript library that allows to work with finite fields“

Durch die Fremdsoftware ist keine vollständige Sicherheit gegeben, dass die Ergebnisse korrekt sind. Rye vermittelt allerdings einen sicheren Eindruck und es ist Vorteilhaft, wenn beide Softwareteile unabhängig auf die gleiche Lösung stoßen.

Es wurde weiterhin mittels Invarianten einige Zusätzliche Eigenschaften der Operationen überprüft. So konnten insgesamt alle mathematischen Operationen sicher getestet werden.

Defizite bestehen bei der Verifizierung der ECDSA-Funktionen. Die Eigenschaften eines öffentlichen Schlüssels lassen sich mit dem Verfahren aus Abschnitt 3.6.1 überprüfen. Auch lassen sich bestimmte Eigenschaften der Signatur überprüfen. Die Funktionsweise der Signierung und Verifizierung lassen sich jedoch ansonsten nicht getrennt voneinander überprüfen, so dass hier eine besondere Schwierigkeit auftrat zu einem Ergebnis zu kommen.

Um dem entgegenzuwirken wurde wie zuvor beschrieben viel Wert auf die Funktionstests der arithmetischen Funktionen gelegt. Es ist anzunehmen, dass diese Funktionen korrekt arbeiten.

4. Ausblick

Nachfolgend werden einige Vorschläge unterbreitet, wie eine Weiterführung dieser Ausarbeitung aussehen könnte.

Zunächst werden einige Vorteile einer Hardwareimplementierung, sowie Optimierungen durch Spezialisierung auf die Eigenschaften der Software- und Hardwarearchitekturen. Anschließend werden einige Algorithmen besprochen, die eine Geschwindigkeitsverbesserung versprechen und im Abschluss werden noch einige Sicherheitsüberlegungen getroffen.

4.1. Hardwareimplementierung

Mittels einer Hardwareimplementierung kann ein Vergleich zwischen der bisher erstellten Softwarelösung und der Umsetzung auf Hardware erfolgen. Wie zuvor erwähnt wurde bei der Umsetzung darauf geachtet, möglichst simpel umzusetzende Datenstrukturen und Programmabläufe zu verwenden.

Durch die Entwicklung einer Hardwarekomponente lässt sich im ersten Schritt ein Vergleich ziehen, welchen Geschwindigkeitsunterschied die Umsetzung in der Hardware erreicht. Als Nächstes kann die Softwarelösung stark auf den Einsatz auf dem Mikroprozessor optimiert werden.

Dazu kann als Erstes die Bitbreite erhöht werden. Bei der Nutzung von 64 bit Datenworten werden die benötigten Operationen deutlich verringert. Anstatt für eine 163 Bit Zahl 21 Datenwörter zu bearbeiten, sind lediglich 3 Datenwörter vonnöten. Zudem kann mittels Parallelisierung die Verarbeitung in einigen Algorithmen beschleunigt werden. Insbesondere bei der hier gewählten Spezialisierung bestehen zwischen den Datenwörtern weniger Abhängigkeiten, so dass z. B. bei der Addition in finiten Feldern alle Datenwörter gleichzeitig verarbeitet werden können.

Hardwareseitig ist es mit einem ASIC möglich Verarbeitungseinheiten zu erstellen, die für eine bestimmte Klasse von Kurven eine vektorbasierte Verarbeitung einzuführen. Anstatt Operationen auf getrennten Datenwörtern auszuführen, kann eine parallele Verarbeitung gesamter Abläufe angestrebt werden. Spezialisierte Hardwarelösungen erlauben es eine Bitbreite zu erlangen, die für konventionelle Mikroprozessorhardware unüblich ist. Mit Reconfigurable Computing wäre auch eine Anpassungsfähigkeit an verschiedene Bitbreiten denkbar.

4.2. Alternative Algorithmen

Es können sowohl im Falle der Software als auch der Hardware unterschiedliche Algorithmen verwendet werden, die sich für die jeweilige Architektur Vorteilhaft zeigen. Nachfolgend werden einige Alternativen zu den in Kapitel 2 und 3 aufgeführten Algorithmen vorgestellt und ihre Tauglichkeit für Hardware und Software besprochen.

Überlegungen zur Leistung der Algorithmen werden durch die Anzahl der Multiplikationen, Inversionen und Quadraturen ausgedrückt. Da es sich bei den Additionen um XOR-Operationen handelt sind die Auswirkungen auf die Leistung vernachlässigbar, sie werden daher nicht aufgeführt.

Die Ausdrucksweise für die Leistung eines Algorithmus wird z. B. mit I + 2M + S ausgedrückt, was bedeutet, dass die Laufzeit durch die Laufzeit einer Inversion, zwei Multiplikationen und einer Quadraturoperation bestimmt ist.

Es muss bei der Umsetzung auf Hardware die relativen Laufzeiten für Inversion, Multiplikation und Quadraturen beachtet werden, um zu bewerten welcher Algorithmus für den Einsatzzweck am besten geeignet ist.

4.2.1. Alternative Koordinatensysteme

Bei der Softwaretechnischen Umsetzung wurden affine Koordinaten verwendet, eben solche die durch das Paar $(x_{1,} y_1)$ beschrieben sind. Das Koordinatensystem findet bei den arithmetischen Punktoperationen seine Relevanz.

In affinen Koordinaten werden die Punktoperationen wie folgt berechnet:

Addition: $x_3=\lambda^2+\lambda+x_1+x_2+a \qquad y_3=\lambda(x_1+x_3)+x_3+y_1 \quad \text{mit } \lambda=\frac{y_1+y_2}{x_1+x_2}$

Dopplung: $x_3=\lambda^2+\lambda+a \qquad y_3=x_1^2+\lambda x_3+x_3 \qquad \text{mit } \lambda=x_1+\frac{y_1}{x_1}$

Beide Operationen sind in $I+2M+S$ auszuführen. Die Inversionsoperation ist in der Regel besonders kostenintensiv.

Zum Vergleich können andere Koordinatensysteme verwendet werden, zum Beispiel Projektive Koordinaten. Es handelt sich hier um ein dreidimensionales Koordinatensystem mit $P=(X_1:Y_1:Z_1)$, so dass $x_1=\frac{X_1}{Z_1}$ und $y_1=\frac{Y_1}{Z_1}$. Wendet

man die Transformation auf die Kurvengleichung an und vervielfacht die, erhält man:

$$y^2+xy=x^3+ax^2+b \equiv Y^2Z+XYZ=X^3+aX^2Z+bZ^3$$

Die Operationen sind nun wie folgt umzusetzen:

Addition:

$$X_3=BE \quad Y_3=C(AX_1+Y_1B)Z_2+(A+B)E \quad Z_3=B^2D$$
$$\text{mit } A=Y_1Z_2+Y_2Z_1 \quad B=X_1Z_2+X_2Z_1 \quad C=B^2 \quad D=Z_1Z_2 \quad E=(A^2+AB+aC)D+BC$$

Dopplung:

$$X_3=CE \quad Y_3=(B+C)E+A^2C \quad Z_3=CD$$
$$\text{mit } A=X_1^2 \quad B=A+Y_1Z_1 \quad C=X_1Z_1 \quad D=C^2 \quad E=(B^2+BC+aD)$$

Zwar sind hier die Kosten augenscheinlich höher, aber es wird keine Inversion benötigt. Die kosten für eine Addition beträgt 16M + 2S und eine Verdopplung kostet 8M + 4S. Weitere Optimierungen sind möglich, wenn die Eingabe eine affine Koordinate ist bei der Z = 1 gilt (Cohen 2005, S.292).

Die nachfolgende Tabelle vergleicht die Kosten in verschiedenen Koordinatensystemen. Weitere Reduzierungen sind ggf. mit Vorabkalkulationen unveränderlicher Werte möglich.

	Addition	Addition Affine	Dopplung
Affine Koordinaten	I + 2M + S	--	I + 2M + S
Projektive Koordinaten	16M + 2S	12M + 2S	8M + 4S
Jacobian Koordinaten	16M + 3S	11M + 3S	5M + 5S
López-Dahab Koordinaten	13M + 4S	9M + 5S	4M + 5S

4.2.2. Alternative Multiplikationsalgorithmen

Die in Abschnitt 3.4.2 und 3.5.4 erwähnten Multiplikationsalgorithmen sind sehr simpel gestaltet und verfolgen das shift-and-add Prinzip.

Eine Verbesserung stellt *Montgomery's Ladder* dar, in Kombination mit auf binäre Kurven spezialisierten Additions- und Dopplungsoperationen, in Projektiven Koordinaten. Für eine Addition sind dabei nur $4M+S$ Operationen nötig, für eine Dopplung $2M+3S$. Zusammenfassend sind hier nur $(6M+4S)(|n|_2-1)$

wobei n den skalaren Wert aus der Multiplikation darstellt. Die Verwendung von Montgomery's Ladder verbessert zwar nicht die Laufzeit, führt aber im Schleifendurchlauf in allen Pfaden die gleiche Anzahl von Operationen aus, so dass die Laufzeit keine Aussage über den Wert des privaten Schlüssels treffen kann (Cohen 2005, S.285, S.298).

4.3. Sicherheitsüberlegungen

Ein weiterer Ansatz das Projekt fortzuführen wäre die Verbesserung der Sicherheit der Algorithmen. Nachfolgend sollen einige Sicherheitsbedenken und Angriffspunkte aufgebracht werden, die in einem Folgeprojekt zu lösen wären.

4.3.1. Power Analysis

Bei der Umsetzung binärer Logik werden die logische 1 in der Regel durch einen hohen Spannungsverlauf dargestellt, während die logische 0 durch einen niedrigen Spannungsverlauf gekennzeichnet ist.

Unter genauer Betrachtung der Leistungsaufnahme der Hardware, die die Signierung ausführt, lassen sich Muster erkennen, die Rückschlüsse auf den privaten Schlüssel erlauben.

Lässt sich eine einzelne Geheimschlüsseloperation beobachten und ist die Art der Punktaddition und Punktverdopplung bekannt, kann die Reihenfolge der Operationen erkannt werden (Hankerson 2003, S239f).

In Diagramm 4.1 wird eine einfache Leistungsanalyse betrachtet, in der sich genau die Additionen (S) und Dopplungen (D) erkennen lassen.

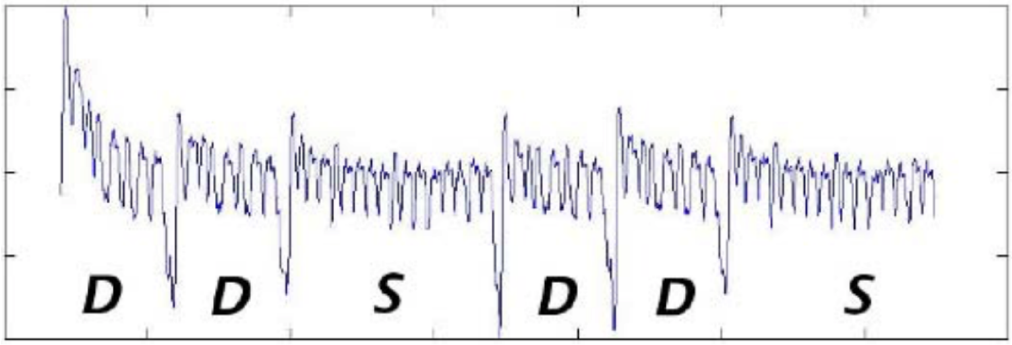

Diagramm 4.1: Ergebnisse einer Leistungsanalyse, aus Hankerson 2003, S.240

Als Gegenmaßnahme können zusätzliche Operationen eingefügt werden, damit die Leistungsaufnahme in allen Pfaden gleich bleibt.

4.3.2. Angriffe auf Domain-Parameter

Ein weiterer Ansatzpunkt zur Verbesserung ist eine sorgfältige Wahl der Domain-Parameter. Auf Grund der ressourcensparenden Einsatzmöglichkeit wurde für diese Arbeit eine Kurve mit kleiner Bitbreite verwendet. Die Testkurve ist zu dem deutlich kleiner.

Bei der Wahl der Domain-Parameter muss darauf geachtet werden, dass die Ordnung n des Generatorpunktes durch eine große Primzahl teilbar ist. Das Pohlig-Hellmann verfahren zeigt, dass bei Nichtbeachtung die Lösung des Elliptic Curve Discrete Logarithmic Problem mit verringertem Aufwand zu erreichen ist. Die Berechnung wird in Untergruppen aufgeteilt und durch Kongruenzen kann schlussendlich der private Schlüssel errechnet werden (Hankerson 2003, S.156f).

Weitere Verfahren sind Pollard‘s rho, Index-calculus und Isomorphismus Angriffe, deren genauere Beschreibung über den Umfang dieser Arbeit hinausgeht.

4.3.3. Fazit

Der Bereich der Sicherheitsüberlegungen ist weitreichend und verspricht spannende Projekte. Eine mögliche Fortführung wäre es eine einfache, unsichere Methode der Umsetzung in Hardware zu übersetzen und eine einfache oder differentiale Leistungsanalyse durchzuführen.

5. Endergebnis

Elliptische Kurven haben vielversprechende mathematische Eigenschaften, die den Einsatz in der Kryptographie begünstigen. Die Aussicht mit einer kleineren Schlüssellänge die gleiche Sicherheit wie ein RSA oder DSA Algorithmus zu erreichen ist positiv und unterstützt die Popularität des Algorithmus.

Insbesondere der Spezialfall der Koblitzkurven lässt sich auf Hardware leicht umsetzen und die im Ausblick erwähnten Optimierungen können die Laufzeit deutlich verringern.

Durch Einsatz von Reconfigurable Computing lassen sich weitere Optimierungsschritte leicht anwenden, ohne den Hardwareentwicklungszyklus von Vorne beginnen zu müssen.

Gleichwohl zeigte sich während der Softwareentwicklung, dass eine mathematisch Korrekte Umsetzung der Algorithmen viel Spezialwissen erfordert. Wird eine digitale Signatur von Nachrichten außerhalb des akademischen Umfelds benötigt, ist es ratsam auf eine vorhandene Bibliothek zurückzugreifen, die sich bereits erprobt hat.

Insgesamt ist das Themengebiet jedoch spannend und das Lösen der Fehlerfälle stellte eine Herausforderung dar, die den Ehrgeiz weckt.

A. Literaturverzeichnis

Carlet 2007	Carlet, C. and Sunar, B., 2007. Arithmetic of Finite Fields: First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings (Vol. 4547). Springer Science & Business Media.
Cohen 2005	Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K. and Vercauteren, F. eds., 2005. Handbook of elliptic and hyperelliptic curve cryptography. CRC press.
Compton 2002	Compton, K. and Hauck, S., 2002. Reconfigurable computing: a survey of systems and software. ACM Computing Surveys (csuR), 34(2), pp.171-210.
Hankerson 2003	Hankerson, D., Menezes, A.J. and Vanstone, S., 2006. Guide to elliptic curve cryptography. Springer Science & Business Media.
Krawczyk 1998	Krawczyk, H., 1998. Advances in Cryptology-CRYPTO'98: 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings (No. 1462). Springer Science & Business Media.
NIST 1999	NIST – Recommeded Elliptic Curves for Federal Government Use.
SECG 2010	Brown, D., 2010. SEC 2: Recommended Elliptic Curve Domain Parameters.
Seo Chung Seo 2008	Seo, S.C., Dong-Guk, H.A.N., Kim, H.C. and Seokhie, H.O.N.G., 2008. TinyECCK: Efficient elliptic curve cryptography implementation over GF (2 m) on 8-bit Micaz mote. *IEICE transactions on information and systems*, *91*(5), pp.1338-1347.
Seroussi 1998	Seroussi, G., 1998. Table of low-weight binary irreducible polynomials. Hewlett-Packard Laboratories.
Shantz 2001	Shantz, S.C., 2001. From Euclid's GCD to Montgomery multiplication to the great divide.