Zur Effizienz von Elliptische Kurven Kryptographie

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

von

Birgit Henhapl

aus Wien

Referenten: Prof. Dr. Johannes Buchmann
Prof. PhD. Nelson Stephens

Tag der Einreichung: 01. Oktober 2003
Tag der mündlichen Prüfung: 25. November 2003

D17

Für meine Großeltern

Gertrude Neuberger

und in memoriam

Dr. Paul Neuberger

Danksagung

Ich danke Prof. Dr. Johannes Buchmann dafür, dass er mir die Möglichkeit gab im Rahmen eines DFG Schwerpunktprojektes zu promovieren. Er hat mich von der Diplomarbeit an stets gefördert und gefordert und mich durch Projektarbeit auf eine Laufbahn in der Industrie vorbereitet. Vielen Dank.

Thanks to Prof. PhD. Nelson Stephens for accepting the task of the second referee and taking the trouble to read a German doctorial thesis.

Vielen Dank auch meinen Kollegen, besonders an:
Ulrike Meyer, für das angenehme Arbeitsklima, die gute Gemeinschaft und die vielen interessanten Gespräche.
Marita Skrobic, für ihre Hilfsbereitschaft und Freundlichkeit.
Markus Ernst, für einen Einblick in eine mir fremde Welt.
Ralf-Philipp Weinmann, für die große Unterstützung mit dem FlexiProvider.
Dr. Harald Baier, für die fachlichen Gespräche.
Dr. Tsuyoshi Takagi, für viele Anregungen.

Vielen Dank all den Korrekturlesern.

Großen Dank meinen Großeltern für all ihre Unterstützung.

Großen Dank meiner Familie, die mich stets in meiner Entwicklung gefördert hat und mir beibrachte, dass ich alles schaffen kann, was ich mir vornehme.

Und nicht zuletzt danke ich Markus.

Vielen Dank!

Zusammenfassung

In dieser Arbeit wird die Effizienz von Elliptische Kurven Kryptographie (ECC) über Primkörpern untersucht, speziell die Punktmultiplikation $n * P$. Dazu werden verschiedene Techniken kombiniert:

Die Multiplikation selbst kann mit Hilfe von Algorithmen beschleunigt werden, die zur schnellen Exponentiation eingesetzt werden. Dabei wird die Anzahl der benötigten Punktadditionen und -verdopplungen minimiert. Unter diesen Algorithmen eignen sich einige besonders für die Signaturerzeugung und wieder andere besonders für die Verifikation. Auch für die Schlüsselerzeugung und den Schlüsselaustausch können optimale Algorithmen gewählt werden.

Eine weitere Möglichkeit ist die Punktaddition und -verdopplung zu beschleunigen. Zu diesem Zweck werden unterschiedliche Koordinatensysteme, in dieser Arbeit die bekanntesten fünf, für jede dieser Operationen eingesetzt. Man spricht dann von gemischten Koordinaten.

Die Kombination dieser beiden Techniken führt zu einer Optimierung der Punktmultiplikation und ist Schwerpunkt dieser Arbeit. Für jeden einzelnen Algorithmus gilt es, die beste Koordinatenkombination in Abhängigkeit der Plattform zu wählen. In der Arbeit werden Gleichungen angegeben, mit die Auswahl durchgeführt werden kann. Pro Algorithmus gibt es Tausende Kombinationen. Daher wurde ein Programm implementiert, das dieKoordinatenwahl berechnet. Ergebnisse für eine Referenzplattform schließen die theoretischen Untersuchungen dieser Arbeit ab.

Im Rahmen dieser Arbeit wurden innerhalb des FlexiProviders (www.flexiprovider.de), einer Bibliothek für kryptographische Algorithmen, alle in dieser Arbeit analysierten Algorithmen zur Punktmultiplikation inklusive der gemischten Koordinaten implementiert. Dabei wird auf Basis der theoretischen Berechnungen des ersten Teils dynamisch in Abhängigkeit des Skalars n die Wahl der verschiedenen Systeme getroffen.

Abstract

In this thesis we study the efficiency of elliptic curve cryptography (ECC) over prime fields, especially the point multiplication $n * P$. There are two ways that lead to a faster point multiplication:

The first one is to use algorithms for fast exponentiation. By this the number of point additions and point doublings are reduced. Some of these algorithms are particularly suited for signing, others for signature verification. For key generation and exchange again other algorithms qualify.

The second way is to optimize the point addition and doubling itself. This can be done by using different coordinate systems for each of these two operations. This technique is called mixed coordinates. In this work the five best known systems are analysed.

We study the combination of these two techniques: For each of the best known algorithms for point multiplication we give equations to compute the best coordinate combination. Since there are thousands of possible combinations we provide a program, which does this computations dependent on the platform. Results of this program for one platform are given.

We also provide an implementation of each of the analysed multiplication algorithms with the mixed coordinates. It is based on the theoretical results of this work and supports dynamical coordinate choice dependent on the scalar n. This implementation is embedded in the FlexiProvider (www.flexiprovider.de), a library for cryptographic algorithms.

Erklärung[1]

Hiermit erkläre ich, dass ich die vorliegende Arbeit - abgesehen von den in ihr ausdrücklich genannten Hilfen - selbständig verfasst habe.

Wissenschaftlicher Werdegang der Verfasserin in Kurzfassung[2]

Okt. 1991 - Okt. 1999	Studium der Mathematik mit Schwerpunkt Informatik an der Technischen Universität Darmstadt
29. Oktober 1999	Diplomabschluss (Dipl.-Math.)
Nov. 1999 - Dez. 2003	Wissenschaftliche Mitarbeiterin am Fachgebiet Theoretische Informatik, Fachbereich Informatik, Technische Universität Darmstadt

[1] gemäß §9 Abs. 1 der Promotionsordnung der TU Darmstadt
[2] gemäß §20 Abs. 3 der Promotionsordnung der TU Darmstadt

Notationen

Domain Parameter:

a, b	Parameter einer elliptischen Kurve E
$\#E$	Anzahl der Punkte auf der elliptischen Kurve E
G	Kurvenpunkt mit Ordnung r
r	Primteiler von $\#E$ und Ordnung von G
k	$\#E/r$, der „Kofaktor“
s, u	EC private Schlüssel (*private keys*), ganze Zahlen
W, V	EC öffentliche Schlüssel (*public keys*), Kurvenpunkte
(s, W), (u, V)	EC Schlüsselpaare
f	Hashwert einer Nachricht M
$\mathcal{A}$	Affines Koordinatensystem
$\mathcal{P}$	Projektives Koordinatensystem
$\mathcal{J}$	Jacobisches Koordinatensystem
$\mathcal{J}^{\mathcal{C}}$	Chudnowsky-Jacobisches Koordinatensystem
$\mathcal{J}^{\mathcal{M}}$	Modifiziert Jacobisches Koordinatensystem
E	elliptische Kurve
$E(\mathbb{F}_q)$	elliptische Kurve über dem Körper $\mathbb{F}_q$
$\mathcal{O}$	Punkt im Unendlichen auf einer elliptischen Kurve E
P, Q, R	Punkte auf einer Kurve $E(\mathbb{F}_q)$
A	Addition im Körper $\mathbb{F}_p$
M	Multiplikation im Körper $\mathbb{F}_p$
S	Quadrierung im Körper $\mathbb{F}_p$
I	Invertierung im Körper $\mathbb{F}_p$

ADD	Punktaddition
DBL	Punktverdopplung
k	Exponent bei Punktmultiplikation der Form $k \cdot P$
k_1, k_2	Exponenten bei Punktmultiplikation der Form $k_1 \cdot P_1 + k_2 \cdot P_2$
n, n_1, n_2	Bitlänge des oder der Exponenten bei einer Punktmultiplikation
w, w_1, w_2	Fenstergröße bei Exponentiationsalgorithmen (*window size*)
h, v	Parameter bei der LimLee Exponentiationsmethode
ECC	Elliptische Kurven Kryptographie
DLP	Diskreter-Logarithmus-Problem
ECDLP	Diskreter-Logarithmus-Problem für elliptische Kurven
uvP	unter Verwendung vorberechneter Punkte
ovP	ohne Verwendung vorberechneter Punkte

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Im Zeitalter der elektronischen Datenverarbeitung, dem E-Commerce und dem elektronischem Datenverkehr wächst die Notwendigkeit zu mehr Sicherheit auch in diesen Bereichen. Denial-Of-Service-Attacken und Abhörangriffe tauchen immer häufiger in den Schlagzeilen auf und fast jeder Email-User wird mit sogenannten *Spams* überschwemmt, Nachrichten und Werbungen, die niemand interessieren, die manchmal Viren-behaftet und gefährlich sind und die zumeist unter falschem Absender versendet wurden.

So vielfältig die Attacken sind, so vielfältig sind die Maßnahmen, die zur Abwehr eingesetzt werden müssen. Die Kryptographie übernimmt eine wichtige Rolle bei diesen Maßnahmen. So wird Kryptographie im Standardwerk *Handbook of Applied Cryptography* [MvV97] wie folgt definiert: „*Cryptography* is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.“ Kryptographie soll also Techniken bereitstellen, die Vertraulichkeit, Integrität von Daten, Authentizität von Datennutzern und Originalität von Daten ermöglichen.

Elliptische Kurven in der Praxis

Elliptische Kurven Kryptographie (ECC) wurde erstmals von Koblitz [Kob87] und Miller [Mil86] vorgeschlagen. Sie gehört zur Public Key Kryptographie[1]. Vor Ko-

[1] D.h., zu einem privaten Schlüssel existiert ein öffentlicher Schlüssel, der vom Eigentümer bekannt gegeben wird, wogegen der private Schlüssel geheim bleibt. Man spricht von einem Schlüsselpaar. Der Kommunikationspartner verwendet den öffentlichen Schlüssel.

blitz und Miller lag das Interesse für elliptische Kurven nur in der Mathemtik. Doch längst ist - durch das Interesse an ihnen in der Kryptographie - die Entwicklung von kryptographischen Primitiven und Protokollen auf Basis elliptischer Kurven vorangeschritten.

Den Sprung in die Praxis hat die Elliptische Kurven Kryptographie ebenfalls erfolgreich geschafft: So wurde beispielsweise in den letzten Jahren in Österreich die *Bürgercard* eingeführt, eine Sozialversicherungskarte, die auch als Chipkarte (PSE) genutzt werden kann. Diese ist bereits ECC-fähig.

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) unterstützt ebenfalls die Weiterentwicklung elliptischer Kurven für kryptographische Zwecke. Die Elliptische Kurven Kryptographie hat den Vorteil, dass sie mit nur 160 Bit Schlüssellänge die gleiche Sicherheit bietet wie beispielsweise RSA mit 1024 Bit Schlüssellänge [fSidI02]. Das BSI empfahl schon mehrmals, RSA-2048 zu verwenden, musste diese Empfehlung jedoch immer wieder zurück ziehen, da die Wirtschaft auf diesen Wechsel noch nicht vorbereitet ist. Auch sind 2048 Bit RSA-Schlüssel für Chipkarten zu groß, RSA wird zu langsam. Hier bietet ECC kleinere Schlüssellängen und daher auch kürzere Laufzeiten.

Die Schnelligkeit der Algorithmen ist neben der Sicherheit der wichtigste Aspekt in der Kryptographie. Die Effizienz ist der Grund, weswegen in vielen Bereichen symmetrische Verfahren (Secret-Key Kryptographie[2]) eingesetzt werden: Der Vorteil ihres hohen Durchsatzes wiegt den Nachteil der komplizierten Schlüsselvergabe auf. Die Public-Key Kryptographie muss sich daher sehr anstrengen, schnelle Verfahren zu bieten, um den Secret-Key Verfahren eine echte Konkurenz zu sein. Und ganz kann auf die Public-Key Kryptographie auch nicht verzichtet werden: Spätestens bei Vertragsabschlüssen und Beglaubigungen werden digitale Signaturen benötigt, die sich nur mit Public-Key Kryptographie in vergleichbarer Weise zu konventionellen Unterschriften nachbilden lassen.

Inhalt dieser Arbeit

Diese Arbeit untersucht, welche Algorithmen und Techniken am besten für die Elliptische Kurven Kryptographie eingesetzt werden, um bestmögliche Effizienz zu erreichen. Es wird eine Krypto-Bibliothek, ein Java-Provider, für Elliptische Kurven Kryptographie über endlichen Primkörpern bereitgestellt, die auf allen Plattformen einfach zu bedienen und in Applikationen zu integrieren ist und alle stan-

[2] D.h., es existiert ein gemeinsamer Schlüssel zwischen zwei Kommunikationspartnern, der gegenüber Dritten geheim bleibt.

dardisierten Kryptoverfahren bietet. Sie ist um andere Kryptoverfahren oder die EC-Arithmetik über anderen Körpern erweiterbar, ohne in den ganzen Provider eingreifen zu müssen. Die untersuchten und optimierten Techniken wurden in den Provider integriert, nach Veröffentlichung dieser Arbeit ist auch dieser Teil Open Source und für die wissenschaftliche Gemeinschaft verfügbar.

Der Provider, sein Name ist *FlexiECProvider*, ist als Teil eines großen Providers, dem FlexiProvider [FG03], entstanden. Er bildet den Rahmen für diese Arbeit, deren Schwerpunkt die Optimierung der Punktmultiplikation ist, die in allen kryptographischen Anwendungen benötigt wird. Dies wird nachfolgend genauer erläutert.

Grundkenntnisse

Eine elliptische Kurve $E(\mathbb{F})$ über einem Körper $\mathbb{F}$ ist kurz gesagt eine kubische Gleichung mit Parametern und Lösungsmenge in $\mathbb{F}^2$. In der Kryptographie werden die elliptischen Kurven über endlichen Körpern definiert. In dieser Arbeit werden nur jene über Primkörpern $\mathbb{F}_p$ betrachtet, deren Gleichung wie folgt aussieht:

$$E(\mathbb{F}_p): \quad y^2 = x^3 + ax + b, \qquad \text{mit } a,\ b,\ x \text{ und } y \in \mathbb{F}_p.$$

Die Elemente der Lösungsmenge (x, y) werden Punkte auf $E(\mathbb{F}_p)$ genannt. Zusammen mit einem weiteren Punkt als neutralem Element, dem *Punkt im Unendlichen* $\mathcal{O}$, und einer inneren Abbildung $+$, der Punktaddition, bildet die Menge der Punkte eine abelsche Gruppe (Punktegruppe). Diese mathematische Struktur ist Grundlage für die Elliptische Kurven Kryptographie.

Die Sicherheit der standardisierten digitalen Signaturverfahren *ECDSA* und *ECNR*, das heißt DSA und Nyberg-Rueppel für elliptische Kurven, und des standardisierten Schlüsselaustauschverfahrens *ECDH*, das heißt Diffie-Hellman für elliptische Kurven, beruht auf der Schwierigkeit des Lösens des Diskreter Logarithmus Problems für elliptische Kurven (ECDLP): Das Berechnen des diskreten Logarithmus in der Punktegruppe einer elliptischen Kurve $E(\mathbb{F}_p)$ über einem endlichen Primkörper, das heißt gegeben zwei Punkte P und R auf einer elliptischen Kurve $E(\mathbb{F}_p)$ mit $R = k \cdot P$ mit $k \in \mathbb{Z}$, finde k, den *diskreten Logarithmus* von R zur Basis P.

Die Operation $k \cdot P$ wird *Punktmultiplikation* genannt und ist das k-malige Aufaddieren von P auf sich selbst. Diese Multiplikation ist einfach auszuführen. Schwierig ist es dagegen zu einem bekannten R und P den diskreten Logarithmus k zu finden.

Für ECC bestehen die Schlüsselpaare $(s,\ W)$ aus einer ganzen Zahl s und einem Punkt W auf einer elliptischen Kurve $E(\mathbb{F}_p)$. Der Punkt W ist öffentlicher Schlüssel und wird mit $W = s \cdot G$ berechnet. G ist *Basispunkt* auf $E(\mathbb{F}_p)$ und gehört zu einem öffentlichen Satz *EC Parameter*, welche den zu Grunde liegenden Körper $\mathbb{F}_p$, dessen Ordnung p, die Kurve E und den Basispunkt G mit seiner Ordnung r definieren. Zu jedem Satz EC Parameter gibt es viele verschiedene Schlüsselpaare.

Mit *Schlüssellänge* bei ECC-Verfahren wird im Allgemeinen die Größe oder Bitlänge der Ordnung r bezeichnet. Um sicherzugehen, dass das ECDLP nicht gebrochen werden kann, werden weitere Anforderungen an die EC Parameter gestellt, die hier nicht näher erläutert werden (siehe zum Beispiel in [fSidI02]).

Ebenso wie die Schlüsselerzeugung $W = s \cdot G$ benötigen die Signaturerzeugung und der Schlüsselaustausch je eine Punktmultiplikation, die Verifikation sogar zwei. Diese Operationen sind innerhalb der einzelnen Verfahren die aufwendigsten. Daher ist die Punktmultiplikation der Hebel, an dem man ansetzen muss, um ECC-Verfahren zu beschleunigen und so auch den Effizienzanforderungen bei Echtzeitanwendungen zu genügen.

Anforderungen und Ergebnisse

Beim Versuch einer Definition dieser Anforderungen an ECC stellt man fest, dass sie sehr stark vom Kontext, dem Anwendungsszenario, abhängen:

Bei einer Email-Anwendung genügt es vollkommen, wenn die Signaturerzeugung und -verifikation jeweils nicht mehr als 100 ms benötigen. Denn dies ist die Zeitspanne, die gerade unterhalb des menschlichen Wahrnehmungsvermögens liegt (siehe zum Beispiel [SH02, Folie 28] oder [Nie94, Kapitel 5.5]).

Betrachtet man jedoch die Auslastung von Bankservern für Privatkunden, sieht man, dass diese Forderung gänzlich unzureichend ist: So erhält die Dresdner Bank [Küh03] vormittags bis zu 200 Anfragen pro Sekunde, dass entspricht 5 ms pro Anfrage.

Eine andere, nur leicht weniger fordernde Anwendung ist das Anbringen von digitalen Signaturen mit Hilfe von Barcodes auf Briefumschlägen, beziehungsweise das Verifizieren dieser Signaturen. Dabei druckt die Post diese Postwertzeichen auf die Umschläge, die Kunden erwerben können. Die Sendungen können dann ohne zusätzlichen Aufwand des Kaufens von Briefmarken und Frankieren versendet werden. Auf der Poststelle wird jede Signatur auf dem Umschlag verifiziert. So

wird sichergestellt, dass die Gebühr für diesen Brief tatsächlich bezahlt ist. Es wird in diesem Szenario davon ausgegangen, dass etwa 4 Millionen Umschläge pro Tag die Maschinen zur Signaturerzeugung und -verifikation durchlaufen. Das bedeutet, dass für jede dieser Operationen höchstens 22 ms zur Verfügung stehen.

Ein Szenario in die entgegengesetzte Richtung ist eine gesicherte, drahtlose Verbindung zwischen einem mobilen Gerät wie einem Handy, PDA oder Pocket PC zu einem Online-Server: Die Toleranzgrenze ist erst bei mehr als einer Sekunde Wartezeit überschritten. Das liegt daran, dass die Benutzer diese Zeit noch tolerieren, da die Geräte generell noch nicht sehr leistungsstark sind, was zum Teil am Betriebssystem und den darauf laufenden Programmiersprachen liegt, die für diese speziellen Anwendungen noch nicht optimiert wurden. Dies wird bestätigt. Wir werden auch zeigen, dass der Provider für das erste Szenario bei weitem schnell genug ist, dass er die Forderungen des zweiten Szenarios ebenfalls befriedigen kann und dass und unter welchen Bedingungen auch den Anforderungen eines Bankservers genüge getan wird.

KryptoProzessor zur Punktmultiplikation

Zusätzlich wird ein FPGA-basierter Krypto-Prozessor zur Punktmultiplikation vorgestellt[3]. Er entstand in Zusammenarbeit mit Markus Ernst aus dem *Fachgebiet Integrierte Systeme und Schaltungen* für das Bundesamt für Sicherheit in der Informationstechnik. Der Beitrag aus Sicht dieser Arbeit war die Auswahl der Algorithmen, die Planung der Implementierung sowie die Integrierung des Prozessors in den Provider.

Die Aufgabe war, einen ersten Prototypen herzustellen, dessen Punktmultiplikation über endlichen Primkörpern in verschiedenen ECC-Anwendungen zum Einsatz kommen kann. Dabei sollte er über die Bitbreiten 160 Bit bis zu 512 Bit skalierbar sein.

Die erreichten Zeiten können sich sehen lassen: Eine Punktmultiplikation benötigt bei 160, 197 und 272 Bit jeweils nur 1.72 ms, 2.66 ms, beziehungsweise 4.53 ms. Eine Signaturerzeugung dauert für die gleichen Bitlängen mit Hilfe des Kryptoprozessors 1.97 ms, 2.97 ms, beziehungsweise 4.95 ms. Eine Verifikation ist - ebenfalls für die gleichen Bitlängen - in 4.2 ms, 6.27 ms beziehungsweise 10.24 ms zu schaffen.

[3]Bei der Karte handelt es sich um eine Alpha Data ADM-XPL PCM-Karte mit *Virtex II Pro* FPGA-Technologie. Sie ist mit einem XC2VP20 FPGA Baustein bestückt.

Vorangegangene Arbeiten

Zur Durchführung einer Punktmultiplikation können die meisten Exponentiationsverfahren [MvV97, MOC97, LL94] für eine multiplikative Gruppe $\mathbb{G}$ verwendet werden. Dabei entspricht eine Multiplikation in $\mathbb{G}$ einer Addition von zwei Punkten auf $E(\mathbb{F}_p)$ und eine Quadrierung in $\mathbb{G}$ einer Punktverdopplung auf $E(\mathbb{F}_p)$. Diese Algorithmen können auch für die Signaturverifikation verwendet werden, welche zwei Punktmultiplikationen und eine Punktaddition benötigt: $P = h_1 \cdot G + h_2 \cdot W$. Es gibt darüber hinaus aber auch Verfahren, die die drei einzelnen Operation in einer einzigen Methode abarbeiten und dabei an Punktoperationen sparen. Sie sind zum Beispiel in [MvV97], [Str64], [YLL94] und [Moe01] zu finden.

Jede Punktmultiplikation besteht aus einer Aneinanderreihung von Punktadditionen und -verdopplungen, die wiederum aus einer Anzahl von Körperoperationen bestehen. Die Anzahl der Körperoperationen pro Punktoperation hängt von dem Koordinatensystem ab, über dem die Kurven-Gleichung definiert ist. Das bekannteste ist das Affine Koordinatensystem. Dabei besteht ein Punkt aus den zwei Koordinaten x und y. Der Nachteil dieses Systems ist, dass jede Punktaddition und -verdopplung je eine Invertierung modulo der Primzahl p benötigt. Dies ist eine sehr teure Körperoperation.

Daher ist es in vielen Fällen günstiger, das Projektive oder das Jacobische Koordinatensystem zu verwenden, bei denen ein Punkt jeweils aus drei Koordinaten X, Y und Z besteht. Hierbei können die Invertierungen auf Kosten von zusätzlichen Körpermultiplikationen eingespart werden. Kryptographische Anwendungen verlangen zwar affine Punkte [IEE, Sta00a, X9.99], es ist aber möglich, die projektiven oder die jacobischen Punkte nach der aufwendigen Punktmultiplikation ins Affine System zu überführen. Dies benötigt neben einigen Körpermultiplikationen nur eine Invertierung modulo p. So wird statt je einer Invertierung pro Punktaddition und -verdopplung nur eine einzige am Ende jeder Punktmultiplikation ausgeführt.

In [DVC87] wird eine erweiterte Art des Jacobischen Systems vorgeschlagen, das Chudnowsky Jacobische, welches besonders effizient für Additionen ist, nicht jedoch für Verdopplungen. Dagegen ist das Modifiziert Jacobische System, das in [CMO98] vorgestellt wird, für Verdopplungen gut geeignet, nicht jedoch für Additionen. In [CMO98] wird anhand eines Algorithmus die Möglichkeit untersucht, die einzelnen Koordinatensysteme so für die Punktmultiplikation einzusetzen, dass ihre jeweiligen Vorteile zu Buche schlagen, nicht jedoch ihre Nachteile. Wir setzen in dieser Arbeit diesen Ansatz systematisch fort und entwickeln ihn weiter.

Eine Kombination verschiedener Koordinatensysteme, die zu einer Punktmultiplikation in einem bestimmten Algorithmus eingesetzt wird, wird in dieser Arbeit *Ausprägung* genannt. Durch verschiedene Ausprägungen kann die Laufzeit einer Punktmultiplikation erheblich beeinflusst werden.

Weitere Möglichkeiten zur Laufzeitoptimierung liegen in den Algorithmen zur Punktmultiplikation: Durch sogenannte *Window-Methoden* kann die Anzahl der Punktadditionen pro Punktmultiplikation reduziert werden. Dabei werden zu Beginn des Algorithmus Vielfache eines Punktes berechnet und gespeichert. Diese bereits berechneten Punkte können während der Hauptrechnung wiederverwendet werden. Die Anzahl der vorberechneten Punkte wird bei einigen dieser Algorithmen durch die *Fenstergröße* w bestimmt[4]. Je größer w ist, desto kleiner ist die Anzahl der benötigten Punktadditionen in der Hauptrechnung.

Eine andere Methode reduziert die Anzahl beider Punktoperationen, Punktaddition und -verdopplung, doch nur unter sehr hohem Aufwand in der Vorberechnung. Dies ist kein Nachteil, wenn es möglich ist, die Vorberechnung in eine zeitunkritische Phase zu verlagern, zum Beispiel bei der Signaturerzeugung: Sie benötigt die Berechnung von $u \cdot G$. Dabei ist u eine zufällig gewählte ganze Zahl und G der Basispunkt. Dieser wird für jede neue Signatur wiederverwendet und ist von der ersten Signaturerzeugung an im Voraus bekannt, bis die EC Parameter neu gewählt werden. Daher können Vielfache von G einmal vorberechnet werden, um sie bei der nächsten Signaturerzeugung erneut einzusetzen. Dies gilt natürlich für jeden Algorithmus zur Punktmultiplikation, der die Verwendung einer solchen Vorberechnung wie beschrieben beinhaltet.

Es stehen also viele Techniken zur Verfügung, die Laufzeit einer Punktmultiplikation und somit die von kryptographischen Elliptische Kurven Verfahren zu beschleunigen. Zur besseren Übersicht werden sie hier nochmals aufgezählt:

- Es stehen fünf verschiedene Koordinatensysteme zur Verfügung, deren jeweilige Komplexitäten sich pro Punktaddition und -verdopplung unterscheiden.
- Diese Koordinatensysteme können auch gemischt verwendet werden und so optimal in den jeweiligen Punktmultiplikationen zum Einsatz kommen.
- Es gibt verschiedene Algorithmen zur Punktmultiplikation. Sie teilen sich insgesamt in vier Gruppen: Algorithmen, mit denen einfache Punktmultiplikationen der Form $k \cdot P$ berechnet werden können, und Algorithmen, die

[4] Einzelne dieser Algorithmen besitzen auch mehr als eine Fenstergröße w.

mehrfache Punktmultiplikationen wie $k_1 \cdot P_1 + k_2 \cdot P_2$ ausführen. Beide Gruppen teilen sich weiter in die Algorithmen, bei denen die Vorberechnung von Punkten aus der Gesamtberechnung herausgezogen werden kann, und in die, bei denen dies nicht möglich ist.

- Die Komplexitäten dieser Algorithmen hängt von der jeweiligen Fenstergröße w ab, die in Abhängigkeit der Bitlänge optimal gewählt werden muss.
- Unterschiedliche Kombinationen dieser Variablen führen zu unterschiedlichen Komplexitäten und somit auch zu unterschiedlichen Laufzeiten.

Fokus der Arbeit

Das Hauptziel dieser Arbeit besteht darin, auf die beste Weise die vorgestellten Techniken miteinander zu vereinen, um die schnellstmögliche Punktmultiplikation zu erreichen. Ein Weg, das schnellste Verfahren mit der optimalen Koordinatenbesetzung (Ausprägung) zu bestimmen, könnte das Implementieren aller Verfahren mit allen möglichen Kombinationen sein. Dann kann die Laufzeit der einzelnen Algorithmen mit ihrer Ausprägung gemessen werden.

Doch durch die große Anzahl der möglichen Ausprägungen pro Algorithmus trifft man gleich auf das erste Problem: Während mit Hilfe von Parametrisierung die Algorithmen mit allen Ausprägungen noch gut zu implementieren sind, scheitert dieser Weg spätestens an der Zeitdauer, die für alle Experimente aufzuwenden ist.

Der rein experimentelle Weg ist also inpraktikabel. Darüber hinaus müsste der gleiche Aufwand für jede Plattform neu investiert werden, da die optimale Ausprägung pro Algorithmus von dieser abhängen kann.

Theoretische Analyse und Strategie

Wir stellen in dieser Arbeit eine andere Strategie vor, mit deren Hilfe berechnet werden kann, welches das optimale Verfahren in Bezug auf die Anwendung ist und welche Koordinaten am besten eingesetzt werden. Dabei wird die Plattform, auf der die Punktmultiplikation eingesetzt werden soll, in Betracht gezogen. Zum Einsatz kommen Tabellen, die die Anzahl der Körperoperationen pro Punktaddition und -verdopplung auflisten. Diese wurden in [CMO98] angefangen und in dieser Arbeit vervollständigt. Ebenso wurden in dieser Arbeit die Laufzeitkonstanten

der Komplexitätsformeln der einzelnen Punktmultiplikationsalgorithmen berechnet, die ebenfalls in die Berechnung des optimalen Verfahrens mit der optimalen Ausprägung einfließen.

Die Strategie teilt sich in folgende Schritte:

1. Messen der jeweiligen Laufzeit der einzelnen Körperoperationen Addition, Multiplikation, Quadrierung und Invertierung auf der festgelegten Plattform im Körper $\mathbb{F}_p$.
2. Unter Verwendung der Zeiten aus Punkt 1. und den Komplexitäten der jeweiligen Punktoperation kann hochgerechnet werden, wie lange jede einzelne Punktaddition und -verdopplung in jedem der Koordinatensysteme benötigt.
3. Unter Verwendung der Zeiten aus Punkt 2. kann mittels der Komplexitätsformeln der einzelnen Punktmultiplikationsalgorithmen hochgerechnet werden, wie lange jeder einzelne der Algorithmen in Abhängigkeit einer Fenstergröße w benötigt.
4. Diese hochgerechneten Laufzeiten werden für jede Ausprägung pro Punktmultiplikationsmethode in aufsteigender Folge geordnet. Daraus lässt sich für die gewählte Plattform und die gewählten EC Parameter die beste Ausprägung einer Punktmultiplikationsmethode und somit das schnellste Punktmultiplikationsverfahren bestimmen.

Die Ausführung dieser Schritte per Hand ist auch noch extrem aufwendig, sie kann jedoch automatisiert werden. Zu diesem Zweck wurde ein Programm in Java entwickelt, das diese Strategie ausführt. Seine Ausführung dauert - ist der erste Schritt erst getan - pro Multiplikationsalgorithmus nur wenige Sekunden. Für jede Ausprägung wird eine Laufzeit - in Millisekunden ausgedrückt - angegeben. Durch den einfachen Vergleich der Laufzeiten ist sofort ersichtlich, welches Verfahren mit welcher Ausprägung zum gewünschten Ziel führt.

Anwendung der Strategie

Nach Beschreibung und Analyse der Algorithmen und Beschreibung der Strategie wird der Bogen zur Praxis gespannt: Das Programm wurde auf einem AMD Athlon mit 1.3 GHz und 256 MB Arbeitsspeicher unter Linux mit Java Version „1.4.1_02“ mit allen vorgestellten Algorithmen zur Punktmultiplikation ausgeführt. Dabei werden 5 verschieden EC Parameter verwendet: Jeder Satz ist über $\mathbb{F}_p$ definiert, wobei p einmal eine Bitlänge von 140 Bit, von 160 Bit, von 192 Bit, von 197

Bit und von 272 Bit besitzt. Weiter wird vorausgesetzt, dass vorberechnete Punkte affin gespeichert werden und dass dabei eine Speicherkapazität von 50 KB nicht überschritten wird. Das Ergebnis der Programmausführung folgt:

Die Signaturerzeugung wird am besten mit dem Algorithmus von Lim und Lee [LL94] ausgeführt. Dabei werden so wenig Punktadditionen und Punktverdopplungen in der Hauptrechnung benötigt, dass eine große Anzahl an verschiedenen Ausprägungen zum gleichen Ergebnis führt:

- Punktverdopplungen werden mit Jacobischen oder Modifiziert Jacobischen Koordinaten ausgeführt.
- Additionen werden mit Chudnowsky Jacobischen oder Jacobischen Koordinaten ausgeführt.
- Jede Kombination dieser Möglichkeiten führt zu dem gleichen Ergebnis. Insbesondere ist die ausschließliche Verwendung Jacobischer Koordinaten in diesem Fall ebenso zu empfehlen, wie die Kombination der beschriebenen Möglichkeiten.
- Die Parameter h und v, die die Anzahl der Punktoperationen bestimmen, sind wie folgt zu wählen: Für 140 Bit mit $h = 10$ und $v = 2$, für 160 Bit mit $h = 7$ und $v = 8$, für 192 Bit mit $h = 6$ und $v = 16$, für 197 Bit mit $h = 8$ und $v = 4$ und für 272 Bit mit $h = 8$ und $v = 3$.
- Bei der ausschließlichen Verwendung von Chudnowsky Jacobischen, Projektiven, Modifiziert Jacobischen oder Affinen Koordinaten ist unabhängig von der Bitlänge des Exponenten ein etwas langsameres Ergebnis zu erwarten.

Die Ergebnisse sind im Detail in den Tabellen 9.20 bis 9.24 auf den Seiten 144 ff aufgeführt.

Beim Schlüsselaustausch kann im Allgemeinen nicht auf vorberechnete Punkte zurückgegriffen werden, da zumeist andere EC Parameter verwendet werden, als für das eigene Schlüsselpaar. Sollte dies nicht der Fall sein, gelten dieselben Ergebnisse wie bei der Signaturerzeugung. Andernfalls führt die Exponentiation mit Non-adjacent-forms mit folgenden Ausprägungen zum besten Ergebnis:

- Die Vorberechnung berechnet alle Punkte in Affinen Koordinaten.
- Jede Punktaddition gebe das Ergebnis in Jacobischen oder Modifiziert Jacobischen Koordinaten zurück. Sie bekommt einen der Summanden

aus den vorberechneten Affinen Punkten und einen aus der vorangegangenen Verdopplung als Jacobischen oder Modifiziert Jacobischen Punkt.

- Jede Punktverdopplung nach einer Addition gibt das Ergebnis in Modifiziert Jacobischen oder in Jacobischen Koordinaten zurück.
- Hintereinander ausgeführte Punktverdopplungen werden ausschließlich mit Modifiziert Jacobischen Koordinaten ausgeführt.
- Damit gibt es vier verschiedene Ausprägungen, die zum jeweils gleichen Ergebnis führen. Für jede der Bitlängen 140, 160, 192, 197 und 272 Bit muss dabei die Fenstergröße $w = 4$ gewählt werden.

Die detaillierten Ergebnisse sind in den Tabellen 9.9 bis 9.11 auf den Seiten 132 und 133 nachzulesen.

Bei einer Signaturverifikation kann sowohl der Fall auftreten, dass vorberechnete Punkte verwendet werden können, als auch, dass dies nicht möglich ist. In beiden Fällen ist die Verwendung der w-NAF-based Interleaving Exponentiation die beste Lösung. Dieser Algorithmus gehört zu denen, die eine Operation der Form $k_1 \cdot P_1 + k_2 \cdot P_2$ auf einmal berechnet.

Die beiden unterschiedlichen Fälle benötigen jedoch verschiedene Ausprägungen. Können keine vorberechneten Punkte verwendet werden, beachte man folgendes:

- Die Vorberechnung wird ausschließlich mit affinen Punkten berechnet.
- In der Hauptrechnung werden Modifiziert Jacobische Koordinaten verwendet.
- Die Fenstergrößen w_1 und w_2 werden bei jeder der Bitlängen 140, 160, 192, 197 und 272 Bit auf $w_1 = w_2 = 4$ gesetzt.

Die Tabellen 9.33 bis 9.37, die die Ergebnisse im Detail aufführen, sind auf den Seiten 167 bis 169 zu finden.

Kann bei der Verifikation auf vorberechnete Punkte zurück gegriffen werden, führen dieselben Ausprägungen zum besten Ergebnis, mit der Ausnahme, dass die erste Fenstergröße auf $w_1 = 11$ für 140, 160, 192 und 197 Bit gesetzt wird und auf $w_1 = 10$ für 272 Bit. Die zweite Fenstergröße wird in allen Fällen auf $w_2 = 4$ gesetzt. Dies kann in den Tabellen 9.43 bis 9.47 auf den Seiten 173 bis 175 nachgelesen werden.

Experimentelle Ergebnisse

Um einen Eindruck davon zu bekommen, welchen Einfluss diese Ergebnisse in der Praxis haben, wurden die besten Ausprägungen je Algorithmus in experimentellen Tests eingesetzt, deren Ergebnisse tabellarisch in Kapitel 10 notiert und analysiert sind.

Die Ergebnisse machen deutlich, dass es sich bei den hochgerechneten Werten um Erwartungswerte handelt. Sie basieren nicht auf der tatsächlichen Anzahl von Punktadditionen. Daher sind die experimentellen Ergebnisse nicht eins zu eins mit den hochgerechneten zu vergleichen. Meistens bestätigen die experimentellen Ergebnisse jedoch die hochgerechneten: Es gibt eine Ausnahme: Laut Hochrechnung müsste die Verwendung der w-Naf-based Interleaving Exponentiation bessere Laufzeiten ergeben als die Simultaneous Sliding Window Exponentiation. Dies ist bei den Experimenten jedoch nicht der Fall. Eine Untersuchung der Exponenten brachte eine Erklärung: Die tatsächlich benötigte Anzahl der Punktadditionen für die Interleaving Methode liegt um einiges höher, als der Erwartungswert vermuten lässt[5].

Die verschiedenen Ausprägungen werden von den Ergebnissen der Experimente bestätigt: Das Verwenden von ausschließlich einem Koordinatensystem - vor allem dem Affinen oder dem Projektiven - ist die schlechteste Wahl. Einzige Ausnahme ist das LimLee-Verfahren, was die Theorie wiederum bestätigt.

Die acht besten hochgerechneten Laufzeiten unterscheiden sich - wenn überhaupt - oft nur um Hundertstel- oder Zehntel Millisekunden. In dem Bereich liegen auch die Schwankungen der experimentellen Ergebnisse. Es werden die besten Laufzeiten für die vier verschiedenen Punktmultiplikationsarten gezeigt, dabei steht *uvP* für *unter Verwendung vorberechneter Punkte* und *ovP* für *ohne Verwendung vorberechneter Punkte*:

$k \cdot P$					
	140	160	192	197	272
Naf ovP	12.22	13.25	18.57	23.23	41.97
LimLee, uvP	1.86	2.2	3.2	3.79	7.24

[5]Diese Beobachtung legt die Vermutung nahe, dass der Erwartungswert nicht korrekt ist. Dies konnte in dieser Arbeit aus zeitlichen Gründen aber nicht verifiziert werden.

$k_1 \cdot P_1 + k_2 \cdot P_2$					
	140	160	192	197	272
Sim. Slid. Window, ovP	14.27	15.28	21.73	27.39	50.44
w-Naf Interl., uvP	13.49	14.76	20.58	25.58	46.75

Übertragung auf Referenzszenarien

Den Abschluss der Untersuchungen bilden die bereits vorgestellten Referenzszenarien: Experimentelle Tests untersuchen, wie sich die berechneten besten Verfahren und Ausprägungen in der Praxis bewähren. Die Ergebnisse sind durchaus befriedigend: Die Anforderungen eines Email-Benutzers werden leicht mit 8 ms bei 160 Bit und 10 ms bei 192 Bit in den Schatten gestellt. Werden viele Signaturen hintereinander erzeugt, sinkt die benötigte Zeit auf 2.23 ms, beziehungsweise auf 3.46 ms.

Eine einzelne Verifikation benötigt 71 ms für 160 Bit und 75 ms bei 192 Bit, wenn die Verwendung vorberechneter Punkt möglich ist. Bei vielen Iterationen sinkt die Laufzeit auch hier stark und zwar auf 15.5 ms und 21.69 ms pro Verifikation.

Können keine vorberechneten Punkte verwendet werden, liegen die Zeiten leicht erhöht bei 78 ms bei 160 Bit und 88 ms bei 192 Bit, beziehungsweise bei 17.89 ms und 24.21 ms bei vielen Verifikationen.

Ein einzelner Schlüsselaustausch ist in 23.0 ms für 160 Bit und 25 ms bei 192 Bit durchzuführen. Bei 1000 Iterationen sinken diese Zeiten auf 14.07 ms und 20.21 ms.

Dies zeigt, dass der Provider auch den Anforderungen genügt, die vom Szenario der digitalen Postwertzeichen gestellt werden.

Die Kundenserver der Bank müssen dagegen schon auf die vorgestellte Hardware-Lösung zurückgreifen. Mit bis zu vier Maschinen, die die Anfragen bearbeiten, ist auch in diesem Szenario der Provider den Anforderungen gewachsen.

Anders verhält es sich zur Zeit noch mit dem Provider auf dem PocketPC iPAQ von Compaq: Eine Signaturerzeugung und ein Schlüsselaustausch liegen bei 900 ms. Die Verifikation dagegen benötigt fast 4 sec, eine Zeitspanne, die auch für geduldige Menschen eine Zumutung ist. Eine Begründung und Details sind im Kapitel 11 nachzulesen.

Kurze Übersicht

Im Folgenden geben wir eine kurze Übersicht über die Gliederung dieser Arbeit:

Im Kapitel 2 werden die kryptographischen Grundbegriffe wie Public-Key Kryptographie, Signaturen, Schlüsselaustausch uvm. erläutert. Ebenso wird eine Einführung in die *Java Cryptography Architecture* (JCA) und *Java Cryptography Extension* (JCE) gegeben, da diese im FlexiProvider verwendet werden. Letzterer wird zum Abschluss dieses Kapitels beschrieben.

Kapitel 3 gibt eine Einführung in die Arithmetik von elliptischen Kurven (EC Arithmetik) über $\mathbb{F}_p$, sowie in das Diskreter-Logarithmus-Problem. Die Signaturverfahren *ECDSA* und *ECNR*, das Schlüsselaustauschverfahren *ECDH*, die Schlüsselgenerierung und ihre Parameter werden direkt nach der EC Arithmetik und dem ECDLP erläutert.

Die Anforderungen, die an eine flexible Krypto-Bibliothek gestellt werden, werden in Kapitel 4 definiert und erläutert. Da die Effizienzanforderung stark von den Anwendungen abhängt, werden vier Beispielszenarien vorgestellt, anhand derer die Anforderung definiert wird.

Das folgende Kapitel 5 beschreibt das Design und die Implementierung des FlexiECProviders. Es wird erläutert, wie der Provider aufgebaut ist, um Flexibilität und Erweiterbarkeit der Körperarithmetik und der Kryptoalgorithmen zu gewährleisten. Auf die Klassen `Point`, in der die Punktmultiplikationen implementiert sind, und `PointMixed`, die unter anderem Methoden zur Punktaddition und -verdopplung enthalten, wird gesondert eingegangen: Es wird erklärt, wie die gemischten Koordinaten innerhalb der Punktmultiplikationsmethoden von *außen* gesteuert werden können.

Kapitel 6 bildet die Grundlage zum Schwerpunkt dieser Arbeit: In ihm werden die elf bekanntesten Algorithmen zur einfachen und mehrfachen Punktmultiplikation vorgestellt. Anschließend werden die fünf bereits erwähnten Koordinatensysteme einander gegenübergestellt und miteinander verglichen.

Die bereits beschriebene Strategie und die Suite zu deren Ausführung werden im Kapitel 7 motiviert und im Detail beschrieben. Die daran anschließenden Kapitel wenden sich der Praxis zu, die Strategie wird auf die einzelnen Punktmultiplikationsalgorithmen angewandt:

Im Kapitel 8 werden die Ergebnisse des ersten Schritts, der Laufzeituntersuchung der Körperoperationen, präsentiert.

Die übrigen drei Schritte der Strategie werden in Kapitel 9 ausgeführt und ihre Ergebnisse kommentiert und tabellarisch aufgelistet. Dieses Kapitel schließt mit einer Analyse der Ergebnisse ab.

Im Kapitel 10 werden die Ergebnisse des vorherigen Kapitels experimentell untersucht. Es werden die Laufzeiten aufgelistet, die tatsächlich für eine Punktmultiplikation mit jedem der Algorithmen und seiner besten Ausprägung benötigt werden. Diese Ergebnisse wurden bereits in Kurzfassung erwähnt, ebenso wie die der Übertragung auf die Referenzszenarien, die im anschließenden Kapitel 11 präsentiert wird.

Die Arbeit schließt mit einer Zusammenfassung aller wichtigen Ergebnisse in Kapitel 12.

Kapitel 2

Kryptographische Grundbegriffe

In diesem Kapitel werden die in dieser Arbeit verwendeten kryptographischen Techniken digitale Signaturen, Verschlüsselung, Hashfunktionen, Schlüsselaustausch und MACs erläutert. Daran anschließend gibt es eine Einführung in die Bedeutung von Standards in der Kryptographie, die *Java Cryptography Architecture* und die *Java Cryptography Extension* [SUN], ein Framework zur einfachen Verwendung sowie Implementierung von kryptographischen Primitiven. Es wird zur Realisierung dieser Arbeit verwendet.

2.1 Public-Key Kryptographie

Kryptographie setzt sich aus der Public-Key- oder auch assymetrischen Kryptographie und der Secret-Key- oder auch symmetrischen Kryptographie zusammen. Ihre Techniken oder Primitive realisieren folgende Eigenschaften:

***Vertraulichkeit* (*Confidentiality*)** ist ein Service, der den Inhalt einer Information für alle außer Authorisierten verbirgt. Andere Synonyme sind *Geheimhaltung*, *secrecy*, oder *privacy*. Die Vertraulichkeit wird durch Verschlüsselung erzielt.

***Authentizität* (*Authentication*)** (*Echtheit, Zuverlässigkeit* oder auch *Glaubwürdigkeit* [Wis01]) bietet die Identifizierung. Dieser Service bietet zweierlei: Erstens soll die Identität einer Person sichergestellt werden, zweitens soll der Inhalt von Daten an eine Person gebunden werden. Zwei Personen, die

miteinander eine Kommunikation aufnehmen, sollen sich vorher authentifizieren und die Inhalte ihrer Nachrichten durch eine Signatur bestätigen. Dieser Service wird durch digitale Signaturen oder MACs realisiert.

***Integrität* (*Integrity*)** (*Makellosigkeit, Unbescholtenheit, Unbestechlichkeit* [Wis01]) hat das Ziel, unauthorisierte Veränderungen zu verhindern. Dies wird ebenfalls durch digitale Signaturen und MACs sowie durch message-digests erzielt.

***Nicht-Abstreitbarkeit* (*Non-Repudiation*)** bindet eine Person an frühere Aussagen oder Aktionen. Diese können im Nachhinein nicht abgestritten werden. Auch dieses Ziel wird mit digitalen Signaturen erreicht.

Im Folgenden beschreiben wir, wie die genannten Techniken diese Ziele durchsetzen.

2.1.1 Digitale Signatur

Digitale Signaturen sind Teil der Public-Key-Kryptographie. Sie realisieren die Authentizität und die Nicht-Abstreitbarkeit wie folgt:

Jeder Person ist ein Schlüsselpaar (e, d) zugeordnet. e ist der *öffentliche Schlüssel* oder *public key*, d ist der *private Schlüssel* oder *private key*. Der öffentliche Schlüssel wird in einem Zertifikat, das jeder einsehen kann, abgelegt. Er ist dort fest einer einzelnen Person zugeordnet. Der private Schlüssel bleibt für Dritte geheim.

Angenommen, Alice möchte ein Dokument m signieren. Sie verwendet ihren privaten Schlüssel d und berechnet die Signatur $s(d, m)$. Bob kann mit dem dazugehörigen öffentlichen Schlüssel e verifizieren, dass $s(d, m)$ wirklich die Signatur des Dokuments m ist.

Damit ist erstens sichergestellt, dass die Signatur $s(d, m)$ von Alice erstellt wurde, da kein Dritter den Schlüssel d kennt. Zweitens kann Alice die Signatur aus dem selben Grund nicht abstreiten. Und drittens garantiert diese Methode, dass das Dokument nicht im Nachhinein abgeändert werden konnte, da die Signatur mit dem Dokument verbunden ist.

Wichtig für das Signatur-Schema ist, dass man weder aus dem öffentlichen Schlüssel e noch aus der Signatur $s(d, m)$ Rückschlüsse auf den privaten Schlüssel d ziehen kann. Dies wird mit *Einweg-Funktionen* sichergestellt. Eine davon ist

das Diskrete-Logarithmus-Problem für endliche Körper (DLP) und für die Punkte-Gruppe elliptischer Kurven (ECDLP). Wir stellen es in Abschnitt 3.2.1 vor.

Es gibt zwei Arten von Signaturschemata:

1. *mit Appendix*: Der Signierer verschickt die Nachricht und die Signatur zusammen. Die Nachricht ist nicht aus der Signatur wiederherstellbar.

2. *mit message-recovery*: Der Signierer verschickt nur die Signatur, da die Nachricht aus der Signatur wiedergewonnen werden kann.

Die in dieser Arbeit behandelten Verfahren ECDSA und ECNR gehören zu der ersten Kategorie. Der Nachteil der ersten Methode ist, dass die Signatur an die originale Nachricht angehängt werden muss. Zur Länge der Nachricht kommt also noch die Länge der Signatur hinzu. Daher sind Signaturen kurzer Länge durchaus attraktiv.

2.1.2 Schlüsselaustausch

Der Vorteil der Public-Key Kryptographie liegt darin, dass für jede Partei nur ein Schlüsselpaar benötigt wird, wogegen symmetrische Verfahren je einen geheimen Schlüssel für jedes Kommunikationspaar brauchen. Der Nachteil ist aber, dass Public-Key Verfahren wesentlich aufwendiger und daher langsamer als symmetrische Verfahren sind. Daher kombiniert man die Techniken: Mit Public-Key Verfahren werden symmetrische Schlüssel ausgetauscht, die dann wiederholt zum Verschlüsseln verwendet werden. Zu diesen Verfahren gehört Diffie-Hellman für elliptische Kurven (ECDH) (siehe 3.2.6) und Menezes-Qu-Vanstone für elliptische Kurven (ECMQV) (siehe [IEE, Kapitel 9.4]), das in dieser Arbeit nicht behandelt wird.

Wir beschreiben das Prinzip anhand des Diffie-Hellman Schlüsselaustauschs: Alice besitzt das Schlüsselpaar (e_A, d_A), Bob das Paar (e_B, d_B). Beide einigen sich auf einen gemeinsamen (öffentlichen) Schlüssel e_C. Alice wählt einen geheimen Schlüssel a und berechnet $E_A = a \cdot e_C$ und schickt das Ergebnis E_A Bob. Dieser wählt seinerseits einen geheimen Schlüssel b und berechnet $E_B = b \cdot e_C$ und schickt das Ergebnis Alice. Alice berechnet nun $a \cdot E_B = a \cdot b \cdot e_C$ und Bob $a \cdot E_B = b \cdot a \cdot e_C$.

2.1.3 Verschlüsselung

Mit Hilfe von Verschlüsselungsverfahren können Nachrichten verschlüsselt werden. Dritte, die nicht im Besitz des nötigen Schlüssels sind, können die Nachricht nicht lesen. Im Allgemeinen sind symmetrische Verfahren schneller und werden daher zur Verschlüsselung eingesetzt. Es gibt aber auch asymmetrische sowie *hybride* Verfahren. Letztere setzen sich aus einem asymmetrischen Schlüsselaustausch mit anschließendem symmetrischen Verschlüsseln zusammen. Hierzu gehört das *Integrated Encryption Scheme* (*IES*). Das EC-Pendant, *ECIES*, wird in Abschnitt 3.2.7 beschrieben.

2.1.4 Kryptographische Hash-Funktion

Hash-Funktionen werden für beinahe jedes digitale Signatur- oder Verschlüsselungs-Schema verwendet. Sie unterstützten die Daten-Integrität und bilden eine beliebig lange Bit-Kette auf eine Bit-Kette fester Länge ab. Die bekannteste Hash-Funktion ist SHA-1, die auch in dem in dieser Arbeit verwendeten Signaturverfahren ECDSA integriert ist.

Um für kryptographische Zwecke geeignet zu sein, muss es praktisch unmöglich sein, zwei verschiedene x und y zu finden, die auf den gleichen Wert gehasht werden ($h(x) = h(y)$). Es muss weiterhin praktisch unmöglich sein, zu einem gegebenen Hashwert y eine Eingabe x zu finden mit $h(x) = y$ [MvV97, Buc01]. Mit diesen Eigenschaften kann eine Hashfunktion wie folgt die Integrität eines Dokuments wahren:

Ein Dokument m wird gehasht. Die Integrität des Hashwerts $h(m)$ wird auf geschützt aufbewahrt. Später, zu einem anderen Zeitpunkt kann von dem Dokument m erneut der Hashwert gebildet werden. Nur, wenn dieser der selbe ist wie der alte, ist das Dokument unverändert.

Es liegt auf der Hand, dass es nicht viele ”sichere” Hashfunktionen gibt. Für ECDSA wird die Hash-Funktion SHA-1 verwendet.

2.1.5 MAC

Message authentication codes (MACs) sind das symmetrische Pendant zu digitalen Signaturen. Während diese neben der Authentizität auch die Nicht-Abstreitbarkeit garantieren, können MACs nur für die Dauer der Kommunikation die Authentizität

erwirken. Eine Nicht-Abstreitbarkeit ist durch einen MAC nicht gegeben.

In dieser Arbeit wird auf MACs nicht weiter eingegangen, da sie kein Teil der Elliptische-Kurven-Kryptographie sind.

2.1.6 Standards

Um zwischen verschiedenen Providern die Interoperabilität zu erhalten, müssen die kryptographischen Protokolle sowie ihre Objekte einheitlich sein. Dafür gibt es Standards.

Ein Standard ist als Referenz für Spezifikationen verschiedener Public Key-Verfahren zu sehen. Er behandelt viele verschieden Typen von Techniken und legt ein abstraktes Schema als Struktur zugrunde. Es wird zwischen folgenden Bausteinen unterschieden:

Primitive: Grundlegende mathematische Operationen wie das Faktorisieren großer Zahlen oder das Lösen diskreter Logarithmen. Mit Primitiven alleine kann man noch keine Sicherheit erzeugen, sie sind aber die Grundbausteine der verschiedenen Verfahren.

***Schemes* oder Schemata:** Schemata verbinden die Basisoperationen zu *Techniken*. Zu ihnen gehören das Prinzip der Schlüsselaustauschverfahren, der Signaturen mit und ohne message recovery und das der Ver- und Entschlüsselungen.

Protokolle: Protokolle sind die Aneinanderreihung von Operationen, die von mehreren Parteien ausgeführt werden, um ein gemeinsames Maß an Sicherheit zu erreichen.

Die Schlüsselerzeugung, das Signieren und das Verifizieren des ECDSA gehören zu den Schemata, die auf den Primitiven des Lösens des *Diskrete Logarithmus-Problems auf elliptischen Kurven* beruhen.
Das in dieser Arbeit entwickelte Verfahren richtet sich nach dem Standard, um die Möglichkeit einer zukünftigen öffentlichen Nutzung nicht zu verbauen.

Die wichtigsten Standards wurden von drei verschiedenen Gremien entworfen: Von IEEE stammt der Standard 1363 [IEE]. Er befasst sich allgemein mit Public Key Kryptographie. Der ANSI Standard X9.62 [X9.99] beschreibt nur den Digitalen Signatur Algorithmus für elliptische Kurven (ECDSA). SEC1 und SEC2

von Certicom [Sta00a, Sta00b] definieren ebenfalls nur elliptische Kurven Primitive. Sie beschränken sich jedoch nicht auf ECDSA. Diese drei Standards beschreiben teilweise nicht nur die gleichen Algorithmen, sie beschreiben sie zum Glück alle gleich. Der Unterschied besteht im Inhalt. Während IEEE's 1363 ein Standard für Public-Key Kryptographie allgemein ist, beinhaltet er keine Codierungs-Anweisungen. Diese kann man wiederum im SEC1 und X9.62 finden, soweit sie bisher vorhanden sind.

2.2 JCA und JCE

Die Java Cryptography Architecture (*JCA*) und die Java Cryptography Extension (*JCE*) wurde von SUN entwickelt und gehört standardgemäß zum *Java Development Kit* (kurz *JDK*) Version 1.2 aufwärts dazu. Sie bieten dem Anwender die Möglichkeit, Sicherheitsfunktionalität in seine Applikationen zu integrieren. Hier soll nur ein kurzer Einblick in die Idee und die Architektur der JCA und JCE gegeben werden. Für ein tieferes Studium wird auf [SM98b] und auf [Knu98] verwiesen.

2.2.1 Die Idee

Die JCA wurde entworfen, um Entwicklern die Möglichkeit zu geben, die drei Grundkonzepte der Public-Key Kryptographie (siehe 2.1) zu realisieren. Sie bietet das Konzept der Verschlüsselung, um Vertraulichkeit zu gewährleisten, das Konzept der Fingerprints, um Dokumentmanipulationen vorzubeugen und das Konzept der Signaturen und Zertifikate, um Fälschern einen Riegel vorzuschieben. Anwender sollen sich aber nicht um Implementierungen kümmern müssen. Als weitere Ziele sind angestrebt:

1. Implementierungsunabhängigkeit
2. Erweiterbarkeit
3. Interoperabilität

Um dies zu erreichen, trennt die JCA und die JCE (Java Cryptographic Extension, eine Erweiterung der JCA) kryptographische Konzepte von ihren Implementierungen. Die Konzepte sind im Paket *java.security* gesammelt, die Implementationen werden von sogenannten Providern gestellt.

Anwender müssen sich nicht mit einer speziellen Implementierung befassen. Sie können beispielsweise ein Signaturobjekt anfordern, ohne sich Gedanken zu machen, welcher Algorithmus hinter welcher Implementation steckt. Andererseits kann ein Anwender sehr spezielle Wünsche anmelden, indem er nicht nur sagt „erzeuge mir das Objekt“, sondern indem er sagt „erzeuge mir das Objekt vom Typ ECDSA implementiert vom Provider ‘FlexiECProvider’ “ - vorausgesetzt natürlich, der Provider „FlexiECProvider’“ ist installiert und bietet die Implementierung für ECDSA an. JDK Version 1.2 aufwärts wird standardmäßig mit dem Provider „SUN“ geliefert, der ein paar der kryptographischen Algorithmen implementiert.

Diese Providerarchitektur sorgt nicht nur für die Implementierungsunabhängigkeit, sondern auch für die Erweiterbarkeit: In dieser Arbeit wird das Angebot an digitalen Signaturalgorithmen um zwei erhöht.

Außerdem ist es einfacher, Algorithmen auf dem neuesten Stand zu halten: Man kann jederzeit neuere Versionen einer Implementierung durch entweder denselben oder einen neuen Provider mit Versionsangabe anbieten.

Unter Interoperabilität versteht man die Zusammenarbeit von verschiedenen Implementierungen. So kann zum Beispiel ein Schlüssel für einen Algorithmus von einem Provider erzeugt und von einem anderen verwendet werden.

2.2.2 Der Aufbau

Die Methoden der Konzeptklassen sind in zwei Gruppen aufgeteilt:

1. Methoden, die man aufrufen kann, um mit einem kryptographischen Verfahren zu arbeiten. Sie sind alle `public` und in der *API* (*Application Programming Interface*, zusammengefasst.
2. Methoden, die die Provider implementieren müssen. Sie gehören zum *SPI* (*Service Provider Interface*). Diese Methoden fangen per Konvention alle mit `engine` an.

Um zum Beispiel einen Signaturalgorithmus zu implementieren, muss man von der Klasse `SignatureSPI` erben und die darin liegenden Methoden definieren.

Die Methoden, die Instanzen von Klassen erzeugen, werden *factory method*s genannt. Sie haben alle den Namen `getInstance`. Um ein Signaturobjekt vom Typ ECDSA zu erhalten, müsste man folgenden Code anwenden:

```
Signature ecdsa;
ecdsa = Signature.getInstance(‘‘ECDSA’’)
```

Wird der spezifizierte Algorithmus in keinem der installierten Provider gefunden, löst dies eine `NoSuchAlgorithmException` aus.
Wünscht man keinen speziellen Algorithmus, ruft man `getInstance` ohne Parameter auf. Man kann auch einen Algorithmus eines bestimmten Providers verlangen, indem man seinen Namen als zweiten Parameter angibt: `ecdsa.getInstance(‘‘ECDSA’’, ‘‘FlexiECProvider’’);` Analog zum Fall des fehlenden Algorithmus wird eine `NoSuchProviderException` erzeugt, sollte der gewünschte Provider nicht installiert sein.
Die zurückgegebene Instanz ist ein Nachfahre der verlangten Klasse. Der Vorteil dieser Struktur liegt auf der Hand: Man muss sich als Anwender nicht um Implementierungen kümmern. Wann immer man ein Objekt, sei es ein Signatur-, ein Schlüssel- oder sonst ein Objekt braucht, ruft man einfach die dazugehörige `getInstance`-Methode auf.

Wie schon erwähnt sind es die Provider, die eine Implementierung letzten Endes zur Verfügung stellen. Wir wollen jetzt präziser werden:
In der JCA wird zwischen dem Begriff *provider* (klein geschrieben) und dem Begriff *Provider* (groß geschrieben) unterschieden. Ein provider ist eine Sammlung von Algorithmusklassen, die durch ein `java.security.Provider`-Objekt aufgelistet werden. Provider (groß geschrieben) ist also eigentlich eine Klasse und wird daher im Folgenden zur Verdeutlichung wie einKlassenname im Truetype geschrieben: `Provider`.

`java.security.Security` ist die Klasse, die die provider verwaltet. Sie führt eine Liste aller `Provider`objekte. Wird eine factory method aufgerufen, fragt sie jeden der `Provider` nach der Implementierung des gefragten Algorithmus. Jeder `Provider` wiederum kennt seine provider und weiß, was für Klassen sie beinhalten. Wenn möglich wird eine entsprechende Klasseninstanz zurückgegeben.
Die installierten provider sind nach Präferenz geordnet. Wird nach keinem bestimmten provider gefragt, wird der erste, der den entsprechenden Algorithmus enthält, verwendet. Das Konfigurieren von Providern ist zum Beispiel in [SM98b] oder [Knu98] nachzulesen.

2.3 FlexiECProvider

Der FlexiECProvider entstand an der TU Darmstadt in der Arbeitsgruppe von Prof. Johannes Buchmann als Teil der großen Kryptobibliothek FlexiProvider [FG03]. Sie umfasst alle gängigen und standardisierten Kryptoverfahren, sowohl symmetrische als auch asymmetrische. Zwei spezielle Teile dieses Providers sind der FlexiNFProvider für Zahlkörperkryptographie und der FlexiECProvider, der Kryptographie mit elliptischen Kurven bietet.

Der gesamte Provider ist in Java geschrieben und JCA-basiert. Die JCA wurde bereits im vorangegangen Abschnitt beschrieben. Für den FlexiECProvider bedeutet das hohe Flexibilität, Plattformunabhängigkeit und einfache Anwendbarkeit durch Entwickler.

Die EC-Kryptographie umfasst die Signaturverfahren ECDSA und ECNR und den Diffie-Hellman-Schlüsselaustausch für elliptische Kurven. Diese Verfahren werden im Kapitel 3.2 erklärt. Zusätzlich wurde das Verschlüsselungsverfahren implementiert, welches in dieser Arbeit aber keinen Schwerpunkt darstellt. Die EC-Arithmetik ist sowohl über Primkörpern als auch über endlichen Körpern der Charakteristik 2 (siehe 3.1) implementiert. Im Zug dieser Arbeit wurde der Schwerpunkt auf die Optimierung der Arithmetik auf die über Primkörper gelegt.

Kapitel 3

Elliptische Kurven Kryptographie - ECC

In den folgenden zwei Abschnitten werden alle Grundbegriffe zu elliptischen Kurven erklärt, die zum Verständnis der Arbeit nötig sind. Abschnitt 3.2 führt in die Elliptische Kurven Kryptographie ein, sowie in das Diskreter-Logarithmus-Problem, Signaturen, Schlüsselaustausch und Verschlüsseln mit elliptischen Kurven.

3.1 Elliptische Kurven

Eine *Weierstraßgleichung* ist eine homogene Gleichung vom Grad 3 der Form

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3, \qquad (3.1)$$

mit $a_1, a_2,\ a_3,\ a_4,\ a_6 \in \overline{K}$.
Sie heißt *glatt*, wenn für alle Punkte $P = (X : Y : Z)$, die die Gleichung

$$F(X, Y, Z) = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3 = 0$$

erfüllen, zumindest eine der drei formalen partiellen Ableitungen $\frac{\partial F}{\partial X},\ \frac{\partial F}{\partial Y},\ \frac{\partial F}{\partial Z}$ ungleich 0 ist. Ansonsten heißt die Weierstraßgleichung *singulär*.

Eine elliptische Kurve E über einem Körper $\mathbb{K}$ ist die Menge aller Punkte, die eine glatte Weierstraßgleichung erfüllen. Es gibt genau einen Punkt P, bei dem die Z-Koordinate 0 ist: $P = (1 : 1 : 0)$. Dieser Punkt wird *Punkt im Unendlichen* genannt und fortan in dieser Arbeit mit $\mathcal{O}$ bezeichnet.

Um zu entscheiden, ob eine Kurve glatt ist, müssen noch einige Größen eingeführt werden:

$$\begin{aligned}
b_2 &= a_1^2 + 4a_2 \\
b_4 &= 2a_4 + a_1a_2 \\
b_6 &= a_3^2 + 4a_6 \\
b_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \\
c_4 &= b_2^2 - 24b_4 \\
c_6 &= b_2^3 - 36b_2b_4 - 216b_6 \\
\Delta(E) &= -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6 \\
j(E) &= \frac{c_4^3}{E(\Delta)}, \quad \text{für } E(\Delta) \neq 0
\end{aligned}$$

$\Delta(E)$ heißt *Diskriminante* von E und $j(E)$ wird *j-Invariante* genannt. Für eine Kurve E in Weierstraßform nach (3.1) gilt:

- E ist glatt $\Leftrightarrow \Delta(E) \neq 0$
- E hat einen Knoten $\Leftrightarrow \Delta(E) = 0$ und $c_4 \neq 0$
- E hat eine Spitze $\Leftrightarrow \Delta(E) = 0$ und $c_4 = 0$

Der Beweis ist in [Sil86, Seite 50 f] nachzulesen.

Um die Form elliptischer Kurven zu verdeutlichen, ist in Abbildungen die affine Darstellung gewählt. Abbildung 3.1 zeigt zwei singuläre Kurven. Die Tangente im *Knoten* der ersten Kurve ist nicht eindeutig, da die Laufrichtung nicht beachtet wird. Die Tangente in der Spitze der zweiten Kurve hat die Steigung ∞.

Für die affine Darstellung der Weierstraßgleichung (3.1) setzt man $x = X/Z$ und $y = Y/Z$. Die Kurvengleichung hat dann die Form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{3.2}$$

mit $a_1,\ a_2,\ a_3,\ a_4,\ a_6 \in \mathbb{F}_q$.

Sei $\mathbb{K}$ ein Körper der Charakteristik $\mathrm{char}(\overline{\mathbb{K}}) \neq 2$. Dann kann die Weierstraßform durch die Substitution $[y \to \frac{1}{2}(y - a_1x - a_3)]$ zu

$$y^2 = x^3 + b_2x^2 + b_4x + b_6 \tag{3.3}$$

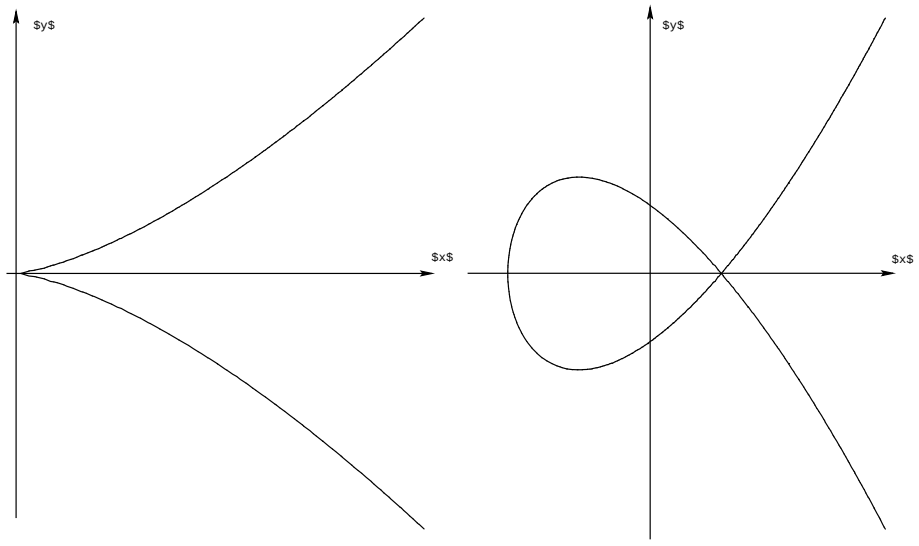


Abbildung 3.1: Singuläre Kurven

vereinfacht werden.
Ist $\mathrm{char}(\overline{\mathbb{K}}) \neq 2, 3$, wird durch die Substitutionen $[x \rightarrow \frac{x-3b_2}{36}]$ und $[y \rightarrow \frac{y}{216}]$ aus (3.3) die Gleichung

$$y^2 = x^3 + ax + b, \tag{3.4}$$

mit $a = -27c_4$ und $b = -54c_6$.
Die schon eingeführte projektive Darstellung ist

$$Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{3.5}$$

und die affine Darstellung ist entsprechend

$$y^2 = x^3 + ax + b \tag{3.6}$$

Es gibt beliebig viele verschiedene Koordinatensysteme, in denen die elliptischen Kurven dargestellt werden können. Durch manche Koordinatensysteme ist eine effizientere Implementierung der Arithmetik möglich. Diese werden für den Körper $\mathbb{F}_p$ im Abschnitt 6.4 vorgestellt und diskutiert.

Eine Abschätzung der Anzahl der Punkte auf elliptischen Kurven ist durch folgenden Satz gegeben:

Hasses Theorem: Sei $E(\mathbb{K})$ eine elliptische Kurve über einem endlichen Körper $\mathbb{K}$ mit q Elementen. Dann kann man über die Anzahl der Punkte von E folgende

Aussage treffen:

$$\#E(\mathbb{K}) = q + 1 - t \qquad \text{mit} \quad t < |2\sqrt{q}|$$

Der Beweis dieses Theorems ist in [Sil86, Seite 130] nachzulesen.

3.1.1 Elliptische Kurven über $\mathbb{F}_p$

In der Kryptographie werden die elliptischen Kurven über $\mathbb{F}_p$ oder $\mathbb{F}_{2^m}$ verwendet. Diese Arbeit befasst sich fast ausschließlich mit denen über $\mathbb{F}_p$.

Die Punkte auf einer elliptischen Kurve E bilden zusammen mit der inneren Abbildung *Punktaddition* und dem Punkt im Unendlichen $\mathcal{O}$ als neutrales Element eine additive Abel'sche Gruppe. Sei E eine elliptische Kurve über $\mathbb{F}_p$ und seien P und Q zwei Punkte auf E, dann gilt Folgendes:

1. $P + \mathcal{O} = \mathcal{O} + P = P$
2. $P + Q$ liegt ebenfalls auf E
3. wenn $P = (x,\ y)$, dann ist $-P = (x,\ -y)$
4. $P + (-P) = \mathcal{O}$

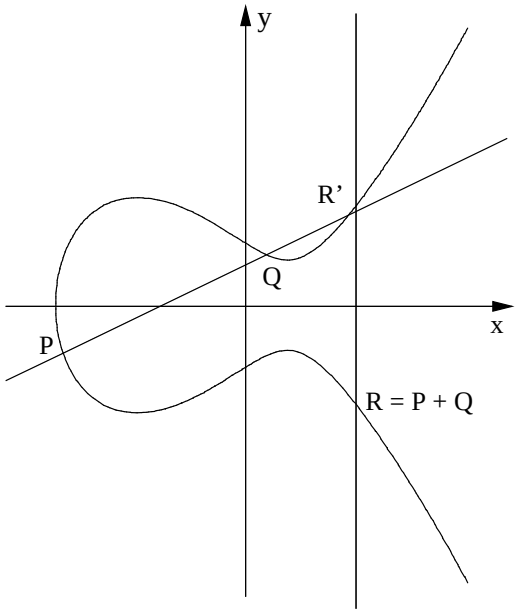
Die Punktaddition lässt sich grafisch beschreiben: Lege durch die beiden Summanden P und Q eine Gerade. Sie schneidet die Kurve in einem weiteren Punkt R', welcher an der x-Achse gespiegelt wird. Der resultierende Punkt ist $R = P + Q$. Abbildung 3.2 illustriert eine Punktaddition.

Es gibt zwei Sonderfälle: Wenn $P = Q$ ist, dann ist die Gerade durch P und Q die Tangente der Kurve im Punkt P, wie in Abbildung 3.3 gezeigt. In diesem Fall spricht man von einer *Punktverdopplung*.

Ist $Q = (-P)$, geht die Gerade, die die Kurve sonst dreimal schneidet, senkrecht durch die Punkte P und Q. Abbildung 3.4 illustriert diesen Fall.

Es seien $P = (x_P,\ y_P)$ und $Q = (x_Q,\ y_Q)$. Dann lässt sich $P + Q = R = (x_R,\ y_R)$ wie folgt berechnen:

$$\begin{aligned} x_R &= \lambda^2 - x_P - x_Q \\ y_R &= \lambda(x_R - x_P) - y_P \\ \lambda &= (y_Q - y_P)/(x_Q - x_P) \end{aligned} \tag{3.7}$$



Die Punktaddition von P und Q.

Abbildung 3.2: Eine Punktaddition $P + Q$

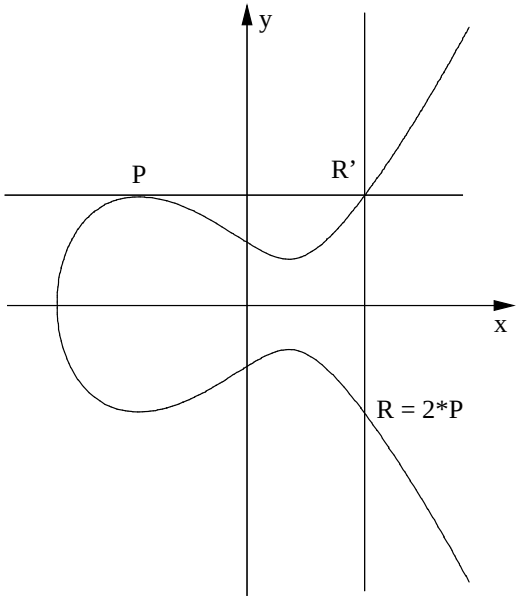


Abbildung 3.3: Eine Punktverdopplung $P + Q$ mit $P = Q$

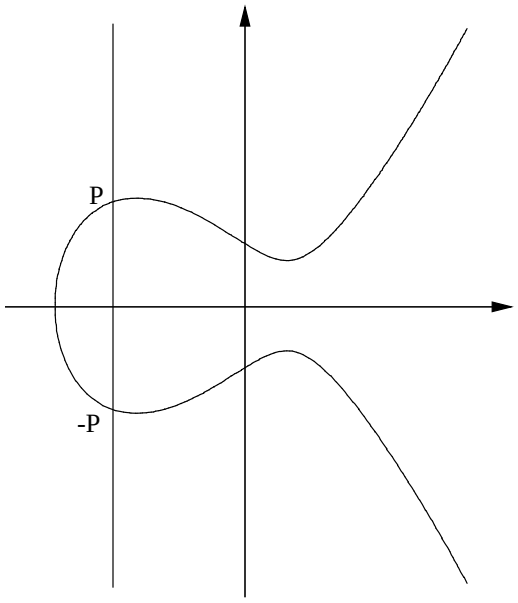


Abbildung 3.4: Eine Punktaddition $P + Q$ mit $Q = (-P)$

Für $R = (x_R,\ y_R) = 2 \cdot P$ gelten folgende Formeln:

$$\begin{aligned} x_R &= \lambda^2 - 2x_P \\ y_R &= \lambda(x_R - x_P) - y_P \\ \lambda &= (3x_P^2 + a)/(2y_P) \end{aligned} \tag{3.8}$$

λ ist die Steigung der Geraden durch P und Q. Daher gibt es auch in der Berechnung von $P + Q = R$ eine Unterscheidung, wenn $P = Q$ ist. Dann ist λ die erste Ableitung in P.

Beispiel 1: Punkte auf einer Kurve

Es sei $E(\mathbb{F}_{23}) = x^3 + 3x + 22$ eine elliptische Kurve über dem Primkörper $\mathbb{F}_{23}$. Sie hat 33 Punkte:
$(1, 16), (1, 7), (2, 6), (2, 17), (3, 9), (3, 14), (4, 12), (4, 14), (5, 1),$
$(5, 22), (6, 16), (6, 7), (7, 8), (7, 15), (8, 12), (8, 11), (11, 12), (11, 11),$
$(13,\ 2),\ (13,\ 21),\ (14,\ 18),\ (14,\ 5),\ (16,\ 16),\ (16,\ 7),\ (17,\ 8)$
$(17,\ 15),\ (20,\ 3),\ (20,\ 20),\ (21,\ 10),\ (21,\ 13),\ (22,\ 8),\ (22,\ 15)$
$(0,\ 0)$

3.2 Kryptographie mit elliptischen Kurven

In diesem Abschnitt wird gezeigt, wie man die Gruppe der Punkte auf einer elliptischen Kurve für die Kryptographie einsetzen kann. Dazu wird in 3.2.1 das Diskrete-Logarithmus-Problem vorgestellt, das zu lösen als sehr schwierig betrachtet wird und daher als Einwegfunktion verwendet wird. In den darauffolgenden Abschnitten, 3.2.2 und 3.2.3 werden die Bedeutung der ECParameter und die Generierung der Schlüssel erläutert. In 3.2.4 und 3.2.5 werden die Signaturverfahren ECDSA (DSA für elliptische Kurven) und ECNR (Nyberg-Rueppel für elliptische Kurven) skizziert, Abschnitt 3.2.6 geht auf das Schlüsselaustauschverfahren ECDH (Diffie-Hellman für elliptische Kurven) ein. Der Abschnitt 3.2.7 beschreibt das hybride Verschlüsselungsverfahren ECIES (*Elliptic Curve Integrated Encryption Scheme*).

3.2.1 Das Diskreter-Logarithmus-Problem

Sei $\mathbb{G}$ eine endliche zyklische Gruppe der Ordnung r, γ sei ein Erzeuger dieser Gruppe und 1 sei das neutrale Element. α sei ein weiteres Gruppenelement und

$$\alpha = \gamma^x.$$

Dann heißt x der **diskrete Logarithmus** von α zur Basis γ. Das Berechnen des diskreten Logarithmus ist schwierig und wird das Diskreter-Logarithmus-Problem (DLP) genannt. Es eignet sich als Einwegfunktion für kryptographische Zwecke.

Beispiel 2: DLP

Es sei $\mathbb{G} = \mathbb{F}_{13}$ der Primkörper mit 13 Elementen. $\gamma = 2 \in \mathbf{F}_{13}$ ist ein Erzeuger und $\alpha = 5$ ist ein weiteres Element dieser Gruppe. Zu lösen sei die Gleichung $5 = 2^x$. Der diskrete Logarithmus x von 5 zur Basis 2 ist 9, denn $2^9 \mod 13 \equiv 5$. Es ist einfach, $2^9 \mod 13$ zu berechnen, es ist dagegen *schwer*, das x zu berechnen.

Das DLP ist direkt auf elliptische Kurven zu übertragen: Sei $\mathbb{G}$ eine endliche Gruppe der Punkte auf einer elliptischen Kurve mit Ordnung r. G sei ein Erzeuger

dieser Gruppe und P ein weiteres Element mit $P = k \cdot G$. Dann ist k der diskrete Logarithmus von Punkt P zur Basis G.

Es gibt verschiedene Algorithmen den diskreten Logarithmus zu berechnen. Zu nennen sind der *Baby-step giant-step algorithm*, der *Pohlig-Hellman algorithm*, der *Pollard-Rho algorihtm* und der *Index-calculus algorithm* [MvV97, Kap. 3.6, S. 103 ff]. Im Gegensatz zu den Kryptoverfahren basierend auf dem DLP über $\mathbb{Z}_p^*$ ist bisher aber kein Algorithmus bekannt, der in subexponentieller Laufzeit das ECDLP für nicht-singuläre Kurven lösen kann [MvV97, Kap. 3.12, S. 130 ff, Abschnitt 3]. Daher sind die Schlüssellängen in der Elliptische Kurven Kryptographie viel kürzer als bei den Verfahren über $\mathbb{Z}_p^*$.

3.2.2 EC Parameter

Wir kommen nun zu den EC Krypto Algorithmen. Dabei halten wir uns an die Notationen der Standards, die schon in 2.1.6 beschrieben wurden.

Alle EC-Schemata haben einen Satz von *EC Domain Parametern* gemeinsam. Sie definieren den zugrunde liegenden Körper, die Kurve, einen Basispunkt auf der Kurve und dessen Ordnung. Mit diesem Satz lässt sich jedes EC-Schema ausführen. Im Detail besteht ein solcher Satz aus:

q	: spezifiziert einen Körper $\mathbb{F}_q$. q ist eine positive, ungerade Primzahl p oder $q = 2^m$, $m \in \mathbb{N}$.
$a,\ b$	: definieren die Kurve E: $y^2 = x^3 + ax + b$, $a,\ b \in \mathbb{F}_q$.
r	: Primzahl, die die Ordnung der Kurve E teilt und Ordnung des Punktes G ist ($\operatorname{ord} G = r$).
G	: Punkt mit Ordnung r, Erzeuger der Untergruppe mit Ordnung r.
k	: der Kofaktor, Quotient $\#E/r$.

In dieser Arbeit ist das q, also die Charakteristik des Körpers $\mathbb{F}_q$, immer eine große Primzahl p. Im Folgenden schreiben wir daher $\mathbb{F}_p$ für den Körper $\mathbb{F}_q$ mit p Elementen.

Beispiel 3: EC Parameter

Folgenden Parameter Satz werden wir für die nachfolgenden Beispiele verwenden:

$q = 23$, also der Körper $\mathbb{F}_{23}$ mit 23 Elementen.
$a = 3,\ b = 22,\ E(\mathbb{F}_{23}) = x^3 + 3x + 22$
$r = 11$. E hat 33 Punkte, $r = 11$ ist Primteiler und Ordnung des Punktes
$G = (6,\ 16)$.
$k = 33/11 = 3$.
Von G wird folgende Gruppe mit 11 Elementen erzeugt:
$\{(6, 16), (20, 3), (13, 2), (8, 11), (21, 10), (21, 13), (8, 12), (13, 21), (20,\ 20),\ (6,\ 7),\ (0,\ 0)\}$

3.2.3 EC Schlüsselpaare

Jedes EC-Schlüsselpaar ist an einen Satz EC Domain Parameter gebunden und besteht aus einem privaten Schlüssel s und dem öffentlichen Schlüssel W. Die Erzeugung ist für alle EC-Schemata der gleiche Vorgang:

Input EC Domain Parameter: $q,\ a,\ b,\ r,\ G$

Output private key $s \in Z$ und public key $W \in E(\mathbb{F}_q)$

Operation

1. Generiere eine zufällige ganze Zahl s im Intervall $[1,\ r-1]$.
2. Berechne $W = sG$ ($W \neq \mathcal{O}$), da G Erzeuger und $s < r$).
3. Das Schlüsselpaar ist $(s,\ W)$, wo s der private und W der geheime Schlüssel ist.

Beispiel 4: EC Schlüsselpaare

Wir verwenden den Parameter Satz aus Beispiel 3: $q = 23$, $E(\mathbb{F}_{23}) = x^3 + 3x + 22$, $r = 11$ mit $G = (6,\ 16)$ und $k = 3$.
Dann ist der private Schlüssel eine ganze Zahl s mit $1 \leq s < 11$. Sei $s = 3$ und $W = s \cdot G = 3 \cdot (6,\ 16) = (13,\ 2)$. Wir werden diese Schlüssel auch in den nachfolgenden Beispielen verwenden.

3.2.4 ECDSA

ECDSA, der *Digital Signatur Algorithm for Elliptic Curves*, veröffentlicht wie in [IEE], basiert auf den Arbeiten von [Mil86], [Kob87] und [Kra93]. ECDSA wird zum digitalen Signieren und Verifizieren verwendet. Bei der Verifikation werden weder die Nachricht selbst noch deren Hashwert wieder hergestellt. Bei der Verifikation wird nur bestimmt, ob die Signatur korrekt ist oder ob sie korrupt ist.

Für jede ECDSA-Signaturerzeugung wird ebenfalls ein Satz von EC Domain Parameter benötigt, sowie ein Schlüsselpaar wie in Abschnitt 3.2.3 beschrieben:

Input EC Domain Parameter $q,\ a,\ b,\ r,\ G$ und außerdem der dazu private key s, der mit denselben Parametern erzeugt wurde, und der ganzzahlige Hashwert f der Nachricht M.

Output die Signatur (c, d), mit $0 < c < r$ und $0 < d < r$.

Operation

1. Generiere ein *one-time key pair* $(u,\ V)$ mit den EC Domain Parametern, die schon für s verwendet wurden.
   $V = uG := (x_V,\ y_V) \neq \mathcal{O}$
2. Berechne $c = x_V \mod r$. Ist $c = 0$, starte erneut bei 1.
3. Berechne $d = u^{-1}(f + sc) \mod r$. Ist $d = 0$, starte erneut bei 1.
4. Die Signatur ist $(c,\ d)$.

Die Verifikation benötigt ebenfalls wieder die EC Domain Parameter sowie den öffentlichen Schlüssel des Signierers.

Input EC Domain Parameter q, a, b, r, G und außerdem der dazu passende public key W, der ganzzahlige Hashwert f der Nachricht M und die Signatur (c, d).

Output `true`, wenn die Signatur authentisch ist, `false`, wenn sie nicht authentisch ist oder die Schlüssel nicht zu den EC Domain Parametern passen.

Operation

1. Prüfe, ob $c,\ d \in [1,\ r - 1]$. Wenn nicht, return `false`.
2. Berechne $h = d^{-1} \mod r$, $h_1 = fh \mod r$ und $h_2 = ch \mod r$.
3. Berechne $P = h_1G + h_2W$.
   Ist $P = \mathcal{O}$, return `false`. Sonst:
   $P := (x_P,\ y_P)$.

4 Ist $c = x_P \mod r$, return `true`, sonst `false`.

Beispiel 5: ECDSA

Wir verwenden die Parameter und die Schlüssel aus den vorherigen Beispielen:

Die EC Domain Parameter:
$q = 23$
$a = 3$ und $b = 22$
$r = 11$, also ist $k = 3$
$G = (6,\ 16)$

Die Schlüssel:
$s \in [1, 10]$: $s = 3$ und $W = (13,\ 2)$

Das Signieren mit ECDSA:
Es soll eine Nachricht verschlüsselt werden, deren Hashwert $f = 10$ ist. Der private Schlüssel ist $s = 3$. Das one-time key pair: $u \in [1,\ 12]$, $V = uG$.
Sei $u = 6$ dann ist $V = (x_V,\ y_V) = (21,\ 13)$.
$c = x_V \mod r = 21 \mod 11 = 10$.
$d = u^{-1}(f + sc) \mod r = 6^{-1}(10 + 3 \cdot 10) \mod 11 = 2(10 + 30) \mod 11 = 3$
Die Signatur $(c,\ d)$ ist also $(10,\ 3)$.

Das Verifizieren mit ECDSA:
Die Signatur $(c,\ d) = (10,\ 3)$ soll mit dem öffentlichen Schlüssel $W = (13,\ 2)$ verifiziert werden. Der Hashwert der Nachricht ist wieder $f = 10$.
$c = 10 \in [1,\ 10]$ und $d = 3 \in [1,\ 10]$ stimmt.
$h = d^{-1} \mod r = 3^{-1} \mod 11 = 4$.
$h_1 = fh \mod r = 10 \cdot 4 \mod 11 = 7$.
$h_2 = ch \mod r = 10 \cdot 4 \mod 11 = 7$.
$P = (x_P,\ y_P) = h_1G + h_2W = 6 \cdot (6,\ 16) + 6 \cdot (13,\ 2) =$
$(6,\ 7) + (8,\ 12) = (21,\ 13)$
$x_P \mod r = 21 \mod 11 = 10 = c$ stimmt und somit ist die Signatur $(10,\ 1)$ verifiziert.

3.2.5 ECNR

ECNR ist ein Elliptische Kurven Signatur-Primitiv, Version Nyberg-Rueppel. So, wie es in [IEE] veröffentlicht ist, basiert es auf den Arbeiten von [Mil86], [Kob87] und [NR93]. Während das Original das Wiederherstellen des Hashwerts der Nachricht bei der Verifikation erlaubt, ist die in [IEE] und hier vorgestellte Version nur so zu verwenden wie ECDSA in 3.2.4. Wichtig für die tatsächliche Verwendung in einer Bibliothek für Public Key Kryptographie ist, dassECNR patentiert ist und somit nicht frei verwendet werden darf.

ECNR und ECDSA sind sehr ähnlich. Der Parameter Satz und das Schlüsselpaar werden wie bei ECDSA gebildet. Auch die Signatur (c, d) ist ein Paar von ganzen Zahlen, das wie folgt berechnet wird:

Input EC Domain Parameter $q,\ a,\ b,\ r,\ G$ und außerdem der dazu private key s, der mit den selben Parametern erzeugt wurde, und der ganzzahlige Hashwert f der Nachricht M.

Output Die Signatur (c, d), mit $0 < c < r$ und $0 < d < r$.

Operation

1. Generiere ein *one-time key pair* $(u,\ V)$ mit den EC Domain Parametern, die schon für s verwendet wurden.
   $V = uG := (x_V,\ y_V) \neq \mathcal{O}$
2. Berechne $c = x_V + f \mod r$. Ist $c = 0$, starte erneut bei 1.
3. Berechne $d = u - sc \mod r$.
4. Die Signatur ist $(c,\ d)$.

Es fällt sofort auf, dass die ECNR-Signaturerzeugung keine Invertierung modulo r benötigt. Sonst ist diese Signaturerzeugung der von ECDSA sehr ähnlich.

Auch die Verifikation ist der von ECDSA sehr ähnlich. Der große Unterschied ist, dass die Verifikation den Hashwert der Signatur zurückgibt:

Input EC Domain Parameter q, a, b, r, G und außerdem der dazu passende public key W und die Signatur $(c,\ d)$.

Output Der ganzzahligen Hashwert f der Nachricht M oder `false`, wenn die Signatur nicht authentisch ist oder die Schlüssel nicht zu den EC Domain Parametern passen.

Operation

1 Prüfe, ob $c,\ d \in [1,\ r-1]$. Wenn nicht, return `false`.

2 Berechne $P = d \cdot G + c \cdot W$. Ist $P = \mathcal{O}$, return `false`. Sonst ist $P = (x_P,\ y_P)$.

3 Berechne die ganze Zahl $f = c - x_P \mod r$.

4 Return f, den Hashwert der Nachricht M.

Beispiel 6: ECNR

Die EC Domain Parameter:
$q = 23$
$a = 3$ und $b = 23$
$r = 11$, und $k = 3$
$G = (6,\ 16)$

Die Schlüssel:
$s \in [1, 10]$: $s = 3$ und $W = (13,\ 2)$

Das Signieren:
Es soll eine Nachricht verschlüsselt werden, deren Hashwert $f = 10$ ist. Der private Schlüssel ist $s = 3$. Das one-time key pair: $u \in [1,\ 10]$, $V = uG$.
Sei $u = 6$, dann ist $V = (x_V,\ y_V) = (21,\ 13)$.
$c = x_V + f \mod r = 21 + 10 \mod 11 = 9$.
$d = u - sc \mod r = 6 - 3 \cdot 10 \mod 11 = 1$
Die Signatur $(c,\ d)$ ist also $(9,\ 1)$.

Das Verifizieren:
Die Signatur $(c,\ d) = (10,\ 8)$ soll mit dem öffentlichen Schlüssel $W = (8,\ 11)$ verifiziert werden.
$c = 10 \in [1,\ 10]$ und $d = 8 \in [1,\ 10]$ stimmt.
$P = (x_P,\ y_P) = d \cdot G + c \cdot W = 1 \cdot (6,\ 16) + 9 \cdot (13,\ 2) =$
$(6,\ 16) + (21,\ 10) = (21,\ 13)$
$f = c - x_P \mod r = 9 - 21 \mod 11 = 10 = f$.

3.2.6 ECDH

ECDH steht für Diffie-Hellman mit Elliptischen Kurven. In [IEE] wird ECDH unter ECKAS-DH1 oder ECKAS-DH2 beschrieben. Dabei steht ECKAS für Ellip-

tic Curve Key Agreement Scheme und DH für Version Diffie-Hellman. ECKAS-DH1 und ECKAS-DH2 sind zwei Versionen dieses Schemas: Bei ECKAS-DH1 bringt jede der zwei Parteien je eine Schlüsselpaar ein, bei ECKAS-DH2 je zwei Schlüsselpaare, um einen gemeinsamen Schlüssel zu erhalten. Ziel beider Schemata ist, dass jede Partei eine Sequenz gemeinsamer geheimer Schlüssel $K_1, K_2, \ldots, K_i$ erzeugt.

Zur Ausführung stehen zwei Primitive, ECSVDP-DH und ECSVDP-DHC, zur Verfügung. ECSVDP steht für *Elliptic Curve Secret Value Primitive* DH für *version Diffie-Hellman* bzw. *version Diffie-Hellman with cofactor multiplication*. Zusätzlich müssen sich die Kommunikationspartner zwischen den zwei möglichen Primitiven ECSVDP-DH und ECSVDP-DHC entscheiden und sich auf eine *Key-Derivation function* (KDF) einigen. In [IEE] wird KDF1 vorgeschlagen, der hier nicht weiter beschrieben wird.

Wir beschreiben nun zuerst die Key-Agreement Schemata ECKAS-DH1 und ECKAS-DH2 und dann die Primitive ECSVDP-DH und ECSVDP-DHC.

ECKAS-DH1

1. Erzeuge einen EC Domain Parameter Satz, für den die Schlüsselpaare beider Parteien gültig sein sollen.
2. Wähle einen privaten Schlüssel s bezüglich der Parameter aus Schritt 1.
3. Beschaffe den öffentlichen Schlüssel W' der anderen Partei, der ebenfalls gültig ist für die Parameter aus Schritt 1.
4. *(Optional)* Ist das ausgewählte secret value derivation Primitiv ECSVDP-DHC, dann prüfe, ob W' Element der Kurvengruppe ist, die in den Parametern festgelegt ist. Sonst überprüfe, ob W' ein gültiger öffentlicher Schlüssel ist. Wenn eine der Überprüfungen fehlschlägt, stop.
5. Berechne einen gemeinsamen geheimen Wert z vom privaten Schlüssel s und dem öffentlichen Schlüssel W' mit Hilfe des ausgewählten secret value derivation Primitivs.
6. Konvertiere den geheimen gemeinsamen Wert z in einen Oktet-String Z mit der Methode *FE2OSP* (siehe [IEE]).
7. Für jeden der zu erzeugenden gemeinsamen geheimen Schlüssel gilt:
   (a) Generiere Schlüssel-Austausch Parameter P_i für den Schlüssel.

(b) Berechne einen gemeinsamen geheimen Schlüssel K_i vom Oktet-String Z und den Schlüssel-Austausch Parametern P_i mit der ausgewählten key derivation Funktion.

ECKAS-DH2

1. Erzeuge einen EC Domain Parameter Satz, für den die ersten Schlüsselpaare beider Parteien gültig sein sollen.
2. Erzeuge einen EC Domain Parameter Satz, für den die zweiten Schlüsselpaare beider Parteien gültig sein sollen.
3. Wähle gültige private Schlüssel s und u bezüglich der Parameter aus Schritt 1 beziehungsweise 2.
4. Beschaffe die öffentlichen Schlüssel W' und V' der anderen Partei, ebenfalls gültig für die Parameter aus Schritt 1, beziehungsweise 2.
5. *(Optional)* Ist das erste ausgewählte secret value derivation Primitiv ECSVDP-DHC, dann prüfe, ob W' Element der Kurvengruppe ist, die in den Parametern aus Schritt 1 festgelegt ist. Sonst überprüfe, ob W' ein gültiger öffentlicher Schlüssel ist. Wenn eine der Überprüfungen fehlschlägt, stop.
   Ist das zweite ausgewählte secret value derivation Primitiv ECSVDP-DHC, dann prüfe, ob V' Element der Kurvengruppe ist, die in den Parametern aus Schritt 2 festgelegt ist. Sonst überprüfe, ob V' ein gültiger öffentlicher Schlüssel ist. Wenn eine der Überprüfungen fehlschlägt, stop.
6. Berechne einen ersten gemeinsamen geheimen Wert z_1 vom privaten Schlüssel s und dem öffentlichen Schlüssel W' mit Hilfe des ersten ausgewählten secret value derivation Primitivs. Konvertiere den geheimen gemeinsamen Wert z in einen Oktet-String Z mit der Methode *FE2OSP* (siehe [IEE]).
7. Berechne einen zweiten gemeinsamen geheimen Wert z_1 vom privaten Schlüssel u und dem öffentlichen Schlüssel v' mit Hilfe des zweiten ausgewählten secret value derivation Primitivs. Konvertiere den geheimen gemeinsamen Wert z in einen Oktet-String Z mit der Methode *FE2OSP* (siehe [IEE]).
8. Wenn Kompatibilität mit ECKSA-DH1 gewünscht ist, und die privaten Schlüssel s und u (mit den entsprechenden Parametern) gleich sind und auch die öffentlichen Schlüssel W' und v' der anderen Partei gleich sind, dann sei $Z = Z_i$. Sonst sei $Z = Z_1||Z_2$.

9. Für jeden der zu erzeugenden gemeinsamen geheimen Schlüssel:
   (a) Generiere Schlüssel-Austausch Parameter P_i für den Schlüssel.
   (b) Berechne einen gemeisamen geheimen Schlüssel K_i vom Oktet-String Z und den Schlüssel-Austausch Parametern P_i mit der ausgewählten key derivation Funktion.

ECSVDP-DH

Input:
1. ein Satz EC Domain Parameter, assoziiert mit den Schlüsseln s und W
2. des Teilnehmers eigener privater Schlüssel s
3. der öffentliche Schlüssel des anderen Teilnehmers W'

Operation:
1. Berechne den Punkt $P = s \cdot W'$.
2. Ist $P = \mathcal{O}$, return `error` und stoppe.
3. Return $z = x_P \mod r$, die x-Koordinate von P.

Beispiel 7: ECSVDP-DH

Die EC Domain Parameter:
$q = 23$
$a = 3$ und $b = 23$
$r = 11$, und $k = 3$
$G = (6,\ 16)$

Das Schlüsselpaar:
$s \in [1, 12]$: es sei $s = 3$.
$W = s \cdot G = 3 \cdot (6,\ 16) = (13,\ 2)$
$W' = s' \cdot G = (20,\ 20)$

Der Schlüsselaustausch:

Alice multipliziert den öffentlichen Schlüssel $W' = (20,\ 20)$ von Bob mit ihrem privaten Schlüssel $s = 3$. Das Ergebnis ist $P = (21,\ 10)$. Der gemeinsame, geheime Wert ist also $z = 21 \mod 11 = 10$.

Bobs privater Schlüssel s' ist 9. Er berechnet also $s' \cdot W = 9 \cdot (13,\ 2) = (21,\ 10)$, wie erwartet ist $z = 21 \mod 11 = 10$.

ECSVDP-DHC

Input: 1. ein Satz EC Domain Parameter, assoziiert mit den Schlüsseln s und W,

2. der private Schlüssel s des ersten Teilnehmers
3. der öffentliche Schlüssel des anderen Teilnehmers W'
4. ein Hinweis, ob Kompatibilität zu ECSVDH-DH gewünscht ist

Operation: 1. Ist Kompatibilität zu ECSVDH-DH gewünscht, berechne die ganze Zahl $t = k^{-1}s \mod r$, sonst setze $t = s$.

2. Berechne den Punkt $P = kt \cdot W'$.
3. Ist $P = \mathcal{O}$, return `error` und stoppe.
4. Return $z = x_P$, die x-Koordinate von P.

Beispiel 8: ECSVDP-DH

Die EC Domain Parameter:
$q = 23$
$a = 3$ und $b = 23$
$r = 11$, und $k = 3$
$G = (6,\ 16)$

Das Schlüsselpaar:
$s \in [1, 12]$: es sei $s = 3$.
$W = s \cdot G = 3 \cdot (6,\ 16) = (13,\ 2)$
$W' = s' \cdot G = (20,\ 20)$

Der Schlüsselaustausch:

Alice berechnet $t = s \mod r = 3 \mod 11 = 3$. Dann ist $P = kt \cdot W' = 3 \cdot 3 \cdot (20,\ 20) = (8,\ 11)$. Der gemeinsame, geheime Wert ist also $z = 8 \mod 11 = 8$.

Die andere Seite:
Bobs privater Schlüssel s' ist 9. Er berechnet also $s' \cdot W = 9 \cdot (13, 2) = (8,\ 11)$, wie erwartet ist $z = 8 \mod 11 = 8$.

3.2.7 ECIES

ECIES steht für *Integrated Encryption Scheme version Elliptic Curve* und ist das einzige standardisierte Verschlüsselungsverfahren für Elliptische Kurven. Es fällt jedoch nicht direkt in die Kategorie Public-Key-Verfahren:

Die zwei Parteien, die geschützt kommunizieren wollen, tauschen mit Hilfe eines der in Abschnitt 3.2.6 vorgestellten Verfahren Schlüssel aus und erzeugen daraus einen gemeinsamen geheimen Schlüssel, wie beschrieben. Mit Hilfe dieses Schlüssels werden die Nachrichten symmetrisch verschlüsselt.

Dieses Verfahren ist von IEEE in den Standard P1363a aufgenommen worden. Jedoch ist weder für den Schlüsselaustausch noch für das Verschlüsselungsverfahren eine Codierung festgelegt. Dies macht die tatsächliche Verwendung dieser beiden Verfahren zur Zeit noch unmöglich.

Auf die weitere Beschreibung und Untersuchung von ECIES wird verzichtet, da nur das Schlüsselaustauschverfahren elliptische Kurven verwendet und dieses im Detail behandelt wird.

Kapitel 4

Anforderungen an eine flexible ECC-Bibliothek

In diesem Kapitel definieren wir die Anforderungen, die an eine Bibliothek für kryptographische Algorithmen mit elliptischen Kurven gestellt werden. Dabei wird auch die Sichtweise unterschiedlicher Anwendungsszenarien auf die Anforderungen in Betracht gezogen. In den folgenden Kapiteln zeigen wir, dass die von uns implementierte Bibliothek den Anforderungen genügt.

4.1 Anwendbarkeit mit ihren Teilaspekten

Die erste wichtige Eigenschaft einer Bibliothek ist ihre Anwendbarkeit: Wenn man davon ausgeht, dass nicht jeder Anwender ein Experte in Public-Key-Kryptographie oder gar ECC ist, muss die Bibliothek ohne großes technisches Know-How zu bedienen sein. Die Schnittstellen nach außen müssen somit die Möglichkeit bieten, auf einer abstrakten Ebene ein Schlüsselpaar oder eine Signatur zu generieren oder diese zu verifizieren. Ebenso muss der Provider die Generierung von Parametern sowohl durch OIDs als auch explizit ermöglichen. Selbst die Kenntnis der zur Sicherheit notwendigen Schlüssellänge darf nicht vorausgesetzt werden. Es wird also zusätzlich Transparenz gefordert. Diese Transparenz wird zum Teil schon durch die Verwendung der JCA sichergestellt. Weiterhin wurde die Parameterklasse um einen Konstruktor erweitert, um auch Laien die Schlüsselgenerierung zu ermöglichen. So besteht die Möglichkeit, vordefinierte Parameter anhand ihrer OID zu verwenden. Diese Klasse ist in Abschnitt 5.4.1 beschrieben.

Eine weitere wichtige Eigenschaft ist die Integrierbarkeit in andere Anwendungen. Ein email-Klient ist ein gutes Beispiel. Es wäre unzumutbar einen Brief auf einem Editor zu schreiben, mit einem eigenen Programmcode dieses Dokument zu signieren, um dann den Brief samt Signatur nach Outlook zu importieren, um ihn dann letztendlich abschicken zu können. Die Verifikation wäre mit ähnlichem Aufwand verbunden. Vielmehr möchte man, dass die email-Anwendung die Bibliothek selbst verwendet, um emails zu signieren oder zu verifizieren. Einen *Proof-of-Concept* für den Provider hat Markus Tak mit seinem FlexiS/MIME Outlook-Plugin [Tak] gegeben. Mit dieser Software und dem FlexiProvider ist es möglich, mit jedem implementierten Algorithmus Zertifikate zu erstellen, emails zu signieren und Signaturen zu verifizieren.

Das Outlook-Plugin ist eine Beispielanwendung für die Windows-Plattform. Ein weiteres Beispiel wäre ein Plugin für Netscape, Konqueror, etc. Da der Provider selbst in Java implementiert und daher plattformunabhängig ist, kann der Provider selbst sehr gut PlugIns für die verschiedenen Klienten bedienen.

Der letzte Aspekt, der zur Anwendbarkeit gehört, ist die Standard-Konformität. Im letzten Kapitel haben wir einige Standards kennengelernt. Sie stellen Regeln auf, an die sich der Entwickler halten muss, um die Interoperabilität mit anderen Bibliotheken zu ermöglichen. Das beginnt bei den Algorithmen und endet bei der Repräsentation der Codierung, der Signatur, der Schlüssel und der Cipher. Wiederholter Zertifikatsaustausch mit der Firma Secude sowie mit CryptoVision hat gezeigt, dass auch diese Anforderung vom FlexiProvider, insbesondere dem FlexiECProvider, erfüllt wird.

4.2 Sicherheit

Gerade für eine Krypto-Bibliothek ist die Sicherheit eine der wichtigsten Anforderungen überhaupt. Doch was bedeutet „Sicherheit“ überhaupt?

Wünschenswert ist ein Verfahren, das *beweisbar* sicher ist. Dies existiert in der Public-Key-Kryptographie jedoch nicht. Man kann nur sagen, dass nach dem heutigen Stand der Forschung dieses oder jenes Problem nicht lösbar ist. So gilt beispielsweise das RSA-Verfahren mit dem Faktorisierungsproblem und einem 1024-Bit Modul als sicher, das BSI (Bundesamt für Sicherheit in der Informationstechnik) jedoch würde ganz klar aus Sicherheitsgründen ein 2048-Bit Modul vorziehen. Auch für Elliptische Kurven Kryptographie gibt es Empfehlungen vom BSI. Hier ein Auszug aus *Geeignete Kryptoalgorithmen* des BSI (Bundesamt für

Sicherheit in der Informationstechnik) [fSidI02]:

> Um die Systemparameter festzulegen, werden eine elliptische Kurve E und ein Punkt P auf $E(\mathbb{F}_p)$ erzeugt, so dass folgende Bedingungen gelten:
>
> - $\#E(\mathbb{F}_p) = a \cdot q$ mit einer von p verschiedenen Primzahl q.
> - $ord(P) = q$.
> - $r_0 := \min(r : q \text{ teilt } p^r - 1)$ ist groß, konkret $r_0 < 10^4$.
> - Die Klassenzahl der Hauptordnung, die zum Endomorphismenring von E gehört, ist mindestens 200.
>
> Für den Zeitraum bis Ende 2007 soll die Bitlänge von p mindestens 192 betragen. Dabei sollte q sich nur um einen kleinen Faktor von p unterscheiden. Auf jeden Fall ist sicherzustellen, dass die Bitlänge von q mindestens 180 Bit beträgt. Für den Zeitraum bis Ende 2005 reicht - ohne weitere Bedingung an die Bitlänge von p - eine Minimallänge des Parameters q von 160 Bit aus.

Lenstra und Verheul haben in [LV99] eine Abschätzung der Entwicklung von Schlüssellängen ausgearbeitet. In einer Tabelle stellen sie die als sicher zu erwartenden Schlüssellängen für symmetrische, RSA- und EC-Schlüssel für die nächsten Jahre bis 2050 zusammen. Aus dieser Tabelle kann man ablesen, dass für das Jahr 2003 die EC-Schlüssellänge von 140 Bit genügt. Für die nächsten 20 Jahre, bis 2023, sei man mit 197 Bit auf der sicheren Seite und bis zum Jahr 2050 reichten 272 Bit.

Die Vergangenheit hat jedoch gezeigt, dass Voraussagen über die Technologieentwicklung sehr schwierig sind. Noch vor 30 Jahren hat sicher niemand daran geglaubt, dass jeder dritte Haushalt einen PC zu Hause stehen haben würde, schon garnicht mit einem Arbeitspeicher von 256 MB und Festplatten im zehn-fachen Giga-Bereich. Ebensowenig ist abzuschätzen, wie die Algorithmen-Entwicklung voran schreitet.

Spätestens dann, wenn die Quantencomputer einsatzfähig werden, muss die Kryptographie auf ganz neuen Techniken beruhen, da die hohe Rechenleistung der neuen Rechnertechnologie jedes der RSA- und EC-Verfahren wird brechen können.

Um zumindest ein gewisses Maß an Sicherheit in Sachen Algorithmen sicherzustellen, ist es ratsam, nur Algorithmen zu verwenden, die weitläufig bekannt und

standardisiert sind und deren zugrunde liegendes Problem auch mathematisch interessant ist. Dann ist die Wahrscheinlichkeit, dass Fortschritte im Berechnen oder Brechen eines Verfahrens bekannt werden, erheblich höher.

4.3 Erweiterbarkeit

Für den Anwender der Bibliothek auf den ersten Blick nicht so auffällig aber dennoch wichtig ist die Möglichkeit der Erweiterung. So wird beispielsweise ein neuer kryptographischer Algorithmus in einen Standard aufgenommen oder einfach nur ein neuer Körpertyp, der von der Krypto-Bibliothek unterstützt werden soll. Oder man muss aufgrund einer festgestellten Sicherheitslücke Änderungen vornehmen. Dies kommt recht häufig vor. Das bekannteste Beispiel ist wohl DES, das durch das AES abgelöst wurde. Es gibt keinen Grund, DES aus der Bibliothek zu entfernen, da es schließlich in der Verantwortung jedes Anwenders liegt, ob er DES weiter verwendet oder nicht. Aber die Bibliothek musste um AES erweitert werden, da dies zum neuen Standardverfahren ernannt wurde.

Auch in der Elliptische Kurven Kryptographie gibt es Neuerungen. So werden die *Optimalen Erweiterungskörper* (OEF für Optimal Extension Field) in IEEE's 1363 mit aufgenommen. Auch um weitere kryptographische Primitive, wie ECIES, wurde der Standard Schritt für Schritt erweitert.

Es ist daher überaus wichtig, dass die Bibliothek flexibel gebaut ist, so dass sie um einige Module gekürzt und um andere erweitert werden kann, um den Anforderungen der Standards und der Industrie gerecht zu werden.

4.4 Effizienz

Unter Effizienz in der Kryptographie versteht man im allgemeinen, wie schnell eine Maschine oder eine Anwendung, ein Programm, ist. Das ist aber ein Ausdruck, mit dem man aus dem Kontext gerissen selten etwas anfangen kann. Das liegt daran, dass unterschiedliche Anwendungen unterschiedliche Erwartungen an die Bibliothek haben. Die Anforderung „Effizienz“ kann also nur aus Sicht von Anwendungen und Anwendern definiert werden.

Die Effizienz ist in dieser Arbeit der Schwerpunkt. Sie ist das Thema der Kapitel 6, 9, 10 und 11 und wird darin noch oft aufgegriffen werden.

Um ein repräsentatives Bild zu erhalten, werden in diesem Abschnitt vier verschiedene Anwendungszenarien vorgestellt und deren Effizienz-Anforderungen definiert. Die Frage ob und wie diese erfüllt werden, wird in Kapitel 11 erläutert und anhand von experimentellen Ergebnissen belegt.

4.4.1 Signieren und Verifizieren von signierten Emails

Das Surfen im Netz und das Versenden von Emails sind heute wohl die am weitesten verbreiteten Anwendungen auf handelsüblichen Rechnern, die schon in jedem dritten Haushalt schon zu finden sind. Nicht jeder nutzt im Privatgebrauch die Möglichkeit seine Emails zu signieren oder zu verschlüsseln. Doch gerade bei geschäftlichen Adressaten und Inhalten sollte darauf nicht mehr länger verzichtet werden. Da der meiste schriftliche Verkehr elektronisch erfolgt, ist es umso wichtiger, dass der Mehraufwand durch Signieren und Verfizieren nicht schmerzhaft spürbar wird.

Bei einer Email-Anwendung wie Outlook von Microsoft oder Netscape von SUN muss zur Zeit, die das Signieren oder Verifizieren benötigt, noch jene für das Laden der Nachricht, des Zertifikats und oder des privaten Schlüssels dazu addieren. Auf diese Zeiten kann in dieser Arbeit kein Einfluss genommen werden. Daher beschränken wir uns auf die Forderung, dass das Signieren oder das Verifizieren zeitlich für den Anwender nicht bemerkbar sein darf. Das „Gefühl der direkten, unmittelbaren Reaktion“ [SH02, Folie 28] oder [Nie94, Kapitel 5.5] stellt sich ein, wenn 100 ms nicht überschritten werden.

Wir werden sehen, dass uns dies bei Anwendungen auf einem PC, nicht aber auf einem mobilen Gerät gelingt.

Bei PCs sind heute ein Platten mit Festplattenkapazität von 3 - 5 GB durchaus schon Standard. Es stellt daher kein Problem dar, zusätzlichen Plattenbedarf von 50 KB für eventuelle vorberechnete Werte zu belegen, die das Signieren und Verifizieren beschleunigen können. Die Vorlaufzeit dagegen sollte auf ein Minimum begrenzt werden, da man nach dem Booten des Rechners nicht noch lange warten will, ehe man die Emails lesen kann.

4.4.2 Gesicherte Verbindung zwischen Online-Server und mobilem Gerät

Sehr viele Handys und PDAs bieten heute neben ihren eigentlichen Aufgaben ebenso kabellose Verbindungen ins Netz und zu verschiedenen Servern. Diese müssen gegen Fälschungen und Abhören abgesichert werden.

In dieser Arbeit wird folgendes Fallbeispiel auf einem iPAQ-PocketPC [Com02] von Compaq betrachtet. Es ist übertragbar auf jedes andere mobile Gerät, das Netzanbindung unterstützt, und auf dem zusätzliche Anwendungen installierbar sind.

Das Szenario: Ein Kunde am Bahnhof möchte verreisen und benötigt dafür ein Ticket. Über eine spontane Vernetzung seines PDAs mit dem Bahnserver schickt er eine Ticket-Anfrage an diesen, bekommt den Preis mitgeteilt und kauft das Ticket mit digitalen Münzen. Dabei authentifizieren sich die beiden Parteien mit Hilfe von digitalen Signaturen, die Kommunikation wird symmetrisch mit AES verschlüsselt.

Diese Anwendungen wurden alle von unterschiedlichen Gruppen im Rahmen eines DFG-Projekts implementiert und auf der CeBIT 2003 vorgestellt. Diese iPAQ-Anwendung umfasst viele kleinere Teil-Projekte, die hier nicht alle erwähnt werden. Für mehr Details siehe [tMWM03] und [WtMJM03]. Der Teil, der für diese Arbeit relevant ist, sind der Schlüsselaustausch und das Signieren und Verifizieren mit elliptischen Kurven auf diesem Gerät.

Für dieses Projekt wurde der iPAQ-PocketPC als Plattform gewählt, da für ihn Linux-Systeme und Java VM existieren und er eine Bluetooth Schnittstelle besitzt. Er hat einen 206 MHz Intel Strong ARM 32-Bit-Prozessor mit 64 MB RAM und 32 MB Flash-ROM [Com02].

Für dieses Gerät werden andere Effizienz-Anforderungen gestellt: Während auf einem herkömmlichen Rechner eine „echte Wartezeit“ von einer Sekunde schon als echte Zumutung empfunden wird, ist der Standard bei mobilen Geräten noch nicht so hoch geschraubt. Eine Sekunde ist noch völlig akzeptabel, mehr als zwei Sekunden stellen aber auch auf einem mobilen Gerät eine Zumutung dar.

Auf einem mobilen Gerät ist erheblich weniger Speicherplatz vorhanden als auf einem PC. Der größte Teil wird schon vom Betriebssystem in Anspruch genommen. Daher sollte sich der Speicherplatz für zusätzliche signaturbeschleunigende Daten, sich auf höchstens 50 KB beschränken. In unseren Fallbeispielen und unseren Experimenten in Kapitel 10 werden sogar nur 50 KB verwendet werden und

es wird sich zeigen, dass auch mit diesem begrenzten Platz sehr gut Ergebnisse zu erzielen sind.

4.4.3 Postwertzeichen

Seit einiger Zeit besteht die Möglichkeit, Postwertzeichen mittels Barcode digital signiert auf Umschläge zu drucken. Ein solches Postwertzeichen ist dann gleichbedeutend mit einer aufgeklebten Briefmarke, d.h. es bedeutet, dass die Gebühr bezahlt wurde. Um sicher zu gehen, dass das Zeichen auch tatsächlich von der Post stammt und auch bezahlt wurde, muss die Signatur überprüft werden. Erwartet wird, dass etwa 4 Millionen Umschläge dieser Art pro Tag durch die Maschinerie laufen. Das sind 47 Signaturen und Verifikationen in der Sekunde. Es wird davon ausgegangen, dass für jede dieser Aufgaben eine Maschine zur Verfügung steht. Das heißt, dass für jede der Aufgaben je 22 ms zur Verfügung stehen. Wir werden in Kapitel 11 zeigen, dass diese Zeiten erreicht werden.

4.4.4 Webserver für Online-Banking

Ein Webserver für Online-Banking ist eine weitere Hochleistungsanwendung. Zum Beispiel werden bei der Dresdner Bank [Küh03] morgens auf dem internen Server für Firmenkunden 1600 Anfragen und mittags 800 - 1000 Anfragen pro 15 Minuten gestellt. In der Früh werden am Privatkundenserver sogar 200 Anfragen pro Sekunde gestellt! Das bedeutet, dass für eine Anfrage nur 5 ms Zeit sind. Dafür stehen mehrere Maschinen zur Verfügung. Die Berechnungen werden mit spezieller Hardware ausgeführt.

Kapitel 5

Design und Implementierung der ECC-Bibliothek

Im vorigen Kapitel haben wir gesehen, dass neben der Effizienz und der Sicherheit auch die Integrierbarkeit, die Erweiterbarkeit, die Plattformunabhängigkeit und die Transparenz eine wichtige Rolle spielen. Diese Anforderungen können mit einem entsprechenden Design der Bibliothek erfüllt werden, wir beschreiben dieses Design in diesem Kapitel. In Abschnitt 5.1 wird ein Blick aus der Vogelperspektive auf die Bibliothek geworfen. Aus diesem Bild lassen sich bereits die Flexibilität, die Integrierbarkeit und die Erweiterbarkeit ersehen. In Abschnitt 5.2 und 5.4 wird dieses Bild im Detail beleuchtet und anschließend in 5.5 diskutiert.

5.1 Unterteilung in Arithmetik und Mechanismen

Bild 5.1, Seite 54, zeigt schematisch, wie der Provider gegliedert ist. Jedes der einzelnen Rechtecke, mit `Truetype` beschriftet, stellt ein Paket dar: Der FlexiECProvider liegt unterhalb des Pakets `de.flexiprovider.ec`. In diesem Verzeichnis liegen die Klassen zu den Kryptoverfahren: `ECDSASignature`, `ECNRSignature` und `ECDH`.

Eine Stufe weiter folgt das Paket `ecparameters`, das die Klassen `ECParameters` und `ECParameterGenerator` enthält. Es enthält im Unterpaket `spec` zusätzlich die Klasse zur Parameterspezifikation, `ECParameterSpec`.

Ein weiteres Paket, `eckeys`, enthält alle Klassen die Schlüssel betreffend. Al-

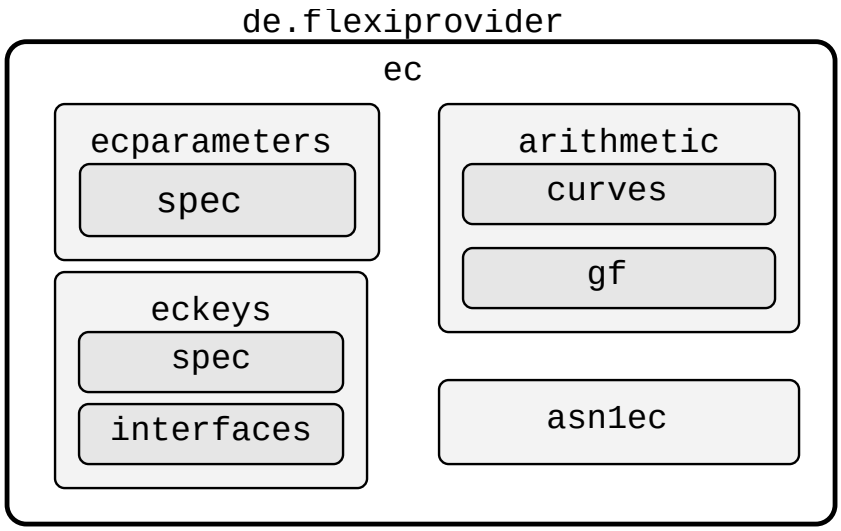


Abbildung 5.1: Der Provider aus der Vogelperspektive

le Schlüssel haben dieselbe Form und werden auf die gleiche Weise erzeugt (siehe Abschnitt 3.2.3). Daher existiert nur je eine Klasse zur Schlüsselpaarerzeugung: `ECKeyPairGenerator`, zur Schlüsselpaar-Konvertierung, `ECKeyFactory`, sowie für den privaten, öffentlichen und geheimen Schlüssel `ECPrivateKey`, `ECPublicKey` und `ECSecretKey`. Auch hier gibt es Spezifikationen in `spec` und zusätzlich die von der JCA verlangten Interfaces in `interfaces`.

Die Kryptoverfahren sind getrennt von der EC-Arithmetik implementiert. Dadurch wird die Erweiterung des Kryptoteils ermöglicht, ohne dass die Arithmetik abgeändert werden muss. Diese ist im Paket `arithmetic` untergebracht und wiederum in die EC-Arithmetic im Paket `curves`, und die Körperarithmetik im Paket `gf` unterteilt. Letztere ist teilt sich in Unterpakete für Körperarithmetik über Primkörper und endliche Körper der Charakteristik 2. Sie wird in dieser Arbeit nicht näher betrachtet. Die Struktur der EC-Arithmetik wird im nächsten Abschnitt beschrieben.

Einen Sonderposten bildet zu guter Letzt das Paket `asn1ec`. In ihm sind die Klassen abgelegt, die zur ASN.1-Codierung der Schlüssel und Signaturverfahren verantwortlich sind. Zusammen mit dem Codec-Paket, das von der SeMoA-Gruppe [SeM] der Frauenhofer Gesellschaft Darmstadt entworfen wurde, der Klasse `ECKeyFactory` und den Klassen des Pakets `asn1ec`, ist der FlexiECProvider im Stande, Provider-fremde Signaturen und Schlüssel zu lesen und mit ihnen zu arbeiten, sofern sie ASN.1-codiert sind. Da diese Pakete und Klassen nicht Teil

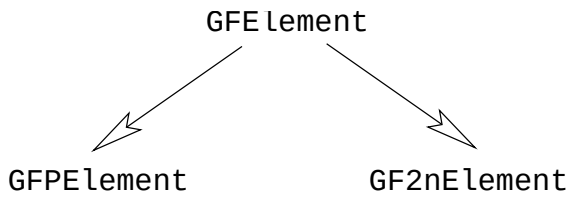


Abbildung 5.2: Design der Körperarithmetik

dieser Arbeit sind, wird auf sie nicht weiter eingegangen.

5.2 Körper-Arithmetik

Die gesamte Arithmetik wurde ebenfalls so entwickelt, dass sie jederzeit erweitert werden kann, ohne dass der gesamte Provider abzuändern ist. So wurde zuerst die EC-Arithmetik über $\mathbb{F}_p$ entwickelt, um sie erst dann um die Arithmetik über $\mathbb{F}_{2^m}$ zu erweitern. Ebenso besteht die Möglichkeit, die Arithmetik um die über Körpern anderer Art zu ergänzen, zum Beispiel über die der optimalen Erweiterungskörper. Die Vererbungshierarchie der bisherigen Körperarithmetik ist dem Bild 5.3 zu entnehmen. Wir beschränken uns hier auf die Beschreibung bezüglich der Primkörper.

Die drei Grundoperationen Addieren, Subtrahieren und Multiplizieren werden durch die BigInteger-Methoden `BigInteger add(BigInteger o)`, `BigInteger subtract(BigInteger o)` und `BigInteger multiply(BigInteger o)` modelliert. Eine Quadrierung wird ebenfalls durch die Methode `BigInteger multiply(BigInteger o)` ausgeführt. Die Reduktion, die bei jeder Körperoperation in $\mathbb{F}_p$ implizit ausgeführt wird, kann durch die Operationen `BigInteger mod(BigInteger modul)` oder `BigInteger remainder(BigInteger modul` verwendet werden. Der Unterschied besteht darin, dass das Ergebnis der Methode `mod` immer nicht-negativ ist, während das andere Ergebnis kleiner Null sein kann. Die Laufzeit beider Methoden ist dieselbe. Weiter kann mit der Methode `BigInteger modInverse(BigInteger modul)` eine Invertierung modulo einer ganzen, nicht negativen Zahl ausgeführt werden.

Um unnötigen Rechenaufwand zu vermeiden, wird nur nach jeder Multiplikation reduziert, nicht nach Additionen oder Subtraktionen.

Da Flexibilität eine Anforderung an einen Elliptische Kurven Provider ist, muss die Implementierung die Verwendung von Elliptischen Kurven über anderen end-

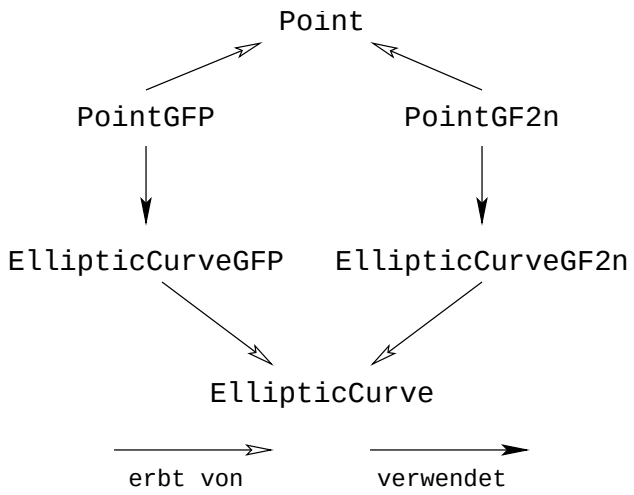


Abbildung 5.3: Design der EC-Arithmetik

lichen Körpern wie $\mathbb{F}_{2^m}$ unterstützen. Das wurde beim Design berücksichtigt. Bild 5.2 zeigt den Aufbau der Körperarithmetik. Die für Körper der Charakteristik 2 wurde, wie erwähnt, bereits implementiert.

5.3 EC-Arithmetik

Die Arithmetik mit Punkten auf einer elliptischen Kurve ist im Paket `de.flexiprovider.ec.arithmetic.curves` implementiert. Eine Übersicht der Vererbungsstruktur gibt Abbildung 5.3. Die Arithmetik setzt sich aus folgenden Klassen zusammen:

EllipticCurve: Die Form einer elliptischen Kurve ist abhängig vom zugrunde liegenden Körper. `EllipticCurve` stellt die Vaterklasse der körperspezifischen Kurven-Klassen `EllipticCurveGFP` und `EllipticCurveGF2n`. Sie enthält das einzige Attribut, das diesen gemeinsam ist: Die Größe des zugrunde liegenden Körpers $\mathbb{F}_q$, q. Die Methoden `BigInteger getQ()`, `boolean equals(EllipticCurve)` und `String toString()` sind die einzigen und als `abstract` deklariert.

EllipticCurveGFP implementiert die Funktionalität der elliptischen Kurve über $\mathbb{F}_p$ und erbt von `EllipticCurve`. Diese Klasse hält die Kurven-Parameter

a und b und implementiert die in `EllipticCurve` deklarierten Methoden, sowie `BigInteger getA()` und `BigInteger getB()`. Sie geben die Kurvenparameter a beziehungsweise b zurück.

Point implementiert alle Körper-unabhängigen Methoden, wie die einfache und mehrfache Punktmultiplikation. Es wurden unterschiedliche Techniken in unterschiedlichen Methoden implementiert, um die sie näher auf ihre Effizienz in experimentellen Tests untersuchen zu können. Diese Techniken werden in Kapitel 6 näher beschrieben. Die Methoden zur Punktaddition und -verdopplung, `Point add(Point)` und `Point multiplyBy2()`, sowie zur Punktnegierung, `Point negate()`, dem Punktvergleich, `equals(Point)` und der Ausgabe `Point toString()` werden hier abstrakt deklariert.

PointGFP Hier werden die in `Point` deklarierten Methoden mit Jacobischen Koordinaten implementiert: Die Addition, die Punktverdopplung, der Punktvergleich und die Ausgabe.

PointMixed Ebenso wie `PointGFP` implementiert die Klasse `PointMixed` die Arithmetik für Punkte über $E(\mathbb{F}_p)$. Doch im Gegensatz zur Klasse `PointGFP` sind in ihr alle fünf Koordinatensysteme realisiert, die in Abschnitt 6.4 detailliert vorgestellt werden. Sie ermöglicht es, Punktadditionen, -verdopplungen und -multiplikationen mit gemischten Koordinaten auszuführen. Die Vorteile, die daraus resultieren, werden in Kapitel 6 erläutert.

Ein Schwerpunkt dieser Arbeit liegt in der Implementierung der Klassen `Point` und `PointMixed`, da sie für die schnelle Laufzeit von Punktmultiplikationen und somit der Effizienz der kryptographischen Anwendungen verantwortlich sind. Daher gehen wir im Folgenden auf Details dieser Klassen ein.

Das Besondere an der Klasse `PointMixed` ist, dass sie alle gut bekannten Koordinatensysteme[1] in sich vereinigt. An dieser Stelle genügt es zu wissen, dass jedes von ihnen seine eigene Anzahl an Körperoperationen zur Punktaddition und -verdopplung benötigt. Dies hat Einfluss auf die Laufzeit der Addition oder Verdopplung und somit auch großen Einfluss auf die Punktmultiplikation. Mit einer Kombination dieser Systeme kann eine Punktmultiplikation erheblich beschleunigt werden. Dies wird in den Kapiteln 9 und 10 bewiesen.

[1] Diese Koordinatensysteme heißen Affines, Projektives, Jacobisches, Chudnowsky Jacobisches und Modifiziert Jacobisches Koordinatensystem. Sie stammen aus unterschiedlichen Quellen und werden in dieser Arbeit im Kapitel 6 vorgestellt.

Um diese Koordinatenkombinationen anwenden und beliebig mit ihnen experimentieren zu können, wurde folgendes Konzept entwickelt:

Zur Punktaddition und -verdopplung existiert je eine Methode, in der mit Hilfe von Parametern angegeben wird, welche mit welcher Koordinatenart das Resultat zurück gegeben werden soll. Innerhalb dieser Methode existiert ein Mechanismus der selbstständig erkennt, in welcher Ar die Summanden vorliegen. Auf diese Weise können innerhalb einer Addition drei und innerhalb einer verdopplung zwei Koordinatenarten kombiniert werden.

Dasselbe Konzept wird bei den Methoden zur Punktmultiplikation angewandt: Für jede Additions- und Verdopplungsaufruf innerhalb einer dieser Methoden existiert wieder je ein Parameter, welcher angibt, in welcher Koordinatenart das jeweilige Ergebnis wiedergegeben werden soll.

Die Koordinatensysteme sind dabei mit Integers kodiert (siehe die Klassenbeschreibung des Providers). Die Signaturen der Punktoperationsroutinen haben folgende Form:

```
Point add(Point other, int result_type)
```
und
```
Point multiplyBy2(int result_type)
```

Entsprechend diesem Muster wurden ebenfalls Konstruktoren, Methoden zur Subtraktion, Negation, zum Vergleich etc. implementiert.

Beispiel 9: Parametrisierte Methode zur Punktmultiplikation

Wir nehmen einen Algorithmus dem Kapitel 6 vorweg. Folgender Code[2] berechnet die Punktmultiplikation nach dem Algorithmus Square-and-Multiply [MvV97] (siehe auch ALG 6.1.1 auf Seite 68):

```
public Point squareMultiply(BigInteger m) {
  Point P = copy(this);
  Point H = new PointMixed(mE, true);
    return P;
    ...
  final int l = m.bitLength() - 1;
  for (int i = l; i >= 0; i--) {
```

[2] Der Code stammt aus dem FlexiECProvider, Klasse `de.flexiprovider.ec.arithmetic.curves.Point`. Er ist um einige Zeilen gekürzt, um nur den relevanten Teil zu zeigen.

```
    if (m.testBit(i)){
      H.multiplyThisBy2();
      H = P.add(H);
    }
    else
      H.multiplyThisBy2();
  }
  return H.getAffin();
}
```

In der Implementierung des FlexiProviders wurde diese Methode um die drei Parameter A B und C erweitert. Diese geben an, welche Koordinaten das Ergebnis der entsprechenden Methode haben soll.

```
public Point squareMultiply(BigInteger m,
                            int A, int B,
                            int C){
  Point P = copy(this);
  Point H = new PointMixed(mE, true);
    return P;
    ...
  final int l = m.bitLength() - 1;
  for (int i = l; i >= 0; i--) {
    if (m.testBit(i)){
      H.multiplyThisBy2(B);
      H = P.add(H, A);
    }
    else
      H.multiplyThisBy2(C);
  }
  return H.getAffin();
}
```

Mit dieser Implementierung werden die theoretischen Untersuchungen aus Kapitel 9 in die Praxis umgesetzt und getestet. Letzteres geschieht im Kapitel 10.

5.4 Kryptographische Algorithmen

Die Kryptographischen Verfahren, die auf elliptischen Kurven basieren, sind von dem zugrunde liegenden Körper unabhängig: Benötigt wird jeweils eine Methode zur Punktaddition und -multiplikation, sowie die modulare Invertierung, Multiplikation und Addition. Hierbei ist das Modul unabhängig von der Wahl des Körpers, da das Modul die Primzahlordnung r des Basispunktes G ist (siehe ECParameter in Abschnit 3.2.2).

Die Klassen für die kryptographischen Verfahren verwenden daher nur die Klasse `Point`, in der die Punktaddition deklariert ist. Durch die Vererbungsstruktur (siehe Bild 5.3 auf Seite 56) werden die Methoden der richtigen Klasse (`PointGFP` oder `PointGF2n`) aufgerufen. Das Design gewährleistet also tatsächlich die Unabhängigkeit der kryptographischen Algorithmen vom Körper.

Die Aufteilung der Klassen für die kryptographischen Algorithmen liegt ebenfalls auf der Hand: Jedes der kryptographischen Verfahren verwendet die gleichen Domain Parameter. Auch die Schlüssel sind die gleichen, sie werden auf ein und dieselbe Art generiert und codiert. Daher liegt es nahe, die Parameter und Schlüssel einmal für alle Verfahren zu implementieren. Die Aufteilung in die Pakete `ecparameters` und `keys` wird der Übersicht halber vorgenommen. Details werden im nachfolgenden Kapitel erläutert.

Die konkrete Implementierung kryptographischer Algorithmen in einem JCA-basierten Provider ist von der JCA vorgegeben: Sie enthält je ein Service Provider Interface (SPI, siehe Abschnitt 2.2) für die Signatur und Parameter, und je einen zur Schlüsselgenerierung, Parametergenerierung, etc. Diese müssen geerbt und implementiert werden, um die Funktionalität spezieller Verfahren bereitzustellen. Die Klassen und ihre Methoden sind daher festgelegt und werden hier nur in Kürze beschrieben, da man Details in [SUN] nachlesen kann.

Im Paket `de.flexiprovider.ec` liegen die Klassen, die Funktionalitäten Signaturgenerierung und -verifikation und Schlüsselaustausch bereit stellen. Das sind die Klassen `ECDSA`, `ECNR`, `ECSVDPDH`, `ECSVDPDHC` und `ECDHTools`. Dabei steht ECSVDPDHC für den Diffie-Hellman Schlüsselaustausch mit, beziehungsweise ECSVDPDH für den Diffie-Hellman Schlüsselaustausch ohne Cofaktor-Multiplikation. `ECDHTools` enthält Hilfsmethoden, die zusätzlich benötigt werden.

5.4.1 EC Domain Parameter

Die Domain Parameter werden von allen EC-basierten Verfahren verwendet. Für die volle Funktionalität werden folgende Klassen benötigt:

`ECParameters` implementiert das Interface `java.security.AlgorithmParametersSPI`. Sie ist eine *blickdichte* Repräsentation der EC Domain Parameter. In ihr sind alle in Abschnitt 3.2.2 beschriebenen Attribute enthalten.

Die Klasse besitzt mehrere Konstruktoren. Einige davon erlauben explizites Initialisieren. Das heißt, alle Parameter, die zur Erzeugung des Körpers, der Kurve, des Basispunktes, seiner Ordnung und des Cofaktors nötig sind, werden der Methode explizit mitgegeben.

Ebenso ist es möglich, ein Set von Parametern anhand seiner OID (*Object Identifier*) zu spezifizieren. Dies geschieht mit Hilfe der in [Sta00b] und [X9.99] definierten Kurven. Sie sind in der Klasse `de.flexiprovider.ec.asn1ec.PrimeCurves` (oder für den Körper $\mathbb{F}_{2^m}$ die Klasse `Char2Curves`), sortiert nach der OID, enthalten. Darüber hinaus wurde von Dr. Harald Baier eine Software zum Erzeugen kryptographisch sicherer Parameter implementiert [Bai02, Bai]. Mit ihr wurde die Klasse `PrimeCurves` um viele Kurven erweitert, die auch von folgender Klasse verwendet werden:

`ECParameterGenerator` implementiert das Interface `java.security.AlgorithmParameterGeneratorSpi`. Sie wird zur Generierung einer Parametermenge vom Typ `ECParameters` verwendet. `AlgorithmParameterGeneratorSpi` gibt vor, dass eine Initialisierungmethode der Art `engineInit(int size, SecureRandom random)` implementiert werden muss. Bei RSA bezeichnet `size` die Größe des RSA-Moduls, der gleichzeitig ein Maß der Sicherheit ist. Die Generierung von EC Parameter ist jedoch ein sehr komplexer Vorgang, der erstens sehr aufwendig ist und zweitens mehr Angaben als eine einzelne Zahl benötigt [Bai02]. Daher wird folgendes angeboten:
Für jede zehnte Bitlänge $100 \leq n \leq 500$ gibt es einen Satz Parameter, deren Ordnung des Basispunktes die Länge von n Bits hat. Beim Aufruf der Methode `engineInit(int size, SecureRandom random)` wird dann der Parametersatz verwendet, dessen Ordnung die gleiche oder die nächsthöhere Bitlänge besitzt.

Mit einer ganzen Zahl allein kann man nicht ausdrücken, über wel-

chem Körper die Kurve definiert sein soll. Um dieses Problem zu umschiffen, wurden zwei neue Klassen implementiert, `ECGFPParameterGenerator` und `ECGF2nParameterGenerator`, die wiederum von `ECParameterGenerator` ableiten. `ECGF2nParameterGenerator` generiert Kurven über dem endlichen Körper $\mathbb{F}_{2^m}$, `ECGFPParameterGenerator` über $\mathbb{F}_p$. Letzteres ist die Standardeinstellung in der Vaterklasse.

`spec.ECParameterSpec` implementiert die Klassen `java.security.spec.AlgorithmParameterSpec` und `interfaces.ECParamsInterface`. Im Gegensatz zu `ECParameters` ist diese Klasse nicht *opaque*, das heißt, dass die Klasse mit ihren Attributen offen ist. Mit dieser Klasse wird im Allgemeinen gearbeitet. Sie besitzt die gleichen Konstruktoren wie `ECParameters`.

`Interface.ECParamsInterface` wird von der JCA benötigt.

5.4.2 Schlüssel

Auch die Schlüssel sind für alle EC-Verfahren dieselben und werden auf gleiche Weise generiert. Alle benötigten Klassen liegen im Paket `de.flexiprovider.ec.eckeys`, deren Spezifikationen im Unterverzeichnis `spec` und deren Interfaces im Unterverzeichnis `interfaces` liegen.

`ECPrivateKey` ist das Interface zu einem EC privaten Schlüssel. Diese Klasse enthält den Schlüssel s, und eine Instanz der dazugehörigen `ECParameterSpec`.

`ECPublicKey` ist das Interface zu einem EC öffentlichen Schlüssel. Diese Klasse enthält den Schlüssel W und eine Instanz der dazugehörigen `ECParameterSpec`. `ECPrivateKey` und `ECPublicKey` erben von der Klasse `ECKeyInterface`, die von der JCA benötigt wird.

`ECSecretKey` ist wie `ECPrivateKey` ein Interface zu einem EC privaten Schlüssel. Diese Klasse enthält den Schlüssel s und eine Instanz der dazugehörigen `ECParameterSpec`. Diese Klasse wird von den Klassen `ECSVDPDH` und `ECSVDPDHC` zum Schlüsselaustausch benötigt.

`ECKeyFactory` wird verwendet, um Schlüssel vom Typ `Key` in eine Schlüsselspezifikation zu konvertieren und vice versa. Diese Klasse unterstützt hierbei die ASN.1/DER-Codierung.

`ECKeyPairGenerator` generiert Schlüsselpaare. Ebenso wie bei der Klasse `ECParameterGenerator` wird die Initialisierungsmethode `engineInit(int size, SecureRandom random)` angeboten. Es wird an dieser Stelle genauso verfahren: Die passenden Parameter werden aus der Klasse `PrimeCurves` ausgelesen und verwendet. Und wie bei den Parametergeneratoren bieten auch hier die Klassen `ECGF2nKeyPairGenerator` und `ECGFPKeyPairGenerator` die Möglichkeit Schlüssel und damit Parameter über $\mathbb{F}_{2^m}$ sowie über $\mathbb{F}_p$ zu erzeugen.

5.4.3 ASN.1 Codierung

Das Paket `de.flexiprovider.ec.asn1ec` enthält alle Klassen, die zur ASN.1 Codierung von den EC Parametern, den öffentlichen und den privaten Schlüsseln und den Signaturen benötigt werden. Wir gehen hier nur kurz auf sie ein.

`ASN1ECParameters` repräsentiert ein Parameterset, wie es in [X9.99] definiert ist.

`ASN1ECPrivateKey` repräsentiert einen privaten Schlüssel, wie er in [X9.99] definiert ist.

`Curve` repräsentiert eine Kurve, wie sie in [X9.99] definiert ist.

`PrimeCurves` enthält eine große Anzahl an kryptographisch sicheren Parametern. Sie setzen sich zusammen aus denen, die in [Sta00b] und [X9.99] definiert sind, und aus solchen, die von der Arbeitsgruppe generiert wurden.

`ECDSASigValue` **und** `ECNRSigValue` repräsentiert eine ECDSA- beziehungsweise eine ECNR-Signatur, wie sie in [X9.99] definiert ist.

`PrimeField` repräsentiert einen Primkörper, wie er in [X9.99] definiert ist.

Diese Klassen verwenden das von der Gruppe SeMOA implementierte Paket *codec* [SeM].

5.4.4 Hardware

In Zusammenarbeit mit Markus Ernst, Fachgebiet Integrierte Schaltungen und Systeme, wurde ein Kryptoprozessor zur Skalarmultiplikation von Punkten über

Primkörpern implementiert. Diese Realisierung ist in erster Linie nur für Kurven über $\mathbb{F}_p$ geeignet. Die Bitlänge der Gruppenordnung r und der Körpergröße p ist zwischen 160 bis 512 Bit skalierbar. Innerhalb der Menge der Domain Parameter über $\mathbb{F}_p$ bietet auch diese Realisierung große Flexibilität. Sie ist nicht für spezielle Anwendungen optimiert worden. Vielmehr soll sie für alle kryptographischen Verfahren eingesetzt werden. Da es sich um eine FPGA-Implementierung handelt, kann sie mit dem entsprechenden Softwarepaket für die Arithmetik über anderen Körpern umgeschrieben werden. Mit dieser Realisierung bleibt die Flexibilität daher erhalten. Die Eingliederung des Kryptoprozessors in den Provider wurde in Form einer Studienarbeit von den zwei Studenten Nima Barraci und Sven Becker vorgenommen und wird in Kürze unter [BB03] zu finden sein.

Der Prozessor selber wird mit Hilfe eine DLL, die `ecc_gfp.dll` angesprochen (siehe Abbildung 5.4). Da sie in C implementiert ist, existiert eine *Java Native Interface* (JNI) Schnittstelle, das `ECPInterface.java` zu der C-Bibliothek `ecpinterface.dll`. Für das Verständnis der JNI sei auf einschlägige Literatur verwiesen (zum Beispiel [SM98a] und [Gor98]. Das `ECPInterface.java` und die `ecpinterface.dll` koordiniert die Kommunikation zwischen Provider und Hardware.

Der Kryptoprozessor ist in der Lage, Punkte zu addieren, sie voneinander zu subtrahieren und eine Punktmultiplikation auszuführen. Die Aufrufe geschehen in der jeweiligen Methode in der Klasse `PointGFP` für die Addition und Subtraktion, beziehungsweise in der Klasse `Point` für die Multiplikation. Es wird bei jedem dieser Aufrufe geprüft, ob die Karte vorhanden ist. Ist das der Fall, wird der Aufruf über die JNI-Klasse `PointGFPHardware`, das `ECPInterface.java` und die `ecpinterface.dll` an die `ecc_gfp.dll` weitergeleitet. Dort besteht die Option, die Multiplikation mit der C-Implementierung[3] zu berechnen, die zu Validierungszwecken entwickelt wurde. Andernfalls wird die Punktmultiplikation im Provider ausgeführt. Somit bleibt die Transparenz auch an dieser Stelle voll erhalten.

5.5 Diskussion

Das Design erlaubt, wie man schon an der Abbildung 5.1 erkennen kann, größtmögliche Flexibilität. Es ist möglich, den Provider um weitere Körperarten sowie kryptographische Algorithmen zu erweitern oder letztere auszutauschen, ohne grundsätzlich in die Struktur einzugreifen. Durch das Verwenden der JCA wird

[3] Diese Implementierung verwendet die Bibliothek NTL.

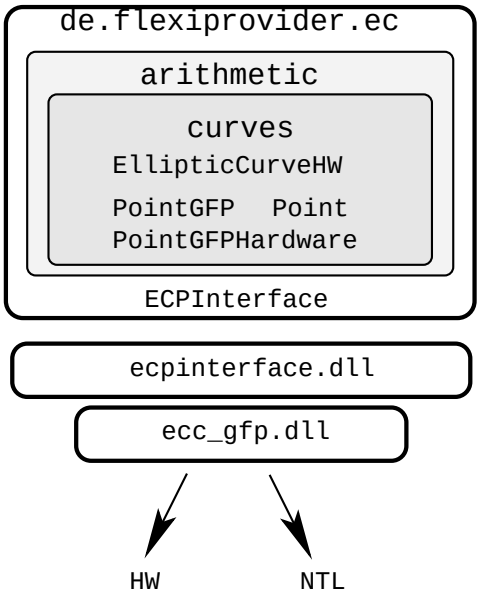


Abbildung 5.4: Eingliederung der Hardware in den FlexiECProvider

auch dem nicht-versierten Anwender die Möglichkeit gegeben, mit Hilfe des Providers einfach kryptographische Funktionalität in seine Applikationen zu integrieren.

Durch die Standardkonformität wird auch die Integrierung in andere Anwendungen wie zum Beispiel Outlook durch das FlexiSMIME-Plugin ermöglicht. Es bleibt daher noch zu zeigen, dass der Provider der Anforderung der Effizienz genügt. Wie das erreicht wird und dass es erreicht wird, ist in den Kapiteln 6 bis 11 nachzulesen.

Kapitel 6

Effizienz

Die Arithmetik der elliptischen Kurven lässt sich in zwei Teile teilen. Der erste Teil ist die Darstellung der Kurve und ihrer Punkte, die Formeln für die Punktaddition und die Punktverdopplung. Sie hängen von dem zugrundeliegenden Körper ab. Der zweite Teil beinhaltet Algorithmen zum schnellen, wiederholten Addieren von Punkten. Diese Operation $P + P + \cdots + P + P$, wo P k-mal auf sich selbst addiert wird, wird *Punktmultiplikation* genannt und entsprechend dazu durch $k \cdot P$ bezeichnet. k heißt *Exponent* in Anlehnung an die Exponentiation. Die Punktmultiplikation ist unabhängig von der Berechnung einer Punktaddition und auch vom gewählten Koordinatensystem. Viele Algorithmen zur schnellen Exponentiation in einer multiplikativen Gruppe lassen sich auf die additive Gruppe der Punkte übertragen. In folgenden Abschnitten werden diese Algorithmen vorgestellt, die auch implementiert und experimentell untersucht wurden.

Im Allgemeinen versteht man unter einer Punktmultiplikation $k \cdot P$ das k-malige Aufaddieren von P. Während einer Signaturverifikation muss aber auch eine Operation der Form $k \cdot P + l \cdot Q$ ausgeführt werden. Sie wird in dieser Arbeit mit mehrfacher Punktmultiplikation bezeichnet. Man kann dies sukzessive tun, das heißt, man berechnet zuerst $k \cdot P$, dann $l \cdot Q$ und addiert schließlich das Ergebnis. Es gibt aber auch die Möglichkeit, diese Operationen in einem Algorithmus zu verbinden und EC-Operationen zu sparen. In Abschnitt 6.1 werden die Algorithmen für die einfache Punktmultiplikation vorgestellt, und im darauffolgenden Abschnitt 6.2 die Algorithmen für die mehrfache Punktmultiplikation.

Im Weiteren werden unterschiedliche Koordinatensysteme auf ihre Komplexität untersucht. Es wird dabei deutlich, dass die optimale Wahl nicht leicht zu erkennen ist. Wir geben daher eine Strategie an, wie die optimale Kombination von

Punktmultiplikationsmethode und Koordinatensystem gefunden werden kann. Diese Kombination kann von Plattform zu Plattform unterschiedlich sein. Wir wenden diese Strategie unter Linux mit Java Version „1.4.1_02“ auf einem AMD Athlon mit 1.3 GHz und 256 MB Arbeitsspeicher an und präsentieren die Ergebnisse.

6.1 Einfache Punktmultiplikation

Die meisten Algorithmen zur schnellen Punktmultiplikation berechnen zuerst eine Menge von Punkten vor, die während der Hauptrechnung immer wieder verwendet werden. Diese Vorberechnung wird im Folgenden immer als solche markiert und zu erkennen sein. Sie ist, abhängig vom Szenario, wichtig für die Effizienzdiskussion. Die Namen der Algorithmen lassen oft erkennen, dass die originale ursprünglich zur schnellen Exponentiation entwickelt wurden. Trotzdem wird immer die Punktmultiplikation gemeint sein.

Im weiteren sei P ein Punkt auf der Kurve E über dem Körper $\mathbb{F}_p$ und k der n-Bit lange Exponent mit der Binärdarstellung $k = \sum_{i=0,\ldots,n-1} k_i 2^i$ mit $k_i \in \{0, 1\}$. Die drei ersten Methoden sind in [MvV97, Seite 615ff] nachzulesen.

6.1.1 Square-and-Multiply

Diese Methode [MvV97, S.615, Alg. 14.79] ist wohl die einfachste und bekannteste, wenn auch nicht die schnellste Methode. Sie starten mit dem Punkt im Unendlichen $\mathcal{O}$ und scannt k bitweise von links nach rechts. Für jeden neuen Index i verdoppelt sie den bisherigen Ergebnispunkt, und für jedes $k_i \neq 0$ addiert sie P dazu:

ALG **6.1.1: Square-and-Multiply**:

INPUT: k, P
OUTPUT: $k \cdot P$

```
R ← O
for i = n − 1 down to 0
  R ← 2 · R
  if k_i = 1 then
    R ← R + P
return R
```

Dieser Algorithmus benötigt immer $n-1$ Punktverdopplungen und $\mathrm{w}(k)$ Punktadditionen. $\mathrm{w}(k)$ bezeichnet die Anzahl der $k_i = 1$ und wird auch die *Gewichtung* von k genannt.

Komplexität:
Vorberechnung: -
Speicher: -
Hauptrechnung: $n/2$ ADD + $n-1$ DBL

6.1.2 Fixed-size-sliding-window Exponentiation

Die Fixed-size sliding window Exponentiation Methode [MvV97, S. 616] betrachtet jeweils w Bit des Exponenten k auf einmal. w wird *Fenstergröße* genannt, da man auf den Exponenten schaut wie durch ein Fenster der Breite w. Diese w Bit werden auf einmal abgearbeitet, danach wird das Fenster um seine Breite weiter geschoben. Durch die Abarbeitung von w Bits auf einmal benötigt diese Methode eine Vorberechnung von $2^w - 1$ Punkten.

ALG **6.1.2: Fixed-size sliding window Exponentiation**:

Vorberechnung:
INPUT: w, P
OUTPUT: $P_i = i \cdot P$ mit $0 \leq i < 2^w$

$P_0 \leftarrow P$
for $i = 1$ to $2^{w-1} - 1$
  $j = 2 \cdot i$
  $P_{j-1} \leftarrow 2 \cdot P_{i-1}$
  $P_j \leftarrow P_{i-1} + P_i$

Hauptrechnung:
INPUT: k, P, $P_i = i \cdot P$ mit $0 \leq i < 2^w$
OUTPUT: $k \cdot P$

$R \leftarrow \mathcal{O}$, $i \leftarrow n-1$
$t = (i+1) \% w$

Fortsetzung ALG **6.1.2**:

```
if (t ≠ 0)
  if (k_{n-1}k_{n-2}...k_{t+1})_2 ≠ 0 then
    R ← R + P_{(k_{n-1}k_{n-2}...k_{t+1})_2},
    i ← i − t
while i ≥ 0
  for j = 0 to j ≤ w − 1
    R ← 2 · R
  if (k_i k_{i-1} ... k_{i-w+2} k_{i-w+1})_2 ≠ 0 then
    R ← R + P_{(k_i k_{i-1}...k_{i-w+2}k_{i-w+1})_2}
    i ← i − w
return R
```

Die Vorberechnungsphase dieses Algorithmus besteht aus 2^{w-1} Punktverdopplungen und 2^{w-1} Punktadditionen. Die Hauptrechnung benötigt zusätzlich im Schnitt $\frac{n(2^w-1)}{w \cdot 2^w}$ Punktadditionen und $n-w$ bis $n-1$ Punktverdopplungen.

Komplexität:
Vorberechnung: $2^{w-1}-1$ ADD + $2^{w-1}-1$ DBL
Speicher: 2^w-1 Punkte
Hauptrechnung: $n(2^w-1)/w \cdot 2^w$ ADD + $n-w$ bis $n-1$ DBL

Die Komplexität ist also von der Fenstergröße w abhängig. Das optimale w für die kleinstmögliche Komplexität ist erst dann zu finden, wenn das Verhältnis der Laufzeit der Addition zu der Laufzeit der Verdopplung bekannt ist. Dies gilt auch für die folgenden Methoden. Die Wahl des ws wird daher erst nach der Laufzeituntersuchung des Kapitels 8 vorgestellt.

6.1.3 Sliding window Exponentiation

Diese Methode [MvV97, S. 616] scannt k nicht bitweise sondern immer w Bits auf einmal. w bezeichnet man als Fenster (engl. window). Diese Methode berechnet zuerst 2^{w-1} Punkte vor, die in der Hauptrechnung wiederverwendet werden:

ALG **6.1.3: Sliding window Exponentiation**:

Vorberechnung:
INPUT: w, P
OUTPUT: $P_i = i \cdot P_i$ mit $0 \leq i < 2^w$

```
P_1 ← P, t ← 2·P
for i = 1 to 2^{w-1} − 1
    P_i ← P_{i−1} + t
```

Hauptrechnung:
INPUT: k, P, P_i
OUTPUT: $k \cdot P$

```
R ← O, i ← n − 1
while i ≥ 0
    if k_i = 0 then
        R ← 2·R, i ← i − 1
    else
        m ← i − w, if (m < −1) then m ← −1
        j ← m + 1
        while (k_j = 0)
            j++
        s ← (k_i k_{i−1} ... k_j)_2
        while (i ≥ j)
            R ← 2·R, i ← i − 1
        R ← R + P_{(s−1)/2}
        while (i > m)
            R ← 2·R, i ← i − 1
return R
```

Die Vorberechnungsphase dieses Algorithmus besteht aus 1 Punktverdopplung und 2^{w-1} Punktadditionen. Die Hauptrechnung benötigt zusätzlich $\frac{n}{w+1}$ Punktadditionen und $n - w$ bis $n - 1$ Punktverdopplungen.

Komplexität:
Vorberechnung: $2^{w-1} - 1$ ADD + 1 DBL
Speicher: $2^{w-1} - 1$ Punkte
Hauptrechnung: $n/(w+1)$ ADD + $n - w$ bis $n - 1$ DBL

6.1.4 Exponentiation mit Non-adjacent-forms

In der additiven Punktegruppe ist die Invertierung eines Punktes $P = (x,\ y)$ sehr einfach, $-P = (x,\ -y)$. Dadurch kann die Punktmultiplikation durch eine Vorzeichen-behaftete-Darstellung des Exponenten k noch effizienter gemacht werden. Folgende Technik wurde in [MOC97] vorgestellt:

Sei k der Exponent mit der Bitlänge n und w eine Fenstergröße. Eine *width-(w+1) non-adjacent-form (w-NAF)* von k ist ein Feld $N[b], \ldots, N[0]$ ganzer Zahlen, so dass:

- jeder Eintrag $N[j]$ ist entweder 0 oder ungerade und $-2^w < N[j] < 2^w$.
- $k = \sum_{0 \leq j \leq b} N[j] \cdot 2^j$
- höchstens einer der $w + 1$ aufeinanderfolgenden Einträge des Feldes N ist Nicht-Null.

w-NAFs existieren immer und können durch folgenden Algorithmus berechnet werden [Sol00]:

$c \leftarrow k$
$j \leftarrow 0$
while $c > 0$ **do**
  if $c[0] > 0$ **then**
    $u \leftarrow c[w \ldots 0]$
    if $u[w] = 1$ **then**
      $u \leftarrow u - 2^{w+1}$
    $c \leftarrow c - u$
  else
    $u \leftarrow 0$
  $N[j] \leftarrow u, j \leftarrow j + 1$
  $c \leftarrow c/2$
while $j \leq b$
  $N[j] \leftarrow 0; j \leftarrow j + 1$
return $N[b] \ldots N[0]$

Für eine n-Bit lange Zahl k ist N höchstens $n+1$ Felder lang. Die durchschnittliche Gewichtung $\mathrm{w}(N)$ beträgt $n/(w + 2)$ für $n \to \infty$ [Sol00].

ALG **6.1.4: Exponentiation mit Non-adjacent-forms**:

Vorberechnung:
INPUT: w, P
OUTPUT: $P_i = (2i+1) \cdot P$ mit $0 \leq i < 2^w$

$t = 2 \cdot P$
$P_0 = P$
for $i = 1$ to 2^{w-1}
  $P_i \leftarrow P_{i-1} + t$

Hauptrechnung:
INPUT: k, P, P_i
OUTPUT: $k \cdot$

$R \leftarrow \mathcal{O}$
for $i = N.length; i \geq 0; i \leftarrow i - 1$
  $R \leftarrow 2 \cdot R$
  if $N[i] \neq 0$ then
    $R \leftarrow R + N[i] \cdot P$
return R

Komplexität:
Vorberechnung: $2^{w-1} - 1$ ADD + 1 DBL
Speicher: $2^{w-1} - 1$ Punkte
Hauptrechnung: $n/(w+2)$ ADD + $n - w$ bis $n - 1$ DBL

6.1.5 Spezialfall: ± 1-Ketten

Die Punktmultiplikation mit ± 1-Ketten ist der Spezialfall von der vorangegangen Exponentiation mit Non-adjacent-forms, nämlich für $w = 1$. Die 1-NAF eines jeden Exponenten k besteht nur aus $-1, 0$ und 1.

Sei n die maximal mögliche Bitlänge des Exponenten k, dann werden $n - 1$ Punkte vorberechnet, alle 2er-Potenzen von P. Die Punktmultiplikation kann dann alleine mit Punktadditionen berechnet werden, Punktverdopplungen sind nicht mehr nötig.

ALG **6.1.5: Exponentiation mit ± 1-Ketten**:

Vorberechnung:
INPUT: P
OUTPUT: $P_i = 2^i \cdot P$ mit $0 \leq i < n$

```
P_0 = P
for i = 1 to n - 1
    P_i ← 2 · P_{i-1}
```

Hauptrechnung:
INPUT: k, P, P_i, $N(k)$
OUTPUT: $k \cdot P$

```
R ← O
for i = N.length; i ≥ 0; i ← i - 1
    if N[i] == -1 then
        R ← R - P_i
    else bif N[i] == 1 then
        R ← R + P_i
return R
```

Komplexität:
Vorberechnung: $n-1$ DBL
Speicher: $n-1$ Punkte
Hauptrechnung: $n/2$ ADD

6.1.6 LimLee Exponentiation

In [LL94] wird eine andere Methode vorgestellt, die mit großem Aufwand in der Vorberechnung eine äußerst effiziente Hauptrechnung hat.

Sei k wieder der n-Bit lange Exponent. Teile k in h gleich große Teile der Länge v ($k_i = (k_{ia+h-1}k_{ia+h-2}\ldots k_{ia+1}k_{ia+0})$, mit $i = 0, \ldots h-1$). Diese h Teile der Länge a teile wiederum in v gleich große Teile der Länge b.

Beispiel: $n = 24$, $h = 4$, $v = 3$:

$$k = \left|\begin{array}{cc|cc|cc} k_{0,2,1} & k_{0,2,0} & k_{0,1,1} & k_{0,1,0} & k_{0,0,1} & k_{0,0,0} \\ k_{1,2,1} & k_{1,2,0} & k_{1,1,1} & k_{1,1,0} & k_{1,0,1} & k_{1,0,0} \\ k_{2,2,1} & k_{2,2,0} & k_{2,1,1} & k_{2,1,0} & k_{2,0,1} & k_{2,0,0} \\ k_{3,2,1} & k_{3,2,0} & k_{3,1,1} & k_{3,1,0} & k_{3,0,1} & k_{3,0,0} \end{array}\right|$$

Dabei läuft bei $k_{i,j,l}$ i von 0 bis $h-1$, j von 0 bis $v-1$ und l von 0 bis $b-1$.

Sei $\mu \in 0, \ldots, 2^h - 1$ in Binärdarstellung und $K_\mu = \sum_{i=0,\ldots,h-1} \mu_i 2^{ai}$. Berechne und speichere alle $P_{K_\mu} = K_\mu \cdot P$ und weiterhin alle $P_{K_\mu,j} = 2^{jb} \cdot P_{K_\mu}$ für $j = 0, \cdot, v-1$. Das sind $v(2^h - 1)$ Einträge.

$k \cdot P$ lässt sich mit folgendem Algorithmus berechnen:

ALG **6.1.6: LimLee Exponentiation**:

Vorberechnung:
INPUT: P
OUTPUT: $P_{i,j} = 2^{i \cdot b} \cdot jP$ mit $0 \le i < v$ und $0 \le j < h$

```
D_{0,0} = P
for i = 0 to h - 1
  for j = 0 to v - 1
    if i = 0 then j = 1
    if j = 0 then
      D_{i,j} = D_{i-1,v-1}
    else
      D_{i,j} = D_{i,j-1}
    for k = 0 to b - 1
      D_{i,j} = 2 · D_{i,j}
for k = 0 to v - 1
  for i = 1 to 2^h - 1
  P_{k,i} = O
    for j = 0, s = 0 to s < i, j = j + 1
      if i&j ≠ 0 then
        s = s + (i&j)
        P_{k,i} = P_{k,i} + D_{k,j-1}
return P_{-,-}
```

Fortsetzung ALG **6.1.6**:

```
Hauptrechnung:
  R ← O
  for i = b − 1 to 0
    R ← 2 · R
    for j = v − 1 to 0
      // μ ist hier die Binärdarstellung
      // von k_{h−1,j,0} ... k_{0,j,0}
      R ← R + P_{K_μ,j}
  return R
```

Komplexität:
Vorberechnung: $v(2^{h-1}-1)$ ADD, $n - \frac{n}{hv}$ DBL
Speicher: $(2^h - 1) \cdot v$ Punkte
Hauptrechnung: $\frac{n}{h}$ ADD, $\frac{n}{hv} - 1$ DBL

6.1.7 Komplexitätsvergleich und Analyse

Vorgestellt worden sind fünf verschiedene Methoden zur einfachen Punktmultiplikation der Form $k \cdot P$. Vier von ihnen profitieren von einer Vorberechnungsphase: Eine Menge von Vielfachen des Punktes P wird einmal berechnet und in der Hauptrechnungsphase mehrfach wiederverwendet. Diese Vorberechnungsphase kann, wenn P im Voraus bekannt ist, einmal berechnet und gespeichert werden. Wenn diese Operation mit dem gleichen Punkt P wiederholt wird, können diese Punkte in der Methode wiederverwendet werden. Ein Anwendungsszenario ist die Signaturerzeugung: P ist in diesem Fall der Basispunkt G der EC Parameter, die man für viele Signaturen wiederverwendet.

Tabelle 6.1 fasst die Komplexitäten zusammen.

6.2 Mehrfache Punktmultiplikation

In diesem Abschnitt beschreiben wir die Methoden zur Punktmultiplikation, die sich besonders zur Signaturverifikation eignen.

Die teuerste Operation der Verifikation ist $h_1 \cdot G + h_2 \cdot W$, also eine zwei Punkt-

Methode	Vorberechnung		Hauptrechnung	
	# ADD	# DBL	# ADD	# DBL
left-to-right	-	-	$\approx \frac{n}{2}$	$n-1$
Fixed Sliding-window	2^{w-1}	2^{w-1}	$\frac{n(2^w-1)}{w \cdot 2^w}$	$n-(n\%w)$ oder $n-1$
slid. window	2^{w-1}	1	$\frac{n}{w+1}$	$n-w$ bis $n-1$
w-NAF	2^{w-1}	1	$\frac{n}{w+2}$	$n-w$ bis $n-1$
LimLee	$v(2^{h-1}-1)$	$n-\frac{n}{hv}$	$\frac{n}{h}$	$\frac{n}{hv}-1$

Tabelle 6.1: Komplexitäten der Algorithmen zur einfachen Punktmultiplikation

multiplikationen, deren Ergebnis noch miteinander addiert wird. Die Idee ist, diese drei Einzeloperationen zu einer einzigen zu verknüpfen und dabei Punktverdopplungen einzusparen. Dafür gibt es zwei Ansätze: Die einzelnen benötigten Punktadditionen und -verdopplungen werden simultan ausgeführt. Damit spart man etwa die Hälfte der Verdopplungen. Dieser Ansatz wird in den Algorithmen von 6.2.1, 6.2.2 und 6.2.3 verwendet. Der zweite führt die Punktverdopplungen auch simultan aus, die Addition aber nur abwechselnd. Dadurch ist im Schnitt die Anzahl der benötigten Punktadditionen kleiner. Dieser Ansatz kommt in 6.2.4 und 6.2.5 zum Tragen.

Jeder dieser Algorithmen ist nicht auf zwei Exponenten mit zwei Basen beschränkt, sie können beliebig erweitert werden. Für unseren Zweck genügt aber der Fall $h_1 \cdot G + h_2 \cdot W$. Wir werden daher die Algorithmen nur für diese Operation beschreiben. Im Folgenden seien mit $i = 1,\ 2$ k_i die ganzzahligen Exponenten und P_i die Basen. R sei der Ergebnispunkt.

Auch die Algorithmen dieses Abschnitts beinhalten eine Vorberechnung. Sie wird wieder entsprechend markiert sein.

6.2.1 Simultane Exponentiation

Die simultane Exponentiation [MvV97, S. 618, Alg. 14.88] ist die einfachste dieser Techniken. Für die Operation $R = k_1 \cdot P_1 + k_2 \cdot P_2$ besteht die Vorberechnung ledig-

lich aus der einen Punktaddition $A = P_1 + P_2$. Das Grundprinzip leitet sich dann vom Algorithmus ALG 6.1.1 ab: Von links nach rechts wird der Zwischenergebnis-Punkt R verdoppelt und für jedes Paar $(k_1[j],\ k_2[j]) \neq (0,\ 0)$, $j = n-1, \ldots, 0$, $P_1,\ P_2$ oder A dazuaddiert:

ALG **6.2.1: Simultane Exponentiation**:

```
Vorberechnung:
  A = P1 + P2

Hauptrechnung:
  R ← O
  for i = n − 1 down to 0
    R ← 2 · R
    if k1[i] = 1 then
      if k2i] = 1 then
        R ← R + A
      else
        R ← R + P1
    else
      if k2[i] = 1then
        R ← R + P2
  return R
```

Komplexität:
Vorberechnung: 1 ADD
Speicher: 1 Punkte
Hauptrechnung: $3/4n$ ADD + $n - 1$ DBL

6.2.2 Simultane 2^w-ary Exponentiation

Die simultane 2^w-adische Exponentiation [Str64] [Moe01] betrachtet w Bits jedes Exponenten gleichzeitig. Sie ist damit die Generalisierung des Algorithmus 6.1.2.

Der Vorteil dieser Methode gegenüber der letzten ist, dass sie bei $2^{2w} - 3$ vorberechneten Punkten nur $\frac{1-\frac{1}{2^{2w}}}{w} \cdot n$ Punktadditionen benötigt.

ALG **6.2.2: Simultane 2^w-ary Exponentiation**:

Vorberechnung:
INPUT: w, P
OUTPUT: $P_{i,j} = i \cdot P_1 + j \cdot P_2$ mit $0 \leq i, j < 2^w$
  $P_{0,0} \leftarrow \mathcal{O}$
  $P_{1,1} \leftarrow P_1 + P_2$
  for $i = 1$ to $2^w - 1$
    $j = 2i$
    $P_{0,j} \leftarrow 2 \cdot P_{0,i}$, $P_{0,j+1} \leftarrow P_{0,i} + P_{0,i+1}$
    $P_{j,0} \leftarrow 2 \cdot P_{i,0}$, $P_{j+1,0} \leftarrow P_{i,0} + P_{i+1,0}$
    $P_{j,j} \leftarrow 2 \cdot P_{i,i}$, $P_{j+1,j+1} \leftarrow P_{i,i} + P_{i+1,i+1}$
  for $i = 1$ to $2^w - 1$
    for $j = 1$ to $2^w - 1$
      if $(i = j)$ then $j \leftarrow j + 1$
      $P_{i,j} \leftarrow Pi, 0 + P_{0,j}$

Hauptrechnung:
INPUT: k_1, P_1, k_2, P_2, w, $P_{i,j} = i \cdot P_1 + j \cdot P_2$ mit $0 \leq i, j < 2^w$
OUTPUT: $k_1 \cdot P_1 + k_2 \cdot P_2$
  $R \leftarrow \mathcal{O}$
  for $j = \lfloor (n-1)/w \rfloor w$ down to 0
    for $i = 1$ to w
      $R \leftarrow 2 \cdot R$
    if $(k_1[j+w-1 \ldots j], (k_2[j+w-1 \ldots j]) \neq (0,0)$ then
      $R \leftarrow R + P_{k_1[j+w-1 \ldots j], (k_2[j+w-1 \ldots j])}$
  return R

Komplexität:
Vorberechnung: $2^{2w} - 2^{2w-2} - 2$ ADD, $2^{2w-2} - 1$ DBL
Speicher: $2^{2w} - 3$ Punkte
Hauptrechnung: $\frac{1 - \frac{1}{2^{2w}}}{w} \cdot n$ ADD, $n - 1$ oder $n - (n \% w)$ DBL

6.2.3 Simultane sliding window Exponentiation

Die letzte der simultanen Exponentiationsmethoden, die simultane sliding window Methode [YLL94] [Moe01], ist eine Generalisierung der einfachen Sliding Window Methode.

ALG **6.2.3: Simultane Sliding Window Exponentiation**:

Vorberechnung:
INPUT: w, P
OUTPUT: $P_{i,j} = i \cdot P_1 + j \cdot P_2$ mit $0 \le i,j < 2^w$, wobei i oder j ungerade ist.

```
P12 ← 2·P1, P22 ← 2·P2
P1,0 ← P1, P0,1 ← P2
for i = 3 to 2^w − 1 step 2
    P0,i ← P0,i−2 + P22, Pi,0 ← Pi−2,0 + P12
for i = 1 to 2^w − 1 step 2
    for j = 1 to 2^w − 1
        Pi,j ← Pi,j−1 + P2
for i = 2 to 2^w − 1 step 2
    for j = 1 to 2^w − 1 step 2
        Pi,j ← Pi−1,j + P1
```

Hauptrechnung:
INPUT: w, P_1, P_2, k_1, k_2, $P_{i,j} = i \cdot P_1 + j \cdot P_2$ mit $0 \le i,j < 2^w$, wo i oder j ungerade ist.
OUTPUT: $k_1 \cdot P_1 + k_2 \cdot P_2$

```
R ← O, j ← n − 1
while j ≥ 0
    if k1[j] = 0 and k2[j] = 0
        R ← 2·R, j ← j − 1
    else
        jnew ← max(j − w, −1), h ← jnew + 1
        while k1[h] = 0 and k2[h] = 0
            h ← h + 1 (d.h. j ≥ h > jnew)
        K1 ← k1[j ... h], K2 ← K2[j ... h]
```

Fortsetzung ALG **6.2.3**:

```
      while j ≥ h
        R ← 2 · R,  j ← j − 1
      R ← R + A_{K,L}
      while j > j_new
        R ← 2 · R,  j ← j − 1
  return R
```

Komplexität:
Vorberechnung: $2^{2w} - 2^{2w-2} - 2$ ADD, 2 DBL
Speicher: $2^{2w} - 2^{2w-2} - 2$ Punkte
Hauptrechnung: $\frac{n}{w+\frac{1}{3}}$ ADD, $n-1$ oder $n-(n\%w)$ DBL

6.2.4 Basic Interleaving Exponentiation

Auch hier geht es um die komplexe Operation $k_1 \cdot P_1 + k_2 \cdot P_2$. Die drei letzten Methoden schöpften ihre Effizienz daraus, dass sie die Punktadditionen sowie die Punktverdopplungen simultan für die Exponenten k_1 und k_2 ausführten. Die *Basic Interleaving Exponentiation* [Moe01] sowie die nachfolgende *w-NAF Interleaving Exponentiation* [Moe01] führen nur die Punktverdopplung simultan aus. Die Punktadditionen jedoch getrennt. Das hat zwei Vorteile: Erstens müssen weniger Punkte vorberechnet werden. Zweitens kann diese Methode beschleunigt werden bei Szenarien, in denen einer der Punkte P_1 oder P_2 im Voraus bekannt ist. Dies kann der Fall sein bei der Signaturverifikation. Dabei ist P_1 der Basispunkt G. Vielfache dieses Punktes können gemäß den folgenden Methoden vorberechnet und bei Bedarf wiederverwendet werden.

Folgende Methode ist eine Generalisierung der Sliding Window Methode mit nur einem Exponenten.

ALG **6.2.4: Basic Interleaving Exponentiation**:

Vorberechnung:
INPUT: w_1, w_2, P_1, P_2
OUTPUT: $P_{1,j} = j \cdot P_1$ und $P_{2,j} = j \cdot P_2$ mit $0 \leq j < 2^w_{1,2}$, j ungerade

Fortsetzung ALG **6.2.4**:

```
  P_12 ← 2·P_1,  P_22 ← 2·P_2
  for i = 3 to 2^w1 step 2
    P_1,i ← P_1,i−1 + P_12
  for i = 3 to 2^w2 step 2
    P_2,i ← P_2,i−1 + P_22

Hauptrechnung:
INPUT: w_1, w_2, P_1, P_2, P_1,j = j·P_1 und
       P_2,j = j·P_2 mit 0 ≤ j < 2^w_1,2, j ungerade
OUTPUT: k_1·P_1 + k_2·P_2
  R ← O
  for i = 1 to 2
    window_handle_i ← null
  for j = n − 1 down to 0
    R ← 2·R
  for i = 1 to 2
    if window_handle_i =null and k_i[j] = 1
      J ← j − w_i + 1
      while k_i[J] = 0
        J ← J + 1    → j ≥ J > j − w und J ≥ 0
      window_handle_i ← J
      K_i ← K_i[j ... J]
    if window_handle_i = j
      R ← R + (E_i · P_i)   (mittels Tabelleneintrag)
      window_handle_i ← null
  return R
```

Komplexität:
Vorberechnung: $2^{w_1-1} + 2^{w_2-1} - 2$ ADD, 2 DBL
Speicher: $2^{w_1-1} + 2^{w_2-1} - 2$ Punkte
Hauptrechnung: $\frac{n\cdot(w_1+w_2+2)}{(w_1+1)\cdot(w_2+1)}$ ADD, $n-1$ oder $n - max_i w_i$ DBL

6.2.5 w-NAF-based Interleaving Exponentiation

Die *w-NAF-based interleaving Exponentiation* [Moe01] ist eine Generalisierung der einfachen Exponentiation mit NAF, ALG 6.1.4. Sie profitiert auch davon, dass

die Negation von Punkten fast ganz umsonst ist. So werden im Vergleich der letzten Methode weitere Punktadditionen eingespart.

ALG **6.2.5: w-NAF-based Interleaving Exponentiation**:

Vorberechnung:
INPUT: w_1, w_2, P_1, P_2
OUTPUT: $P_{1,j} = j \cdot P_1$ und $P_{2,j} = j \cdot P_2$ mit $0 \leq j < 2^w_{1,2}$, j ungerade

```
P12 ← 2 · P1, P22 ← 2 · P2
for i = 3 to 2^w1 step 2
    P1,i ← P1,i−1 + P12
for i = 3 to 2^w2 step 2
    P2,i ← P2,i−1 + P22
```

Hauptrechnung:
INPUT: w_1, w_2, P_1, P_2, $P_{1,j} = j \cdot P_1$ und $P_{2,j} = j \cdot P_2$ mit $0 \leq j < 2^w_{1,2}$, j ungerade
OUTPUT: $k_1 \cdot P_1 + k_2 \cdot P_2$

```
R ← O
for i = 1 to 2
    Ni[n], ..., Ni[ni + 1] ← 0, ..., 0
    Ni[ni], ..., Ni[0] ← w-NAF von ki
for j = n down to 0
    R ← 2 · R
    for i = 1 to 2
        if Ni[j] = 0
            R ← R + Ni[j] · Pi
return R
```

Komplexität:
Vorberechnung: $2^{w_1-1} + 2^{w_2-1} - 2$ ADD, 2 DBL
Speicher: $2^{w_1-1} + 2^{w_2-1} - 2$ Punkte
Hauptrechnung: $\frac{n \cdot (w_1+w_2+4)}{(w_1+2) \cdot (w_2+2)}$ ADD, $n-1$ oder $n - max_i w_i$ DBL

Methode	Vorberechnung		Hauptrechnung	
	# ADD	# DBL	# ADD	# DBL
Sim. Exp.	1	-	$\frac{3}{4} \cdot n$	$n-1$
Sim. 2^w-ary	$2^{2w} - 2^{2w-2} - 2$	$2^{2w-2} - 1$	$\frac{1-\frac{1}{2^{2w}}}{w} \cdot n$	$n-(n\%w)$ oder $n-1$
Sim. slid. win.	$2^{2w} - 2^{2w-2} - 2$	2	$\frac{n}{w+\frac{1}{3}}$	$n-(n\%w)$ oder $n-1$
Basic Inter.	$2^{w_1-1} + 2^{w_2-1} - 2$	2	$\frac{n \cdot (w_1+w_2+2)}{(w_1+1) \cdot (w_2+1)}$	$n - max_i w_i$ oder $n-1$
w-NAF Inter.	$2^{w_1-1} + 2^{w_2-1} - 2$	2	$\frac{n \cdot (w_1+w_2+4)}{(w_1+2) \cdot (w_2+2)}$	$n - max_i w_i$ oder $n-1$

Tabelle 6.2: Komplexitäten der Algorithmen zur mehrfachen Punktmultiplikation

6.2.6 Komplexitätsvergleich und Analyse

Auch in der Klasse der mehrfachen Punktmultiplikationen der Form $k_1 \cdot P_1 + k_2 \cdot P_2$ wurden fünf Algorithmen vorgestellt. Die ersten drei führen Punktadditionen und -verdopplungen simultan für die zwei Exponenten k_1 und k_2 aus. Dafür müssen in der Vorberechnungsphase Linearkombinationen der Punkte P_1 und P_2 berechnet werden. Die letzten zwei dieser Algorithmen führen zwar die Verdopplungen simultan aus, die Punktadditionen aber getrennt. Dementsprechend werden in der Vorberechnungsphase jeweils Vielfache der beiden Punkte, nicht aber deren Linearkombinationen berechnet. Tabelle 6.2 fasst die Komplexitäten der vorgestellten Algorithmen zusammen. Dabei ist die Anzahl der Punktadditionen (#ADD) als Erwartungwert zu lesen und nicht absolut.

6.3 Die Algorithmen in Bezug auf die Anwendung

Die Punktmultiplikation wird in der Elliptische Kurven Kryptographie in verschiedenen Szenarien verwendet. Diese lassen sich in drei Gruppen teilen.

In die erste Gruppe fällt die Signaturerzeugung. Die aufwendigste Operation ist die Berechnung von $k \cdot G$, wobei G der in den EC Domain Parameter festgelegte Basispunkt ist. Dieser ist als Teil der EC Domain Parameter also vor der Signaturerzeugung bekannt und bleibt für viele Signaturen bestehen. Daher ist es möglich,

Vielfache dieses Punktes im Voraus zu berechnen, abzuspeichern und bei Bedarf zu verwenden.

In die zweite Gruppe fallen die kryptographischen Primitive, bei denen die EC Domain Parameter nicht im Voraus bekannt sind. Vorberechnungen von Vielfachen P's sind nicht möglich. Dazu gehört der Schlüsselaustausch mit elliptischen Kurven.

Die dritte Gruppe umfasst die Primitive, die die Operation $k \cdot G + l \cdot W$ benötigen. Hierbei ist W ein fremder öffentlicher Schlüssel und G der Basispunkt der dazugehörigen ECParameters. Zu dieser Gruppe gehören die Signaturverifikationen.

Mit Hilfe der verschiedenen Techniken aus den Abschnitten 6.1 und 6.2 können der Schlüsselaustausch, die Signaturerzeugung und die Signaturverifikation optimal implementiert werden. Es gilt nun, die beste Methode zu jeder Anwendung zu finden. Es folgt eine Grobeinteilung, die im weiteren Verlauf dieses Kapitels verfeinert wird.

6.3.1 Geeignete Wahl für den Schlüsselaustausch

Die aufwendigste Operation beim Schlüsselaustausch ist die einfache Punktmultiplikation $s \cdot W'$. Das Problem ist, dass der öffentliche Schlüssel W' nicht zwingend im Voraus bekannt ist. Die optimale Methode muss also geeignet sein für eine einfache Punktmultiplikation, die auf keine vorberechneten Punkte zurückgreifen muss.

Zu dieser Gruppe gehören die Square-and-Multiply (ALG 6.1.1), die Fixed-size-sliding-window Exponentiation(ALG 6.1.2), die Sliding window Exponentiation (ALG 6.1.3) und die Exponentiation mit Non-adjacent-forms (ALG 6.1.4).

Die Vorberechnung bei der Exponentiation mit ± 1-Ketten oder mit LimLee erfordert immer eine große Anzahl an Punktadditionen und -verdopplungen, die, wenn während der Hauptrechnung ausgeführt, die Laufzeit unnötig verlängern würden.

Es gibt noch eine weitere Möglichkeit, die Punktmultiplikation für den Schlüsselaustausch effizient zu implementieren: Bei der Exponentiation mit Non-adjacent-forms kann die NAF-Kette in t gleich große Teile der Länge n/t geteilt werden. Für jeden dieser t Teile wird der Punkt $G_i = 2^{i \cdot \frac{n}{t}} \cdot G, i = 0, \ldots, t-1$ vorberechnet. Diese G_is sind die neuen Basispunkte, die für die w-NAF Interleaving

Exponentiation (ALG 6.2.5) verwendet werden.

Es sind also sechs passende Algorithmen vorhanden, die genauer auf ihre Komplexitäten hin untersucht werden müssen. Dazu muss in den Algorithmen ALG 6.1.2, ALG 6.1.3 und die optimale Wahl der Window-Größe w in Abhängigkeit von der Exponentenlänge n bestimmt werden. Dann können die Komplexitäten der einzelnen Algorithmen miteinander verglichen werden.

6.3.2 Geeignete Wahl für die Signaturerzeugung

Bei der Signaturerzeugung kann davon ausgegangen werden, dass für die Punktmultiplikation $s \cdot G$ der Basispunkt G vorher bekannt ist und für viele Signaturen gültig bleibt. Da lohnt es sich, eine Tabelle mit vorberechneten Punkten im Voraus anzulegen und sie bei jeder Signaturerzeugung wieder zu verwenden. Das bedeutet unter Anderem, dass man einen mitunter großen Teil der Laufzeit auf eine unkritische Phase auslagert und so die eigentliche Laufzeit erheblich verkürzt.

Algorithmen, die diese Methode zulassen, sind all jene, die eine Vorberechnungsphase besitzen: Die Fixed-size-sliding-window Exponentiation(6.1.2), die Sliding window Exponentiation (ALG 6.1.3), die Exponentiation mit Non-adjacent-forms (ALG 6.1.4), die Exponentiation mit ± 1-Ketten (ALG 6.1.5) und mit der LimLee Exponentiation (ALG 6.1.6).

6.3.3 Geeignete Wahl für die Verifikation

Die Signaturverifikation benötigt zwei einfache Punktmultiplikationen: $h_1 \cdot G + h_2 \cdot W$. Es können die gleichen Methoden wie zur Signaturerzeugung verwendet werden. Oder diese zwei Operationen werden zu einer einzigen Operation zusammengefasst und mit einem der Algorithmen aus Abschnitt 6.2 berechnet werden. Hierbei ist W der öffentliche Schlüssel, der dem Signierer gehört, G ist wieder der Basispunkt. Nun ist es natürlich möglich, dass die EC Domain Parameter des Signierers andere sind als die eigenen. Dementsprechend ist das Verwenden von vorberechneten Vielfachen von G nicht unbedingt möglich. Dann müssen die Parameter aus Abschnitt 6.2 mit entsprechend kleinen Parametern gewählt werden.

6.4 Koordinatensysteme

In den folgenden Abschnitten werden verschiedene mögliche Koordinatensysteme vorgestellt. Das erste, das Affine, ist das bekannteste. Ein Punkt besitzt im Affinen System zwei Koordinaten x und y. Da jede Punktaddition und -verdopplung in diesem System eine Invertierung in $\mathbb{F}_p$ beinhaltet, wurden bisher auch projektive Koordinatensysteme untersucht, die ohne diese kostspielige Operation auskommen: Das Projektive, das Jacobische, das Chudnowsky Jacobische und das Modifiziert Jacobische Koordinatensystem. Letzteres wurde in [CMO98] vorgeschlagen, das Chudnwsky Jacobische in [DVC87]. Diese fünf verschiedenen Systeme wurden in [CMO98] teilweise untereinander kombiniert und auf ihre Komplexität untersucht. Dies wird in Abschnitt 6.4.7 fortgeführt. Das Einsetzen dieser Kombinationen in den Algorithmus ALG 6.1.4 in [CMO98] wird im nächsten Kapitel auf alle in dieser Arbeit vorgestellten Verfahren erweitert.

6.4.1 Affines Koordinatensystem

Sei $E : y^2 = x^3 + ax + b$ die Kurvengleichung mit $a, b \in \mathbb{F}_p$ und $4a^3 + 27b^2 \neq 0$. Ein Punkt P auf E wird durch die x- und die y-Koordinate durch $P = (x,\ y)$ dargestellt. Diese Koordinaten nennt man affin, das Affine System werden wir künftig mit dem Symbol $\mathcal{A}$ bezeichnen. Der Punkt im Unendlichen wird mit $\mathcal{O}$ bezeichnet und hat die Form $\mathcal{O} \stackrel{\text{def}}{=} (0,\ 0)$.

Seien $P = (x_P,\ y_P)$ und $Q = (x_Q,\ y_Q)$ zwei Punkte auf der Kurve E. Dann lässt sich $R = (x_R,\ y_R) = P + Q$ wie folgt berechnen:

$$\begin{aligned} x_R &= \lambda^2 - x_P - x_Q \\ y_R &= \lambda(x_R - x_P) - y_P \\ \lambda &= (y_Q - y_P)/(x_Q - x_P) \end{aligned} \tag{6.1}$$

Für $R = (x_R,\ y_R) = 2 \cdot P$ gelten folgende Formeln:

$$\begin{aligned} x_R &= \lambda^2 - 2x_P \\ y_R &= \lambda(x_R - x_P) - y_P \\ \lambda &= (3x_P^2 + a)/(2y_P) \end{aligned} \tag{6.2}$$

Operation	A	M	S	I
$\mathcal{A}+\mathcal{A}$	6	2	1	1
$2\cdot\mathcal{A}$	8	1	2	1

6.4.2 Projektives Koordinatensystem

Sei $E : Y^2Z = X^3 + aXZ^2 + bZ^3$ die Kurvengleichung mit $a, b \in \mathbb{F}_p$ und $4a^3+27b^2 \neq 0$. Ein Punkt P auf E wird durch die X-, die Y- und die Z-Koordinate durch $P = (X,\ Y,\ Z)$ dargestellt. Diese Koordinaten nennt man projektiv, das Projektive System werden wir künftig mit dem Symbol $\mathcal{P}$ bezeichnen. Der Punkt im Unendlichen hat die Form $\mathcal{O} \stackrel{\text{def}}{=} (1,\ 1,\ 0)$.

Man kann für einen Punkt P die Koordinaten vom Affinen System zum projektiven System überführen und umgekehrt:

$$\begin{aligned} \mathcal{A} \to \mathcal{P} &: X = x,\ Y = y,\ Z = 1 \\ \mathcal{P} \to \mathcal{A} &: x = X/Z,\ y = Y/Z \end{aligned}$$

Seien $P = (X_P,\ Y_P,\ Z_P)$ und $Q = (X_Q,\ Y_Q,\ Z_Q)$ zwei Punkte auf der Kurve E. Dann lässt sich $R = (X_R,\ Y_R,\ Z_R) = P + Q$ wie folgt berechnen:

$$\begin{aligned} X_R &= vA \\ Y_R &= u(v^2X_PZ_Q - A) - v^3Y_PZ_Q \\ Z_R &= v^3Z_PZ_Q \\ \text{mit} & \\ u &= Y_QZ_P - Y_PZ_Q \\ v &= X_QZ_P - X_PZ_Q \\ A &= u^2Z_PZ_Q - v^3 - 2v^2Z_PZ_Q \end{aligned} \tag{6.3}$$

Für $R = (X_R,\ Y_R,\ Z_P) = 2 \cdot P$ gelten folgende Formeln:

$$
\begin{aligned}
X_R &= 2hs \\
Y_R &= w(4B - h) - 8Y_P^2 s^2 \\
Z_R &= 8s^3 \\
\text{mit} & \\
w &= aZ_P^2 + 3X_P^2 \\
s &= Y_P Z_P \\
B &= X_P Y_P s \\
h &= w^2 - 8B
\end{aligned}
\tag{6.4}
$$

Operation	A	M	S	I
$\mathcal{P} + \mathcal{P}$	7	13	2	-
$\mathcal{P} + \mathcal{A}$	7	10	2	-
$\mathcal{A} + \mathcal{A}$	7	5	2	-
$2 \cdot \mathcal{P}$	15	7	5	-
$2 \cdot \mathcal{A}$	15	5	4	-

6.4.3 Jacobisches Koordinatensystem

Sei $E : Y^2Z^4 = X^3 + aXZ^4 + bZ^6$ die Kurvengleichung mit $a, b \in \mathbb{F}_p$ und $4a^3+27b^2 \neq 0$. Ein Punkt P auf E wird durch die X-, die Y- und die Z-Koordinate durch $P = (X,\ Y,\ Z)$ dargestellt. Diese Koordinaten nennt man jacobisch, das Jacobische System werden wir künftig mit dem Symbol $\mathcal{J}$ bezeichnen. Der Punkt im Unendlichen hat wie im projektiven System die Form $\mathcal{O} \stackrel{\text{def}}{=} (1,\ 1,\ 0)$.

Man kann für einen Punkt P die Koordinaten vom Affinen System zum jacobischen System überführen und umgekehrt:

$$
\begin{aligned}
\mathcal{A} \to \mathcal{P} &: \quad X = x,\ Y = y,\ Z = 1 \\
\mathcal{P} \to \mathcal{A} &: \quad x = X/Z^2,\ y = Y/Z^3
\end{aligned}
$$

Seien $P = (X_P,\ Y_P,\ Z_P)$ und $Q = (X_Q,\ Y_Q,\ Z_Q)$ zwei Punkte auf der Kurve E. Dann lässt sich $R = (X_R,\ Y_R,\ Z_R) = P + Q$ wie folgt berechnen:

$$
\begin{aligned}
X_R &= -H^3 - 2U_1H^2 + r^2 \\
Y_R &= -S_1H^3 + r(U_1H^2 - X_R) \\
Z_R &= Z_P Z_Q H \\
&\text{mit} \\
U_1 &= X_P Z_Q^2 \\
U_2 &= X_Q Z_P^2 \\
S_1 &= Y_P Z_Q^3 \\
S_2 &= Y_Q Z_P^3 \\
H &= U_2 - U_1 \\
r &= S_2 - S_1
\end{aligned}
\tag{6.5}
$$

Für $R = (X_R,\ Y_R,\ Z_P) = 2 \cdot P$ gelten folgende Formeln:

$$
\begin{aligned}
X_R &= T \\
Y_R &= -8Y_P^4 + M(S - T) \\
Z_R &= 2Y_P Z_P \\
&\text{mit} \\
S &= 4X_P Y_P^2 \\
M &= 3X_P^2 + aZ_P^4 \\
T &= -2S + M^2
\end{aligned}
\tag{6.6}
$$

Operation	A	M	S	I
$\mathcal{J} + \mathcal{J}$	7	12	4	-
$\mathcal{J} + \mathcal{A}$	7	8	3	-
$\mathcal{A} + \mathcal{A}$	7	4	2	-
$2 \cdot \mathcal{J}$	13	4	6	-
$2 \cdot \mathcal{A}$	13	2	4	-

6.4.4 Chudnowsky-Jacobisches Koordinatensystem

Sei $E : Y^2Z^4 = X^3 + aXZ^4 + bZ^6$ die Kurvengleichung mit $a, b \in \mathbb{F}_p$ und $4a^3+27b^2 \neq 0$. Ein Punkt P auf E wird durch die X-, die Y- und die Z-Koordinate und zusätzlich den Werten (Z^2) und (Z^3) durch $P = (X,\ Y,\ Z,\ (Z^2),\ (Z^3))$ dar-

gestellt. Diese Koordinaten nennt man Chudnowsky-jacobisch, das Chudnowsky-Jacobische System werden wir künftig mit dem Symbol $\mathcal{J}^{\mathcal{C}}$ bezeichnen. Der Punkt im Unendlichen hat die Form $\mathcal{O} \stackrel{\text{def}}{=} (1,\ 1,\ 0,\ 0,\ 0)$.

Man kann für einen Punkt P die Koordinaten vom Affinen System zum jacobischen System überführen und umgekehrt:

$$\begin{aligned} \mathcal{A} \to \mathcal{P} &: X = x,\ Y = y,\ Z = 1,\ (Z^2) = 1,\ (Z^3) = 1 \\ \mathcal{P} \to \mathcal{A} &: x = X/(Z^2),\ y = Y/(Z^3) \end{aligned}$$

Seien $P = (X_P,\ Y_P,\ Z_P,\ (Z_P^2),\ (Z_P^3))$ und $Q = (X_Q,\ Y_Q,\ Z_Q,\ (Z_Q^2),\ (Z_Q^3))$ zwei Punkte auf der Kurve E. Dann lässt sich $R = (X_R,\ Y_R,\ Z_R,\ (Z_R^2),\ (Z_R^3)) = P + Q$ wie folgt berechnen:

$$\begin{aligned} X_R &= -H^3 - 2U_1H^2 + r^2 \\ Y_R &= -S_1H^3 + r(U_1H^2 - X_R) \\ Z_R &= Z_PZ_QH \\ (Z_R^2) &= Z_R^2 \\ (Z_R^3) &= Z_R^3 \\ &\text{mit} \\ U_1 &= X_P(Z_Q^2) \\ U_2 &= X_Q(Z_P^2) \\ S_1 &= Y_P(Z_Q^3) \\ S_2 &= Y_Q(Z_P^3) \\ H &= U_2 - U_1 \\ r &= S_2 - S_1 \end{aligned} \tag{6.7}$$

Für $R = (X_R,\ Y_R,\ Z_P) = 2 \cdot P$ gelten folgende Formeln:

$$
\begin{aligned}
X_R &= T \\
Y_R &= -8Y_P^4 + M(S-T) \\
Z_R &= 2Y_P Z_P \\
(Z_R^2) &= Z_R^2 \\
(Z_R^3) &= Z_R^3 \\
&\text{mit} \\
S &= 4X_P Y_P^2 \\
M &= 3X_1^2 + a(Z_P^2)^2 \\
T &= -2S + M^2
\end{aligned}
\tag{6.8}
$$

Operation	A	M	S	I
$\mathcal{J}^{\mathcal{C}} + \mathcal{J}^{\mathcal{C}}$	7	11	3	-
$\mathcal{J}^{\mathcal{C}} + \mathcal{A}$	7	8	3	-
$\mathcal{A} + \mathcal{A}$	7	5	3	-
$2 \cdot \mathcal{J}^{\mathcal{C}}$	13	5	6	-
$2 \cdot \mathcal{A}$	13	3	5	-

6.4.5 Modifiziert Jacobisches Koordinatensystem

Sei $E : Y^2Z^4 = X^3 + aXZ^4 + bZ^6$ die Kurvengleichung mit $a, b \in \mathbb{F}_p$ und $4a^3 + 27b^2 \neq 0$. Ein Punkt P auf E wird durch die X-, die Y- und die Z-Koordinate und zusätzlich durch den Wert (aZ^4) durch $P = (X,\ Y,\ Z,\ (aZ^4))$ dargestellt. Diese Koordinaten nennt man Modifiziert Jacobisch, das odifiziert Jacobische System werden wir künftig mit dem Symbol $\mathcal{J}^{\mathcal{M}}$ bezeichnen. Der Punkt im Unendlichen hat die Form $\mathcal{O} \stackrel{\text{def}}{=} (1,\ 1,\ 0,\ 0)$.

Man kann für einen Punkt P die Koordinaten vom Affinen System zum jacobischen System überführen und umgekehrt:

$$
\begin{aligned}
\mathcal{A} \to \mathcal{P} &: \quad X = x,\ Y = y,\ Z = 1,\ (aZ^4) = a \\
\mathcal{P} \to \mathcal{A} &: \quad x = X/Z^2,\ y = Y/Z^3
\end{aligned}
$$

Seien $P = (X_P,\ Y_P,\ Z_P,\ (aZ_P^4))$ und $Q = (X_Q,\ Y_Q,\ Z_Q,\ (aZ_Q^4))$ zwei Punkte auf der Kurve E. Dann lässt sich $R = (X_R,\ Y_R,\ Z_R,\ (aZ_R^4)) = P + Q$ wie folgt

Operation	A	M	S	I
$\mathcal{J}^{\mathcal{M}} + \mathcal{J}^{\mathcal{M}}$	7	13	6	-
$\mathcal{J}^{\mathcal{M}} + \mathcal{A}$	7	9	5	-
$\mathcal{A} + \mathcal{A}$	7	5	4	-
$2 \cdot \mathcal{J}^{\mathcal{M}}$	14	4	4	-
$2 \cdot \mathcal{A}$	14	3	4	-

$$
\begin{aligned}
X_R &= -H^3 - 2U_1H^2 + r^2 \\
Y_R &= -S_1H^3 + r(U_1H^2 - X_R) \\
Z_R &= Z_P Z_Q H \\
(aZ_R^4) &= aZ_R^4 \\
&\text{mit} \\
U_1 &= X_P Z_Q^2 \\
U_2 &= X_Q Z_P^2 \\
S_1 &= Y_P Z_Q^3 \\
S_2 &= Y_Q Z_P^3 \\
H &= U_2 - U_1 \\
r &= S_2 - S_1
\end{aligned}
\tag{6.9}
$$

berechnen:

Für $R = (X_R,\ Y_R,\ Z_P) = 2 \cdot P$ gelten folgende Formeln:

$$
\begin{aligned}
X_R &= T \\
Y_R &= -U + M(S - T) \\
Z_R &= 2Y_P Z_P \\
(aZ_R^4) &= 2U(aZ_P^4) \\
&\text{mit} \\
S &= 4X_P Y_P^2 \\
U &= 8Y_P^4 \\
M &= 3X_1^2 + (aZ_P^4) \\
T &= -2S + M^2
\end{aligned}
\tag{6.10}
$$

Koordinatensystem	Punktaddition				Punktverdopplung			
	A	M	S	I	A	M	S	I
$\mathcal{A}$	6	2	1	1	8	2	2	1
$\mathcal{P}$	7	13	2	-	16	7	5	-
$\mathcal{J}$	7	12	4	-	13	4	6	-
$\mathcal{J}^{\mathcal{C}}$	7	11	3	-	13	5	6	-
$\mathcal{J}^{\mathcal{M}}$	7	13	6	-	13	4	4	-

Tabelle 6.3: Komplexitätsvergleich der Koordinatensysteme

6.4.6 Vergleich der Koordinatensysteme

Es fällt sofort auf, dass sich die Darstellung eines Punktes in den Koordinatensystemen $\mathcal{J}$, $\mathcal{J}^{\mathcal{C}}$ und $\mathcal{J}^{\mathcal{M}}$ nur in der Anzahl der gespeicherten Werte unterscheidet. Die Formeln zur Berechnung einer Addition oder Verdopplung unterscheiden sich nur in der Anzahl der nötigen Körperoperationen. In Tabelle 6.3 stellen wir die Anzahl der Körperoperationen zusammen. Dabei bezeichnet A eine Addition, M eine Multiplikation, S eine Quadrierung und I eine Invertierung in $\mathbb{F}_p$.

Um die Komplexitäten vergleichen zu können ist es nötig, das Laufzeitverhalten der Addition, der Multiplikation, der Quadrierung und der Invertierung zu kennen. Diese werden im Kapitel 8 untersucht.

Wir haben nun eine große Anzahl von Werkzeugen, um eine schnelle EC-Arithmetik für jede Anwendung und jedes Szenario zu erzielen. In Kapitel 4 wurde deutlich, dass die Anforderung Effizienz von Anwendungsszenarien zu Anwendungsszenarien sehr unterschiedlich ist. Daher muss die Wahl des Exponentiationsalgorithmus für jedes dieser Szenarien sowie für jede der möglichen Anwendungen explizit getroffen werden. Das Koordinatensystem wiederum muss in Abhängigkeit eben dieses Exponentiationsalgorithmus gewählt werden.

6.4.7 Gemischte Koordinaten

Verschiedene Koordinatensysteme führen zu einer unterschiedlichen Anzahl von Körperoperationen je Punktaddition und -verdopplung. Während die Jacobi Chudnowsky Koordinaten zum besten Ergebnis bei der Addition führen, sind bei der Punktverdopplung die Modifiziert Jacobischen Koordinaten vorzuziehen. Daher liegt es nahe, sich bei der Punktmultiplikation nicht auf ein Koordinatensystem

Koordinatensystem Überführung			
Operation	M	S	I
$\mathcal{A} \rightarrow \mathcal{P}, \mathcal{J}, \mathcal{J}^{\mathcal{C}}, \mathcal{J}^{\mathcal{M}}$	-	-	-
$\mathcal{J}, \mathcal{J}^{\mathcal{M}} \rightarrow \mathcal{J}^{\mathcal{C}}$	1	1	-
$\mathcal{J} \rightarrow \mathcal{J}^{\mathcal{M}}$	1	2	-
$\mathcal{J}^{\mathcal{C}} \rightarrow \mathcal{J}^{\mathcal{M}}$	1	1	-
$\mathcal{J}^{\mathcal{C}}, \mathcal{J}^{\mathcal{M}} \rightarrow \mathcal{J}$	-	-	-
$\mathcal{P} \rightarrow \mathcal{A}, \mathcal{J}, \mathcal{J}^{\mathcal{C}}, \mathcal{J}^{\mathcal{M}}$	2	-	1
$\mathcal{J}, \mathcal{J}^{\mathcal{C}}, \mathcal{J}^{\mathcal{M}} \rightarrow \mathcal{A}, \mathcal{P}$	3	1	1

Tabelle 6.4: Komplexitäten der Koordinaten-Überführung

zu beschränken.

In diesem Abschnitt wird untersucht, ob die Verwendung verschiedenener Koordinatenarten innerhalb einer Punktmultiplikation Effizienzvorteile bringt. In Tabelle 6.3 auf Seite 94 wurden bereits die Anzahl der einzelnen Körperoperationen zusammengefasst.

Um zu entscheiden, welche Koordinatenkombinationen einen Vorteil bringen können, muss ebenfalls untersucht werden, wieviel das Überführen von einem System in ein anderes kostet.

Die Jacobischen Systeme sind einfach in einander überzuführen. Sie benötigen jeweils höchstens eine Multiplikation und keine, eine oder zwei Quadrierungen. Der interessantere Fall ist das Überführen eines Jacobischen Systems in ein Projektives und umgekehrt. Der einzige Weg besteht darin, über - beziehungsweise in - das Affine System zu gehen. Der Aufwand der einzelnen Überführungen ist in Tabelle 6.4.7 zusammengefasst.

Zu erkennen ist, dass das Mischen von Jacobischen mit Projektiven Koordinaten immer eine Invertierung benötigt und daher vermieden werden sollte. Ebenso ist ein unnötiges Normieren der Koordinaten zu vermeiden, das heißt ein Überführen von projektiven Koordinaten ins Affine System.

In Tabelle 6.5 werden die Komplexitäten der einzelnen Kombinationen für die Punktaddition und in Tabelle 6.6 die für die Punktverdopplung aufgeführt. Die erste Spalte beschreibt die jeweils ausgeführte Punkt-Operation, die zweite Spalte listet die erforderlichen Körperoperationen auf. Da man durch verschiedene Überführungen jede Punktoperation vollständig mit Projektiven oder Jacobischen Koordinaten durchführen kann, ist in der ersten Spalte rechtsbündig angegeben,

welche Art verwendet wurde. “(P)” steht dabei für die Projektiven Formeln und “(J)” für die Jacobischen.

Komplexität der Punktaddition					
—Operation—		—# Körperoperationen—			
$\mathcal{A}+\mathcal{A}\to\mathcal{A}$	(A)	2 M	1 S	6 A	1 I
$\mathcal{A}+\mathcal{A}\to\mathcal{P}$	(P)	5 M	2 S	7 A	
$\mathcal{A}+\mathcal{A}\to\mathcal{J}$	(J)	4 M	2 S	7 A	
$\mathcal{A}+\mathcal{A}\to\mathcal{J}^{\mathcal{C}}$	(J)	5 M	3 S	7 A	
$\mathcal{A}+\mathcal{A}\to\mathcal{J}^{\mathcal{M}}$	(J)	5 M	4 S	7 A	
$\mathcal{A}+\mathcal{P}\to\mathcal{A}$	(P)	12 M	2 S	7 A	1 I
$\mathcal{A}+\mathcal{P}\to\mathcal{P}$	(P)	10 M	2 S	7 A	
$\mathcal{A}+\mathcal{P}\to\mathcal{J}$	(J)	6 M	2 S	7 A	1 I
$\mathcal{A}+\mathcal{P}\to\mathcal{J}^{\mathcal{C}}$	(J)	7 M	3 S	7 A	1 I
$\mathcal{A}+\mathcal{P}\to\mathcal{J}^{\mathcal{M}}$	(J)	7 M	4 S	7 A	1 I
$\mathcal{A}+\mathcal{J}\to\mathcal{A}$	(J)	11 M	4 S	7 A	1 I
$\mathcal{A}+\mathcal{J}\to\mathcal{P}$	(P)	8 M	3 S	7 A	1 I
$\mathcal{A}+\mathcal{J}\to\mathcal{J}$	(J)	8 M	3 S	7 A	
$\mathcal{A}+\mathcal{J}\to\mathcal{J}^{\mathcal{C}}$	(J)	9 M	4 S	7 A	
$\mathcal{A}+\mathcal{J}\to\mathcal{J}^{\mathcal{M}}$	(J)	9 M	5 S	7 A	
$\mathcal{A}+\mathcal{J}^{\mathcal{C}}\to\mathcal{A}$	(J)	10 M	3 S	7 A	1 I
$\mathcal{A}+\mathcal{J}^{\mathcal{C}}\to\mathcal{P}$	(P)	8 M	3 S	7 A	1 I
$\mathcal{A}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}$	(J)	7 M	2 S	7 A	
$\mathcal{A}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}^{\mathcal{C}}$	(J)	8 M	3 S	7 A	
$\mathcal{A}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}^{\mathcal{M}}$	(J)	8 M	4 S	7 A	
$\mathcal{A}+\mathcal{J}^{\mathcal{M}}\to\mathcal{A}$	(J)	11 M	4 S	7 A	1 I
$\mathcal{A}+\mathcal{J}^{\mathcal{M}}\to\mathcal{P}$	(P)	8 M	3 S	7 A	1 I
$\mathcal{A}+\mathcal{J}^{\mathcal{M}}\to\mathcal{J}$	(J)	8 M	3 S	7 A	
$\mathcal{A}+\mathcal{J}^{\mathcal{M}}\to\mathcal{J}^{\mathcal{C}}$	(J)	9 M	4 S	7 A	
$\mathcal{A}+\mathcal{J}^{\mathcal{M}}\to\mathcal{J}^{\mathcal{M}}$	(J)	9 M	5 S	7 A	
$\mathcal{P}+\mathcal{P}\to\mathcal{A}$	(P)	15 M	2 S	7 A	1 I
$\mathcal{P}+\mathcal{P}\to\mathcal{P}$	(P)	13 M	2 S	7 A	
$\mathcal{P}+\mathcal{P}\to\mathcal{J}$	(P)	15 M	2 S	7 A	1 I
$\mathcal{P}+\mathcal{P}\to\mathcal{J}^{\mathcal{C}}$	(P)	15 M	2 S	7 A	1 I
$\mathcal{P}+\mathcal{P}\to\mathcal{J}^{\mathcal{M}}$	(P)	15 M	2 S	7 A	1 I
$\mathcal{P}+\mathcal{J}\to\mathcal{A}$	(J)	13 M	4 S	7 A	2 I
Fortsetzung auf der nächsten Seite					

Fortsetzung der letzten Seite					
Komplexität der Punktaddition					
—Operation—		—# Körperoperationen—			
$\mathcal{P}+\mathcal{J}\to\mathcal{P}$	(P)	13 M	3 S	7 A	1 I
$\mathcal{P}+\mathcal{J}\to\mathcal{J}$	(J)	10 M	3 S	7 A	1 I
$\mathcal{P}+\mathcal{J}\to\mathcal{J}^{\mathcal{C}}$	(J)	11 M	4 S	7 A	1 I
$\mathcal{P}+\mathcal{J}\to\mathcal{J}^{\mathcal{M}}$	(J)	11 M	5 S	7 A	1 I
$\mathcal{P}+\mathcal{J}^{\mathcal{C}}\to\mathcal{A}$	(J)	12 M	3 S	7 A	2 I
$\mathcal{P}+\mathcal{J}^{\mathcal{C}}\to\mathcal{P}$	(P)	13 M	3 S	7 A	1 I
$\mathcal{P}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}$	(J)	9 M	2 S	7 A	1 I
$\mathcal{P}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}^{\mathcal{C}}$	(J)	10 M	3 S	7 A	1 I
$\mathcal{P}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}^{\mathcal{M}}$	(J)	10 M	4 S	7 A	1 I
$\mathcal{P}+\mathcal{J}^{\mathcal{M}}\to\mathcal{A}$	(J)	13 M	4 S	7 A	2 I
$\mathcal{P}+\mathcal{J}^{\mathcal{M}}\to\mathcal{P}$	(P)	13 M	3 S	7 A	1 I
$\mathcal{P}+\mathcal{J}^{\mathcal{M}}\to\mathcal{J}$	(J)	10 M	3 S	7 A	1 I
$\mathcal{P}+\mathcal{J}^{\mathcal{M}}\to\mathcal{J}^{\mathcal{C}}$	(J)	11 M	4 S	7 A	1 I
$\mathcal{P}+\mathcal{J}^{\mathcal{M}}\to\mathcal{J}^{\mathcal{M}}$	(J)	11 M	5 S	7 A	1 I
$\mathcal{J}+\mathcal{J}\to\mathcal{A}$	(J)	15 M	5 S	7 A	1 I
$\mathcal{J}+\mathcal{J}\to\mathcal{P}$	(J)	15 M	5 S	7 A	1 I
$\mathcal{J}+\mathcal{J}\to\mathcal{J}$	(J)	12 M	4 S	7 A	
$\mathcal{J}+\mathcal{J}\to\mathcal{J}^{\mathcal{C}}$	(J)	13 M	5 S	7 A	
$\mathcal{J}+\mathcal{J}\to\mathcal{J}^{\mathcal{M}}$	(J)	13 M	6 S	7 A	
$\mathcal{J}+\mathcal{J}^{\mathcal{C}}\to\mathcal{A}$	(J)	14 M	4 S	7 A	1 I
$\mathcal{J}+\mathcal{J}^{\mathcal{C}}\to\mathcal{P}$	(J)	14 M	4 S	7 A	1 I
$\mathcal{J}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}$	(J)	11 M	3 S	7 A	
$\mathcal{J}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}^{\mathcal{C}}$	(J)	12 M	4 S	7 A	
$\mathcal{J}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}^{\mathcal{M}}$	(J)	12 M	5 S	7 A	
$\mathcal{J}+\mathcal{J}^{\mathcal{M}}\to\mathcal{A}$	(J)	15 M	5 S	7 A	1 I
$\mathcal{J}+\mathcal{J}^{\mathcal{M}}\to\mathcal{P}$	(J)	15 M	5 S	7 A	1 I
$\mathcal{J}+\mathcal{J}^{\mathcal{M}}\to\mathcal{J}$	(J)	12 M	4 S	7 A	
$\mathcal{J}+\mathcal{J}^{\mathcal{M}}\to\mathcal{J}^{\mathcal{C}}$	(J)	13 M	5 S	7 A	
$\mathcal{J}+\mathcal{J}^{\mathcal{M}}\to\mathcal{J}^{\mathcal{M}}$	(J)	13 M	6 S	7 A	
$\mathcal{J}^{\mathcal{C}}+\mathcal{J}^{\mathcal{C}}\to\mathcal{A}$	(J)	13 M	3 S	7 A	1 I
$\mathcal{J}^{\mathcal{C}}+\mathcal{J}^{\mathcal{C}}\to\mathcal{P}$	(J)	13 M	3 S	7 A	1 I
$\mathcal{J}^{\mathcal{C}}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}$	(J)	10 M	2 S	7 A	
$\mathcal{J}^{\mathcal{C}}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}^{\mathcal{C}}$	(J)	11 M	3 S	7 A	
$\mathcal{J}^{\mathcal{C}}+\mathcal{J}^{\mathcal{C}}\to\mathcal{J}^{\mathcal{M}}$	(J)	11 M	4 S	7 A	
$\mathcal{J}^{\mathcal{C}}+\mathcal{J}^{\mathcal{M}}\to\mathcal{A}$	(J)	14 M	4 S	7 A	1 I
Fortsetzung auf der nächsten Seite					

Fortsetzung der letzten Seite					
Komplexität der Punktaddition					
—Operation—		—# Körperoperationen—			
$\mathcal{J}^{\mathcal{C}} + \mathcal{J}^{\mathcal{M}} \to \mathcal{P}$	(J)	14 M	4 S	7 A	1 I
$\mathcal{J}^{\mathcal{C}} + \mathcal{J}^{\mathcal{M}} \to \mathcal{J}$	(J)	11 M	3 S	7 A	
$\mathcal{J}^{\mathcal{C}} + \mathcal{J}^{\mathcal{M}} \to \mathcal{J}^{\mathcal{C}}$	(J)	12 M	4 S	7 A	
$\mathcal{J}^{\mathcal{C}} + \mathcal{J}^{\mathcal{M}} \to \mathcal{J}^{\mathcal{M}}$	(J)	12 M	5 S	7 A	
$\mathcal{J}^{\mathcal{M}} + \mathcal{J}^{\mathcal{M}} \to \mathcal{A}$	(J)	15 M	5 S	7 A	1 I
$\mathcal{J}^{\mathcal{M}} + \mathcal{J}^{\mathcal{M}} \to \mathcal{P}$	(J)	15 M	5 S	7 A	1 I
$\mathcal{J}^{\mathcal{M}} + \mathcal{J}^{\mathcal{M}} \to \mathcal{J}$	(J)	12 M	4 S	7 A	
$\mathcal{J}^{\mathcal{M}} + \mathcal{J}^{\mathcal{M}} \to \mathcal{J}^{\mathcal{C}}$	(J)	13 M	5 S	7 A	
$\mathcal{J}^{\mathcal{M}} + \mathcal{J}^{\mathcal{M}} \to \mathcal{J}^{\mathcal{M}}$	(J)	13 M	6 S	7 A	

Tabelle 6.5: Komplexität der Punktaddition mit gemischten Koordinaten

Deutlich ist in Tabelle 6.5 zu sehen, dass Affine Koordinaten einen großen Effizienzvorteil bringen, wenn sie als Summand auftreten, das Ergebnis jedoch projektiv ist. Es liegt daher auf der Hand, Punkte, die - wie beim LimLee-Verfahren - im Voraus berechnet werden können, gleich affin abzuspeichern.

Das Modifiziert Jacobische Koordinatensystem bringt bei der Addition keine Vorteile gegenüber dem Jacobischen oder dem Chudnowsky Jacobischen System. Während zur Ergebnisberechnung zwei Quadrierungen und eine Multiplikation zusätzlich nötig sind, wird in der Grundrechnung nichts eingespart.

Das Chudnowsky Jacobische System benötigt ebenfalls mehr Körperoperationen zur Berechnung von Z^2 und Z^3, nämlich eine Quadrierung und eine Multiplikation. Dieser Aufwand wird jedoch eingespart, wenn einer der Summanden schon in dieser Darstellung vorliegt. Gilt dies für beide Summanden, werden sogar eine Quadrierung und eine Multiplikation eingespart.

Mittels Tabelle 6.6 können die gleichen Untersuchungen bezüglich der Punktverdopplung angestellt werden:

Komplexität der Punktverdopplung					
—Operation—		—# Körperoperationen—			
$2 \cdot \mathcal{A} \to \mathcal{A}$	(J)	1 M	2 S	8 A	1 I
$2 \cdot \mathcal{A} \to \mathcal{P}$	(P)	5 M	4 S	15 A	
$2 \cdot \mathcal{A} \to \mathcal{J}$	(J)	2 M	4 S	13 A	
Fortsetzung auf der nächsten Seite					

Fortsetzung der letzten Seite					
Komplexität der Punktverdopplung					
—Operation—		—# Körperoperationen—			
$2 \cdot \mathcal{A} \to \mathcal{J}^{\mathcal{C}}$	(J)	3 M	5 S	13 A	
$2 \cdot \mathcal{A} \to \mathcal{J}^{\mathcal{M}}$	(J)	3 M	4 S	14 A	
$2 \cdot \mathcal{P} \to \mathcal{A}$	(P)	9 M	5 S	15 A	1 I
$2 \cdot \mathcal{P} \to \mathcal{P}$	(P)	7 M	5 S	15 A	
$2 \cdot \mathcal{P} \to \mathcal{J}$	(J)	3 M	4 S	13 A	1 I
$2 \cdot \mathcal{P} \to \mathcal{J}^{\mathcal{C}}$	(J)	4 M	5 S	13 A	1 I
$2 \cdot \mathcal{P} \to \mathcal{J}^{\mathcal{M}}$	(J)	4 M	4 S	14 A	1 I
$2 \cdot \mathcal{J} \to \mathcal{A}$	(J)	7 M	7 S	13 A	1 I
$2 \cdot \mathcal{J} \to \mathcal{P}$	(P)	8 M	5 S	15 A	1 I
$2 \cdot \mathcal{J} \to \mathcal{J}$	(J)	4 M	6 S	13 A	
$2 \cdot \mathcal{J} \to \mathcal{J}^{\mathcal{C}}$	(J)	5 M	7 S	13 A	
$2 \cdot \mathcal{J} \to \mathcal{J}^{\mathcal{M}}$	(J)	5 M	6 S	14 A	
$2 \cdot \mathcal{J}^{\mathcal{C}} \to \mathcal{A}$	(J)	7 M	6 S	13 A	1 I
$2 \cdot \mathcal{J}^{\mathcal{C}} \to \mathcal{P}$	(J)	7 M	6 S	13 A	1 I
$2 \cdot \mathcal{J}^{\mathcal{C}} \to \mathcal{J}$	(J)	4 M	5 S	13 A	
$2 \cdot \mathcal{J}^{\mathcal{C}} \to \mathcal{J}^{\mathcal{C}}$	(J)	5 M	6 S	13 A	
$2 \cdot \mathcal{J}^{\mathcal{C}} \to \mathcal{J}^{\mathcal{M}}$	(J)	5 M	5 S	14 A	
$2 \cdot \mathcal{J}^{\mathcal{M}} \to \mathcal{A}$	(J)	6 M	5 S	13 A	1 I
$2 \cdot \mathcal{J}^{\mathcal{M}} \to \mathcal{P}$	(J)	6 M	5 S	13 A	1 I
$2 \cdot \mathcal{J}^{\mathcal{M}} \to \mathcal{J}$	(J)	3 M	4 S	13 A	
$2 \cdot \mathcal{J}^{\mathcal{M}} \to \mathcal{J}^{\mathcal{C}}$	(J)	4 M	5 S	13 A	
$2 \cdot \mathcal{J}^{\mathcal{M}} \to \mathcal{J}^{\mathcal{M}}$	(J)	4 M	4 S	14 A	

Tabelle 6.6: Komplexität der Punktverdopplung mit gemischten Koordinaten

Auch für die Punktverdopplung gilt, dass das Mischen von Jacobischen mit Projektiven Koordinaten vermieden werden sollte, da jede Überführung eine Invertierung beinhaltet. Dagegen bringt die Verwendung von Affinen Summanden auch hier einen großen Effizienzvorteil.

Die Punktverdopplung im Jacobischen Koordinatensystem ist um drei Multiplikationen, eine Quadrierung und zwei Additionen günstiger als die Punktverdopplung im Projektiven System. Die Chudnowsky Darstellung spart nur dann eine Quadrierung ein, wenn schon der Summand in dieser Darstellung vorliegt und das Ergebnis rein Jacobisch sein soll. Sonst kostet sie bis zu einer Multiplikation und einer Quadrierung mehr.

Anders die Modifiziert Jacobischen Koordinaten: Das Berechnen von aZ^4 kostet nur eine Multiplikation und eine Addition zusätzlich: Liegen die Summanden bereits in dieser Darstellung vor, werden eine Multiplikation und zwei Quadrierungen eingespart.

Zusammenfassend muss zur Optimierung folgendes beachtet werden:

- Affine Summanden und nicht-Affine Resultate sind am effizientesten zu berechnen.
- Die Überführung von Projektiven in Jacobische, von Jacobischen in Projektive und von projektiven Koordinaten in Affine kostet mindestens je eine Invertierung. Invertierungen sind sehr teuer und sollten vermieden werden.
- Chudnowsky Jacobische Koordinaten eignen sich besonders für die Hintereinanderausführung von Punktadditionen.
- Ist bei einer Punktaddition einer der zwei Summanden affin, benötigt diese Operation mit Jacobischen Koordinaten ebenso wenig Körperoperationen wie mit Chudnowsky Jacobischen Koordinaten.
- Modifiziert Jacobische Koordinaten eignen sich am besten zur Hintereinanderausführung von Punktverdopplungen.
- Die Hintereinanderausführung von Punktadditionen ist mit Jacobischen Koordinaten um eine Körpermultiplikationen günstiger und zwei Quadrierungen teurer als die mit Projektiven Koordinaten.
- Ist einer der zwei Summanden einer Punktaddition affin, ist die Hintereinanderausführung von Punktadditionen mit Jacobischen Koordinaten um zwei Körpermultiplikationen günstiger und nur um eine Körperquadrierung teurer als die mit Projektiven Koordinaten.
- Die Hintereinanderausführung von Punktverdopplungen mit Jacobischen Koordinaten ist pro Verdopplung um drei Körpermultiplikationen und einer Körperaddition günstiger und nur um eine Körperquadrierung teurer als die mit Projektiven Koordinaten.

Mit diesen Erkenntnissen können nun die in den Abschnitten 6.1 und 6.2 vorgestellten Verfahren optimiert werden. Im nächsten Kapitel wird dazu eine Strategie vorgestellt und in den nachfolgenden Kapiteln, 8 und 9, durchgeführt. In Kapitel 10 werden die experimentellen Ergebnisse der optimierten Verfahren vorgetellt.

Kapitel 7

Strategie zur Optimierung der Punktmultiplikation

Wie in den vorangegangen Kapiteln zu sehen, gibt es viele verschiedene Möglichkeiten eine Punktmultiplikation auszuführen. Neben den Algorithmen müssen Fenstergrößen und Koordinatensystem gewählt werden. Auch könnte die Kombination verschiedener Koordinatensysteme weitere Effizienzvorteile bringen.

7.1 Problemstellung

Ein Weg, das schnellste Verfahren mit der optimalen Koordinatenbesetzung und Fenstergröße zu bestimmen, könnte das Implementieren aller Verfahren mit allen möglichen Kombinationen sein, um dann ihre Laufzeit zu testen. Dabei treten folgende Probleme auf:

- Jede Methode zur Punktmultiplikation ist eine Zusammensetzung aus Punktadditionen und -verdopplungen. Jede dieser einzelnen Operationen müsste mit jeder Koordinatenkombinationen durchgespielt werden. Pro Addition gibt es 75 mögliche Kombinationen und pro Punktverdopplung nochmals 25. Das macht im besten Fall, im Fall der Exponentiation mit ± 1-Ketten, alleine 75 Durchläufe, im nächst besten Fall, dem Fixed-Size-Sliding-Window, 75*25 Durchläufe.

- Von nur 3 Methoden abgesehen wird darüber hinaus noch eine Fenstergröße

benötigt. Die optimale ist wiederum abhängig von der Bitlänge des Exponenten und muss daher für jede dieser Bitlängen neu gefunden werden.

- Die Laufzeit ist - unabhängig von diesen vielen Möglichkeiten - zusätzlich sehr stark vom Exponenten abhängig und kann so bei wiederholten Durchläufen mit unterschiedlichen Exponenten stark schwanken. Wünschenswert ist der Erwartungswert der Laufzeit jeder der einzelnen Möglichkeiten. Er würde zumindest eine Aussage zulassen über die Laufzeit, die bei sehr vielen Durchläufen im Schnitt auftreten würde.

Es liegt auf der Hand, dass rein experimentelle Tests über alle Möglichkeiten nicht möglich sind, sie sind zu aufwendig. Und höchstwahrscheinlich nicht repräsentativ.

7.2 Verfahrensweise

Daher wurde eine Strategie entwickelt, die in Abhängigkeit von der Bitlänge des Exponenten, der Multiplikationsmethode, der Koordinatenkombination und der Fenstergröße den Erwartungswert der Laufzeit berechnet. Diese Strategie wurde ausimplementiert und für diese Arbeit in den folgenden Kapiteln verwendet. Dieses Programm ordnet darüber hinaus je Methode und Bitlänge die möglichen Kombinationen ihrer Effizienz nach, so dass nur noch die pro Methode besten Kombinationen untereinander verglichen werden müssen.

Diese Strategie mit ihrem Programm wird in diesem Kapitel vorgestellt. Die Ausführung folgt in den Kapiteln 8 und 9. Experimentelle Ergebnisse, anhand derer verifiziert werden kann, dass die Strategie greift, werden in Kapitel 10 vorgestellt.

Wie schon erwähnt, hängt die Laufzeit einer Punktmultiplikation von verschiedenen Faktoren ab, die hier aufgelistet werden:

- Plattform: Rechnerleistung, Javaversion, virtuelle Maschine, nebenläufige Prozesse
- Größe des zugrunde liegenden Körpers $\mathbb{F}_p$
- Ordnung der Punktegruppe $E(\mathbb{F}_p)$
- Verwendung vorberechneter Punkte
- Wahl des Koordinatensystems oder mehrerer Koordinatensysteme

- Wahl der Multiplikationsmethode
- gegebenfalls Wahl der Fenstergröße w

Der erste Faktor, abgesehen von den nebenläufigen Prozessen, steht im Allgemeinen fest. Die Körpergröße und die Ordnung werden von der Sicherheitsanforderung bestimmt. Die letzten vier Punkte beeinflussen direkt die Laufzeit und können bestmöglich gewählt werden. Folgende Strategie berechnet die optimale Kombination von Multiplikationsmethode, Fenstergröße w und Koordinatensystem:

Input: Körpergröße p, Gruppenordnung r, Anwendungsszenario: das heißt, die Möglichkeit der Verwendung vorberechneter Punkte.

Output: Optimale Kombination von Koordinatensystem, Multiplikationsmethode und Fenstergröße, inklusive Laufzeitabschätzung

1. Messen der jeweiligen Laufzeit der einzelnen Körperoperationen Addition (A), Multiplikation (M), Quadrierung (S) und Invertierung (I) auf der festgelegten Plattform im Körper $\mathbb{F}_p$.

2. Unter Verwendung der Zeiten aus Punkt 1. und den Tabellen 6.5 und 6.6 kann hochgerechnet werden, wie lange jede einzelne Punktaddition und -verdopplung in jedem der Koordinatensysteme benötigt.

3. Unter Verwendung der Zeiten aus Punkt 2. kann mittels der Koplexitätsformeln der einzelnen Multiplikationsalgorithmen in den Tabellen 6.1 und 6.2 hochgerechnet werden, wie lange jeder einzelne der Algorithmen im Abhängigkeit der Fenstergöße w benötigt.

4. Diese hochgerechneten Laufzeiten werden für jede Ausprägung aller Punktmultiplikationsmethoden in aufsteigender Folge geordnet. Die oberste Ausprägung ist die, die auf der gewählten Plattform für die gewählte Punktordnung und Körpergröße optimal ist.

Das zur Strategie implementierte Programm kann beliebig oft auf verschiedenen Plattformen und unterschiedlichen EC Parametern ausgeführt werden. Die Strategie selbst ist natürlich auch auf andere Programmiersprachen übertragbar.

7.3 Programm zur Komplexitätsminimierung

In diesem Abschnitt wird das Programm zur Durchführung der Strategie vorgestellt.

Messen der Zeit der einzelnen Körperoperationen

Der erste Schritt ist das Messen der Zeit einer Addition, Multiplikation, Quadrierung und Invertierung im Körper $\mathbb{F}_p$. Das Paket `javatests.biginteger-tests` umfasst die dazu notwendigen Klassen `Init`, `Add`, `Mult`, `Square` und `Inv`. Sei im Folgenden p die zugrunde liegende Körpergröße und r die Punktordnung.

ComputeBigInts berechnet 5000 zufällige Primzahlen mit Bitlänge p, sowie 5000 zufällige Zahlen der gleichen Bitlänge, die kleiner sind als jede der Primzahlen, und speichert sie in dem File `/src/BigIntegers/Mod`p`.dat` beziehungsweise `/src/BigIntegers/Big`p`.dat` ab. Dabei wird p durch die Zahl p ersetzt. Für p können die Bitlängen eingesetzt werden, die der Bitlänge der Gruppenordnung $r = \operatorname{ord} G$ entspricht. So wird die Zeit der Operationen im Körper $\mathbb{F}_p$ gemessen, über dem später auch die Punktmultiplikation stattfindet.

Init liest diese BigInteger aus diesen Dateien ein und misst die dafür notwendige Zeit.

Add, Mult, Square und Inv lesen ebenfalls jeweils diese BigInteger ein. So werden die folgenden Tests unter den gleichen Voraussetzungen gemacht:

`Add` addiert in einer doppelten Schleife jede BigInteger mit jeder BigInteger des Files `Big`p`.dat`. Das sind insgesamt $5000 \cdot 5000$ Additionen.

`Mult` multipliziert in einer doppelten Schleife jede BigInteger mit jeder BigInteger des Files `Big`p`.dat` und reduziert das Ergebnis mit jeder der Primzahlen aus der Datei `Mod`p`.dat`. Das sind $5000 \cdot 5000$ modulare Multiplikationen.

`Square` multipliziert in einer doppelten Schleife jede der BigIntegers des Files `Big`p`.dat` mit sich selbst und reduziert das Ergebnis mit jeder der Primzahlen aus der Datei `Mod`p`.dat`. Das sind $5000 \cdot 5000$ modulare Quadrierungen.

`Inv` invertiert in einer doppelten Schleife jede der BigIntegers des Files Big*p*.dat modulo jede der Primzahlen aus der Datei Mod*p*.dat. Das sind $5000 \cdot 5000$ modulare Invertierungen.

Jede der Klassen startet also unter den gleichen Voraussetzungen. Die Zeit, die jede dieser Klassen benötigt, wird mit dem Unix-Tool *time* gemessen. Mit diesem Tool kann die Prozessorzeit des Programms gemessen werden. So wird das Ergebnis nicht durch nebenläufige Prozesse verfälscht. Der Einfachheit halber existiert ein Shellscript, Biginteger.sh, das die Tests ausführt. Das Ergebnis der Klasse `Init` wird von dem der Klassen `Add`, `Mult`, `Square` und `Inv` abgezogen und die Ausgabe in eine Datei umgeleitet.

Hochrechnung der Zeit zur Punktaddition und -verdopplung

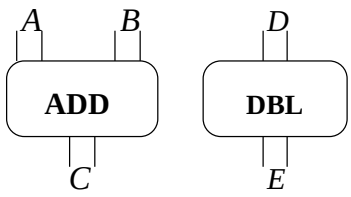
Anhand der Tabellen 6.5 und 6.6 und der gemessenen Laufzeit der einzelnen Körperoperationen aus dem vorangegangenen Abschnitt lässt sich berechnen, wie lange eine Punktoperation mit beliebigen Koordinaten benötigt. Diese Berechnung geht in das nächste Testpaket des nächsten Strategiepunkts ein.

Hochrechnen der Zeit der Punktmultiplikation

Das letzte Testpaket heißt `javatests.komplex`. Es beinhaltet die Klassen, die die zu erwartenden Laufzeiten der einzelnen Multiplikationsmethoden berechnet. Die Ergebnisse werden in absteigender Folge in eine Datei ausgegeben. Es gibt für jede vorgestellte Methode aus den Abschnitten 6.1 und 6.2 eine Klasse. Jede von ihnen hat Zugriff auf die bisher gemessenen Zeiten der Körperoperationen und außerdem auf die Anzahl der einzelnen Körperoperationen, die jede Koordinatensystemkombination für eine Punktaddition und -verdopplung benötigt.

Jede der Klassen, die die Komplexität einer Methode berechnen, multipliziert nun die Anzahl der benötigten Punktadditionen und -verdopplungen mit der jeweils benötigten Zeit. Dies wird für jedes Koordinatensystem sowie für jede Koordinatensystem-Kombinationen und eine Reihe von verschiedenen Fenstergrößen durchgeführt. Die Ergebnisse werden zwischengespeichert, sortiert und zum Schluss in eine Datei ausgegeben.

Eine Punktmultiplikation besteht aus Punktadditionen und Punktverdopplungen. Wie in den Tabellen 6.5 und 6.6 zusammengefasst, können



die Summanden und das Resultat bei beiden Punktoperationen unterschiedliche Koordinatensysteme besitzen. Um alle Kombinationen zu erreichen ist es hilfreich, sich diese Operationen als Maschinen vorzustellen, wie in der Abbildung rechts dargestellt. Die erste Maschinenart, links, addiert zwei verschiedene Punkte und gibt das Ergebnis zurück, die zweite, rechts, verdoppelt einen Punkt und gibt das Ergebnis zurück. Dabei stehen A, B und D als Platzhalter für die Koordinatensysteme der Summanden, und C und E als Platzhalter für die Ergebnisse. Es gilt also: $A, B, C, D, E \in \{\mathcal{A}, \mathcal{P}, \mathcal{J}, \mathcal{J}^{\mathcal{C}} \mathcal{J}^{\mathcal{M}}\}$.

7.4 Erläuterung an einem Beispiel

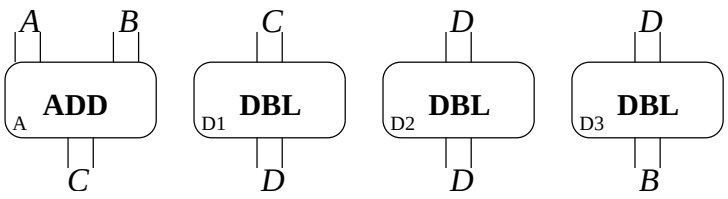
Anhand eines Codefragments wird im Folgenden beschrieben, wie die Komplexität eines Algorithmus in Abhängigkeit von der Laufzeit der einzelnen Körperoperationen, den Koordinatensystemen $\mathcal{A}$, $\mathcal{P}$, $\mathcal{J}$, $\mathcal{J}^{\mathcal{C}}$ und $\mathcal{J}^{\mathcal{M}}$ und der Fenstergröße w berechnet werden kann. Als Beispiel soll die Punktmultiplikation $n \cdot P$ mit $n = 2895_{10} = 101101001111_2$ ausgeführt werden. Die Punkte P_1 bis P_7 mit $P_i = i \cdot P$, $1 \leq i \leq 7$ liegen vorberechnet vor.

$R \leftarrow \mathcal{O}, i \leftarrow n - 1$
$t = (i + 1)\%3$
`if` $(t \neq 0)$
  `if` $(k_{n-1}k_{n-2} \ldots k_{n-t+1})_2 \neq 0$ `then`
    $R \leftarrow R + P_{(k_{n-1}k_{n-2} \ldots k_{t+1})_2}$,
    $i \leftarrow i - t$
`while` $i \geq 0$
  `for` $j = 0$ `to` 2
    $R \leftarrow 2 \cdot R$
  `if` $(k_i k_{i-1} k_{i-2})_2 \neq 0$ `then`
    $R \leftarrow R + P_{(k_i k_{i-1} k_{i-2})_2}$
    $i \leftarrow i - 3$
`return` R

$R = \mathcal{O}, t = 0$
$i = 11$:
  $j = 0$: $R = \mathcal{O}$, $j = 1$: $R = \mathcal{O}$, $j = 2$: $R = \mathcal{O}$
  $R = R + P_{101_2} = 5 \cdot P$

$i = 8$:
$j = 0$: $R = 10 \cdot P$, $j = 1$: $R = 20 \cdot P$, $j = 2$: $R = 40 \cdot P$
$R = R + P_{101_2} = 40 \cdot P + 5 \cdot P = 45 \cdot P$
$i = 5$:
$j = 0$: $R = 90 \cdot P$, $j = 1$: $R = 180 \cdot P$, $j = 2$: $R = 360 \cdot P$
$R = R + P_{001_2} = 360 \cdot P + P = 361 \cdot P$
$i = 2$:
$j = 0$: $R = 722 \cdot P$, $j = 1$: $R = 1444 \cdot P$, $j = 2$: $R = 2888 \cdot P$
$R = R + P_{111_2} = 2888 \cdot P + 7 \cdot P = 2895 \cdot P$

Mit diesem Codefragment folgen jeder Punktaddition immer mindestens 3 Punktverdopplungen. Es liege der Punkt P in den Koordinaten der Art A vor. Nach Abbildung auf Seite 105 erhält die Additionsmaschine einen weiteren Punkt der Art B und gibt das Ergebnis in den Koordinaten der Art C zurück. Die anschließende Punktverdopplung bekommt daher offensichtlich einen Punkt der Art C. Daraus folgt, dass $C = D$ ist. Die Maschine zur Punktverdopplung gibt das Ergebnis in Koordinaten der Art E zurück. Das Ergebnis ist jedoch gleichzeitig der neue Eingabewert für die zweite Punktverdopplung. Daher muss laut der Abbildung $C = D = E$ sein. Nach der dritten Punktverdopplung kann wieder eine Addition mit der Additionsmaschine folgen. Die Ausgabe der Punktverdopplungsmaschine ist der zweite Summand für die Additionsmaschine. Daraus folgt wiederum, dass $C = D = E = B$ ist. Wir haben es in diesem Fall also nur mit zwei Koordinatensystemen A und B zu tun.

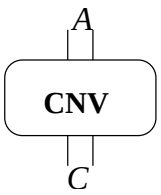


Aus den Tabellen 6.5 und 6.6 ist bekannt, dass hintereinander auszuführende Punktverdopplungen am effizientesten mit Modifiziert Jacobischen Koordinaten ausgeführt werden, diese jedoch die schlechteste Wahl für die Punktaddition darstellen. Um dennoch diesen Vorteil bei den Verdopplungen nutzen zu können, werden neue Maschinen eingeführt: Die Additionsmaschine bleibt wie sie ist, es werden drei neue Verdopplungsmaschinen eingeführt. Maschine D1 bekommt das Ergebnis der Maschine A in Koordinaten der Art C, verdoppelt dieses und gibt das Ergebnis in Koordinaten der Art D zurück. Maschine D2 bleibt anschließend mit seiner Verdopplung in diesem System und Maschine D3 führt mit seiner Verdopplung den Punkt wieder in Koordinaten der Art B zurück, wo er als Parameter wieder in

Maschine A einfließen kann.

Die Multiplikation $n \cdot P$ mit $n = 2895_{10} = 101101001111_2$ wird durch diese Maschinen wie folgt ausgeführt:

$R = \mathcal{O}, t = 0$
$i = 11$:
  / $j = 0$: $R = \mathcal{O}$, $j = 1$: $R = \mathcal{O}$, $j = 2$: $R = \mathcal{O}$
  A: $R = R + P_{101_2} = 5 \cdot P$
$i = 8$:
  D1: $j = 0$: $R = 10 \cdot P$,
  D2: $j = 1$: $R = 20 \cdot P$,
  D3: $j = 2$: $R = 40 \cdot P$
  A: $R = R + P_{101_2} = 40 \cdot P + 5 \cdot P = 45 \cdot P$
$i = 5$:
  D1: $j = 0$: $R = 90 \cdot P$,
  D2: $j = 1$: $R = 180 \cdot P$,
  D3: $j = 2$: $R = 360 \cdot P$
  A: $R = R + P_{001_2} = 360 \cdot P + P = 361 \cdot P$
$i = 2$:
  D1: $j = 0$: $R = 722 \cdot P$,
  D2: $j = 1$: $R = 1444 \cdot P$,
  D3: $j = 2$: $R = 2888 \cdot P$
  A: $R = R + P_{111_2} = 2888 \cdot P + 7 \cdot P = 2895 \cdot P$



Mit den 4 Maschinen auf Seite 107 kann das gegebene Codefragment also ausgeführt werden. Zwei kleine Ausnahmen gibt es: Die ersten drei Punktverdopplungen sind von der Form $\mathcal{O} = \mathcal{O}$ und die erste Addition ist von der Form $R = \mathcal{O} + P_{101_2}$. Diese Verdopplungen können ignoriert werden, da sie keine echten Punktoperationen darstellen. Die Addition ist eigentlich nur eine Konvertierung des Punktes von Koordinaten der Art A in Koordinaten der Art C. Durch Ergänzen mit einer Maschine, die einen Punkt mit Koordinaten der Art A in Koordinaten C konvertiert, wie links abgebildet, kann dieses Problem behoben werden.

In der obigen Berechnung von $2895 \cdot P$ werden die Konvertierungsmaschine einmal und die Maschinen A, D1, D2 und D3 je dreimal durchlaufen. Das Programm zur Hochrechnung der Laufzeit dieses Codefragments würde folgende Formeln enthalten:

$$
\begin{array}{lcll}
1 & \times & R^C \leftarrow P_{-}^A & + \\
(\lceil \frac{n}{3} \rceil - 1) & \times & R^C \leftarrow R^B + P_{-}^A & + \\
(n - 3 - 2 \cdot (\lceil \frac{n}{3} \rceil - 1)) & \times & R^D \leftarrow 2 \cdot P_{-}^D & + \\
(\lceil \frac{n}{3} \rceil - 1) & \times & R^D \leftarrow 2 \cdot P_{-}^C & + \\
(\lceil \frac{n}{3} \rceil - 1) & \times & R^D \leftarrow 2 \cdot P_{-}^B. &
\end{array}
$$

Das Programm hat Zugriff auf die Laufzeit jeder dieser Operationen mit jeder der Belegungen der Platzhalter A bis D und kann so die hochgerechnete Laufzeit dieses Fragments für jede der Koordinaten-Kombinationen berechnen und sie der Größe nach sortieren. Die Kombination mit dem kleinsten Ergebnis ist den anderen dann vorzuziehen.

Im Folgenden werden die Maschinen durch Formeln ersetzt. Wir zählen die Operationen auf, mit denen man jeden Algorithmus zur Punktmultiplikation zusammensetzen kann:

1. $R^A \leftarrow 2 \cdot R^A$: Punktverdopplungen in ein und demselben Koordinatensystem
2. $R^B \leftarrow 2 \cdot R^C$: Punktverdopplungen von einem in ein anderes Koordinatensystem
3. $R^D \leftarrow R^D + R^D$: Punktadditionen in ein und demselben Koordinatensystem
4. $R^G \leftarrow R^E + R^F$: Punktadditionen mit unterschiedlichen Koordinatensystemen (möglich: $G = E$ oder $G = F$ oder $E = F$)
5. $R^I \leftarrow R^H$: Konvertieren eines Punktes von einem in ein anderes Koordinatensystem.

Die Buchstaben A bis I sind als Platzhalter für verschiedene Koordinatensysteme zu lesen. Mit diesen Operationen können alle Punktmultiplikationsmethoden zusammen gesetzt werden. Dabei werden folgende Regeln befolgt:

- Bei jedem Wechsel von Punktaddition zu Punktverdopplung und umgekehrt wird ein Koordinatenwechsel zugelassen.
- Aufeinanderfolgende Punktverdopplungen werden im selben Koordinatensystem ausgeführt.
- Aufeinanderfolgende Punktadditionen werden im selben Koordinatensystem ausgeführt.

- Am Ende jedes Algorithmus zur Punktmultiplikation wird der Ergebnispunkt normiert, das heißt, der Ergebnispunkt wird ins Affine System überführt.

Im Folgenden führen wir diese Strategie anhand der in Kapitel 6 vorgestellten Algorithmen vor. Die Körpergrößen werden zusammen mit der Punktordnung so gewählt, dass sie jeweils einmal 140, 160, 192, 197 und 272 Bit lang sind. Diese Wahl beruht auf den Empfehlungen des BSI [fSidI02] und von Lenstra und Verheul [LV99] (siehe auch im Abschnitt 4.2) begründet.

Kapitel 8

Laufzeitverhalten der Körperoperationen

Wie schon beschrieben, werden die Körperoperationen mit der Klasse `BigInteger` ausgeführt. Jede Addition wird mit der Methode `BigInteger add(BigInteger)` (A) und jede Invertierung mit `BigInteger modInverse(BigInteger)` (I) ausgeführt. Dagegen wird jede Multiplikation und jede Quadrierung mit der Hintereinanderausführung der Methoden `BigInteger multiply(BigInteger)` und `BigInteger mod(BigInteger)` (M) ausgeführt. In diesem Kapitel wird das Laufzeitverhalten dieser Operationen untersucht. Alle folgenden Tests wurden unter Linux mit Java Version „1.4.1_02“ auf einem AMD Athlon mit 1.3 GHz und 256 MB Arbeitsspeicher gemessen. Auf einer anderen Plattform, unter anderen Voraussetzung, werden die Ergebnisse sicher anders ausfallen. Der Versuchsaufbau wurde bereits im Kapitel 7 beschrieben. Das Ergebnis wird in den drei Graphen 8.1, 8.2 und 8.3 dargestellt. Aus ihnen lässt sich Folgendes ablesen:

- Im Verhältnis zu der Multiplikation, Quadrierung und Invertierung ist die Laufzeit für eine Addition verschwindend gering (Abbildung 8.1).

- Da die Klasse `BigInteger` keine eigene Methode zur Quadrierung anbietet, ist die Kurve der Multiplikation von der der Quadrierung kaum zu unterscheiden (Abbildung 8.1 und 8.2). Dies gilt natürlich nur für die in dieser Arbeit verwendeten Referenzimplementierung. Bei anderen Implementierungen können sehr wohl größere Unterschiede auftreten.

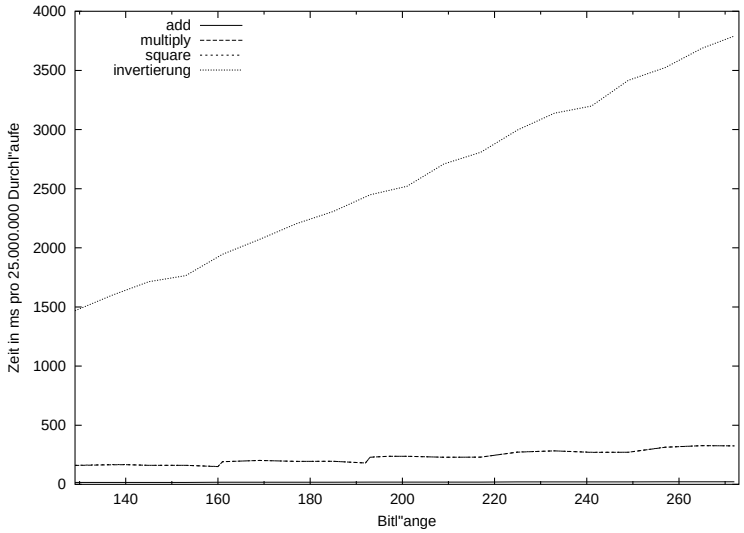


Abbildung 8.1: Laufzeit der vier Körperoperationen Körperoperationen: Addition, Multiplikation, Quadrierung und Invertierung

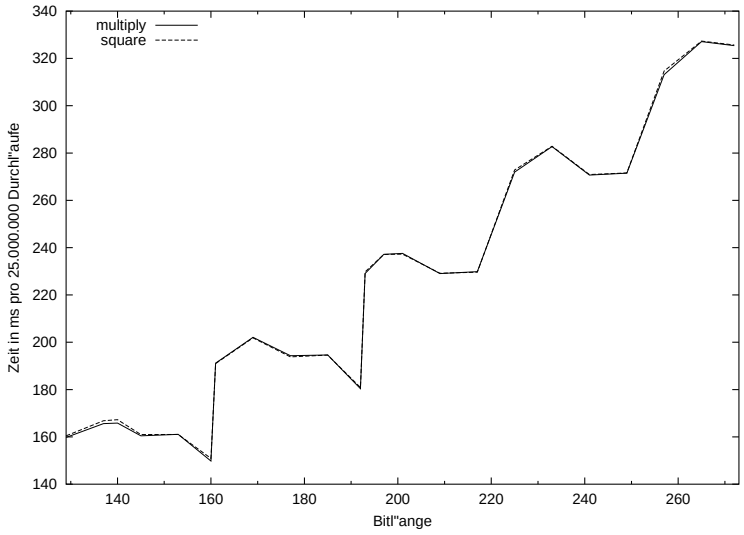


Abbildung 8.2: Laufzeit der Körpermultiplikation und -quadrierung

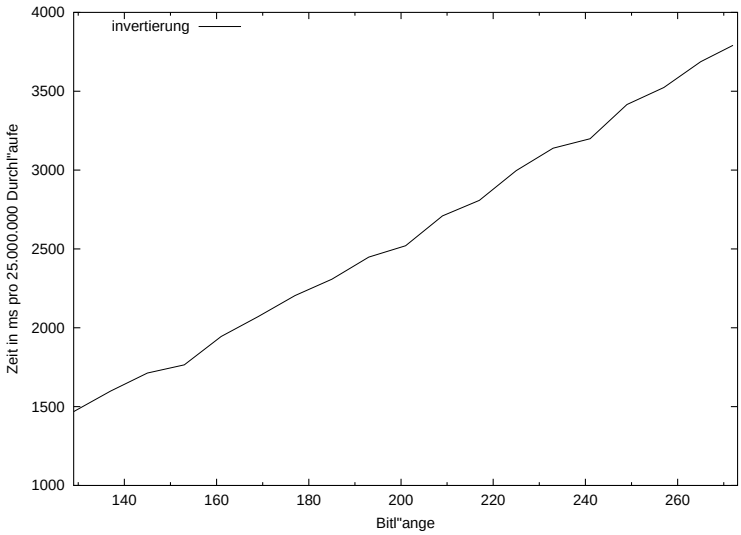


Abbildung 8.3: Laufzeit der Körperinvertierung

Bitlänge	Operation			
	A	M	S	I
140	0.000615	0.006964	0.006978	0.067372
160	0.000847	0.006452	0.006363	0.078098
192	0.000851	0.007468	0.007525	0.099296
197	0.000526	0.009779	0.009786	0.102068
272	0.000754	0.013272	0.013302	0.1576

Tabelle 8.1: Laufzeit von je 1Körperoperation in ms

- Das Verhältnis Invertierung : Multiplikation beziehungsweise Invertierung : Quadrierung schwankt (Abbildung 8.1).
- Die Kurve der Invertierung steigt nahezu linear (Abbildung 8.3).

Tabelle 8.1 führt die Laufzeiten exemplarisch für verschiedene Bitlängen auf. Sie werden in Sekunden pro 25.000.000 Durchläufe angegeben.

Kapitel 9

Hochgerechnete Laufzeiten der Multiplikationsmethoden

In diesem Kapitel werden die Zeiten präsentiert, die sich rein rechnerisch mit den Laufzeit-Ergebnissen aus Kapitel 8 und der Strategie aus Kapitel 7 pro Punktmultiplikationsmethode unter Anwendung der Tabellen 6.5 und 6.6 ergeben. An diesen Ergebnissen wird man erkennen, dass die Wahl des Koordinatensystems sehr wohl zeitkritisch ist. Unter den vielen Kombinationen gibt es jedoch viele, die zu nahezu den gleichen Ergebnissen führen. Die Differenz liegt hierbei im $1/100$ Millisekunden-Bereich.

Für jede der vorgestellten Methoden werden in Kürze die Komplexitätsformeln der Abschnitte 6.1 und 6.2 präzisiert, damit auch die Komplexitäten der Algorithmen berechnetet werden können, wenn mehrere Koordinatensysteme innerhalb eines Algorithmus verwenden werden. Da es mit der Möglichkeit der Verwendung von verschiedenen Koordinatensystemen schon ohne Wahl der Fenstergröße w für einen Algorithmus zwischen 75 Permutation bei der Exponentition mit ± 1-Ketten und 1250 Permutationen bei der Exponentiation mit LimLee gibt, werden nur Auszüge der Ergebnisse gezeigt, nämlich die ersten acht besten Ergebnisse und zusätzlich die, die mit nur einem Koordinatensystem entstehen. Dabei wird deutlich werden, dass das Affine Koordinatensystem fast immer die schlechteste Lösung ist, gefolgt von dem Projektiven Koordinatensystem.

Die Punktmultiplikationsmethoden werden in die folgenden vier Gruppen unterteilt: Einfache Punktmultiplikation ohne Verwendung vorberechneter Punkte (Abk: ovP), einfache Punktmultiplikation unter Verwendung vorberechneter Punk-

te (Abk: uvP), mehrfache Punktmultiplikation ohne Verwendung vorberechneter Punkte und mehrfache Punktmultiplikation unter Verwendung vorberechneter Punkte.

9.1 Einfache Punktmultiplikationsmethoden ohne Verwendung vorberechneter Punkte

In diesem Abschnitt werden die besten Kombinationen für Punktmultiplikationen der Form $k \cdot P$ berechnet, die keine vorberechneten Punkte verwenden. Dabei wird auf die Laufzeit der Körperoperationen aus Kapitel 8 und die Anzahl der einzelnen Körperoperationen je Punktoperation aus Kapitel 6, Tabelle 6.5 und 6.6, benötigt.

Die Methode nach Lim und Lee, ALG 6.1.6, wird in diesem Abschnitt nicht aufgenommen, da sie entweder ein Minimum an $n/2$ Punktoperationen in der Vorberechnung benötigt, was in der Hauptrechnung wirklich zu viel ist, oder diese Methode dieselbe ist wie die Square-and-Multiply.

Die Vorberechnung der Exponentiation mit ± 1-Ketten besteht sogar aus $n-1$ Punktverdopplungen und wird daher auch nicht in diesem Abschnitt aufgeführt.

9.1.1 Square-and-Multiply ohne Verwendung vorberechneter Punkte

Die Square-and-Multiply benötigt keine Vorberechnung und im Schnitt $n/2$ Punktadditionen und immer $n-1$ Punktverdopplungen in der Hauptrechnung:

INPUT: k, P
OUTPUT: $k \cdot P$

$R \leftarrow \mathcal{O}$
for $i = n-1$ down to 0
  $R \leftarrow 2 \cdot R$
  if $k_i = 1$ then
    $R \leftarrow R + P$
return R

Jede Addition gebe das Ergebnis in Koordinaten der Art B zurück. Die jeweils erste Verdopplung nach einer Addition sowie hintereinander ausgeführte Punktverdopplungen sollen einen Punkt in Koordinaten der Art C zurück geben, die Ver-

dopplung vor einer Addition in Koordinaten der Art A. Am Ende wird das Resultat in Affine Koordinaten überführt.

Im Schnitt sind bei einer zufälligen n-Bit Zahl $n/2$ Bit gesetzt. Ebenfalls sind im Schnitt jeweils $(n-1)/4$ mal die Ketten 11_2, 10_2, 00_2 und 01_2 zu erwarten. Die Wahrscheinlichkeit, dass das letzte Bit im Exponenten k_0 gesetzt ist, ist $1/2$. Dann liegt das Resultat in Koordianten B vor, sonst in Koordinaten der Art C. Man kann daher die Komplexität der Square-and-Multiply wie folgt abschätzen:

Hauptrechnung:

1	$\times$	$R^B \leftarrow P^A + \mathcal{O}$	$+$
$(n/2-1)$	$\times$	$R^B \leftarrow P^A + R^A$	$+$
$(n-1)/4$	$\times$	$R^A \leftarrow 2 \cdot R^B$	$+$
$(n-1)/4$	$\times$	$R^C \leftarrow 2 \cdot R^B$	$+$
$(n-1)/4$	$\times$	$R^C \leftarrow 2 \cdot R^C$	$+$
$(n-1)/4$	$\times$	$R^A \leftarrow 2 \cdot R^C$	$+$
$1/2$	$\times$	$R^A \leftarrow R^B$	$+$
$1/2$	$\times$	$R^A \leftarrow R^C$.	

Das Programm zur Berechnung der günstigsten Koordinatenkombinationen berechnet die beste Wahl für A, B und C. Die ersten acht besten Kombinationen werden in der Tabelle 9.1 zusammengefasst.

Bei dieser Methode ist die Koordinatenwahl wohl unabhängig von der Bitlänge. Die einfachste Lösung, das Verwenden von Affinen oder zumindest Projektiven Koordinaten ist eindeutig nicht die beste. Schon bei Bitlänge 140 beträgt der zeitliche Unterschied zwischen der besten Ausprägung und der mit ausschließlich Affinen Koordinaten knappe 4 ms, bei 272 Bit sogar schon fast 25 ms. Die Projektiven Koordinaten schneiden etwas besser ab. Der Unterschied zwischen der besten Kombination und der rein Modifiziert Jacobischen beträgt bei der kleinsten Bitlänge etwa 0.5 ms, bei der größten fast 1.9 ms. 1 bis zwei Sekunden können bei Echtzeitanwendungen durchaus schon einen Unterschied machen.

9.1.2 Fixed-size-sliding-window Exponentiation ohne Verwendung vorberechneter Punkte

Die Vorberechnung des Algorithmus 6.1.2 auf Seite 69 kann man auf zwei verschiedene Arten berechnen:

1. *Vorberechnung:*

Square-and-Multiply ovP, alle Bitlängen							
Koordinaten			hochgerechnete Zeit in ms				
A	B	C	140	160	192	197	272
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	16.08	17.71	24.39	30.64	57.7
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	16.49	18.09	24.96	31.51	59.32
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	17.46	19.11	26.39	33.43	62.92
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	19.06	20.93	28.82	36.48	68.68
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	20.28	25.67	38.05	41.62	86.02
$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	**15.55**	**17.13**	**23.6**	**29.63**	**55.8**
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	15.56	17.14	23.6	29.64	55.81
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	15.56	17.14	23.61	29.64	55.82
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	15.57	17.15	23.61	29.65	55.83
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	16.04	17.65	24.31	30.59	57.6
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	16.05	17.66	24.32	30.6	57.62
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	16.08	17.71	24.39	30.64	57.7
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	16.09	17.72	24.4	30.65	57.72

Tabelle 9.1: Hochgerechnete Laufzeit Square-and-Multiply (ALG 6.1.1) ohne Verwendung vorberechneter Punkte, alle Bitlängen

INPUT: w, P
OUTPUT: $P_i = (i+1) \cdot P$ mit $0 \leq i < 2^w - 1$

$P_0 \leftarrow P$
`for` $i = 1$ `to` $2^{w-1} - 1$
  $j = 2 \cdot i$
a $P_{j-1} \leftarrow 2 \cdot P_{i-1}$
b $P_j \leftarrow P_{j-1} + P_0$

2. *Vorberechnung:*
INPUT: w, P
OUTPUT: $P_i = (i+1) \cdot P$ mit $0 \leq i < 2^w - 1$

$P_0 \leftarrow P$
`for` $i = 1$ `to` $2^{w-1} - 1$
a $P_i \leftarrow P_{i-1} + P_0$

Die erste Art benötigt dafür jeweils $2^{w-1} - 1$ Punktadditionen und $2^{w-1} - 1$ Verdopplungen. Die zweite Art dagegen benötigt 1 Punktverdopplung ($P_1 \leftarrow P_0 + P_0$) und $2^w - 3$ Punktadditionen.

Eine Verdopplung in der additiven Punktegruppe entspricht einer Multiplikation in einer multiplikativen Gruppe, eine Punktverdopplung einer Quadrierung. In vielen multiplikativen Gruppen ist eine Quadrierung schneller zu berechnen als die Multiplikation. In der Punktegruppe wird auch allgemein gesagt, eine Verdopplung sei schneller zu berechnen als eine Addition. So benötigt eine Punktverdopplung im Jacobischen System 4 Multiplikationen, 2 Quadrierungungen und 13 Additionen in $\mathbb{F}_p$, wogegen eine Punktaddition 12 Multiplikationen, 4 Quadrierungen und 7 Additionen benötigt. Trotzdem kann die Vorberechnung nach der zweiten Art Vorteile bringen:

In jedem Schleifendurchgang wird P als Affiner Punkt zum Ergebnis addiert. Das Addieren von Affinen Punkten ist in allen projektiven Systemen günstiger als das von projektiven Koordinaten (siehe Tabelle 6.5 auf Seite 96). Wir lassen für Verdopplungen und Additionen unterschiedliche Koordinatensysteme zu.

Vorberechnung nach Art 1:

Die Addition in Schritt a gebe das Ergebnis im Koordinatensystem A zurück, die Punktverdopplung in Schritt b im Koordinatensystem B. Das hat zur Folge, dass die Hälfte alle Schritte 1 der Form $P^A_{j-1} \leftarrow 2 \cdot P^A_{i-1}$ ist und die andere Hälfte von der Form $P^A_{j-1} \leftarrow 2 \cdot P^B_{i-1}$. Schritt 2 wiederum ist immer vom Typ $P^B_j \leftarrow P^A_{i-1} + P^{\mathcal{A}}_i$. Es gibt eine Ausnahme: Im ersten Schleifendurchgang, $i = 1$ und $j = 2$, gilt: $P^A_1 \leftarrow 2 \cdot P^{\mathcal{A}}_0$. Es ergibt sich folgende Komplexitätsberechnung:

Vorberechnung:

1	$\times$	$P^A_1 \leftarrow 2 \cdot P^{\mathcal{A}}_0$	$+$	
$(2^{w-2}-1)$	$\times$	$P^A_{j-1} \leftarrow 2 \cdot P^A_{i-1}$	$+$	
$(2^{w-2}-1)$	$\times$	$P^A_{j-1} \leftarrow 2 \cdot P^B_{i-1}$	$+$	
$(2^{w-1}-1)$	$\times$	$P^B_j \leftarrow P^A_{\cdot} + P^{\mathcal{A}}_{\cdot}$.		

Die Hälfte der vorberechneten Punkte P_i, $1 \leq i < 2^w - 2$, liegt also in Koordinatensystem A vor, die andere Hälfte im Koordinatensystem B.

Vorberechnung nach Art 2:

Bei dieser Vorberechnungsart gibt es eine Punktverdopplung im ersten Schleifendurchlauf, $P^A_1 \leftarrow P^{\mathcal{A}}_0 + P^{\mathcal{A}}_0 = 2 \cdot P^{\mathcal{A}}_0$. Die nächste Addition ist dann vom Typ $P^B_2 \leftarrow P^A_1 + P^{\mathcal{A}}_0$. Es gibt dann noch $2^w - 4$ weitere Additionen vom Typ $P^B_2 \leftarrow P^B_1 + P^{\mathcal{A}}_0$:

Vorberechnung:

1	$\times$	$P_1^A \leftarrow 2 \cdot P_0^{\mathcal{A}}$		$+$
1	$\times$	$P_2^B \leftarrow P_1^A + P_0^{\mathcal{A}}$		
$(2^w - 4)$	$\times$	$P_-^B \leftarrow P_-^B + P_0^{\mathcal{A}}$.		

Hauptrechnung

Die Hauptrechnung benötigt im Schnitt $n(2^w - 1)/(w \cdot 2^w)$ Additionen und $\lfloor \frac{n-1}{w} \rfloor \cdot w$ Verdopplungen:

Hauptrechnung:
INPUT: k, P, $P_i = i \cdot P$ mit $0 \leq i < 2^w$
OUTPUT: $k \cdot P$

$R \leftarrow \mathcal{O}, i \leftarrow n - 1$
$t = n \% w$
`if` $(t \neq 0)$
  `if` $(k_{n-1}k_{n-2} \ldots k_{t+1})_2 \neq 0$ `then`
1 $R \leftarrow R + P_{(k_{n-1}k_{n-2} \ldots k_{t+1})_2}$,
    $i \leftarrow i - t$
`while` $i \geq 0$
  `for` $j = 0$ `to` $j \leq w - 1$
2 $R \leftarrow 2 \cdot R$
  `if` $(k_i k_{i-1} \ldots k_{i-w+2}k_{i-w+1})_2 \neq 0$ `then`
3 $R \leftarrow R + P_{(k_i k_{i-1} \ldots k_{i-w+2}k_{i-w+1})_2}$
    $i \leftarrow i - w$
`return` $R^{\mathcal{A}}$

Wir lassen für die Punktaddition verschiedene Koordinatensysteme zu: $R^E \leftarrow R^D + P_{\cdot}^A$ oder $R^E \leftarrow R^D + P_{\cdot}^B$ in Schritt 1 und 3 (siehe Vorberechnung). Ob Schritt 1 ausgeführt wird, hängt davon ab, ob n durch w teilbar ist oder nicht. Je nach Art der Vorberechnung setzen sich die Punktadditionen zusammen:

1. Art: Von den $n(2^w - 1)/(w \cdot 2^w)$ Additionen ist die erste, $R^E \leftarrow R^D + P_-^{A/B}$, eine Zuweisung, da R bei der ersten Addition noch der Nullpunkt ist: $R^E \leftarrow \mathcal{O} + P_-^{A/B} = R^E \leftarrow P_-^{A/B}$. Von den $2^w - 1$ vorberechneten Punkten liegt P_0 in Affiner Darstellung, $2^{w-1} - 1$ Punkte in Darstellung A und $2^{w-1} - 1$ Punkte in Darstellung B vor. Somit ist die Wahrscheinlichkeit, dass keine Addition erfolgt $P((k_- \ldots k_-)_2 = 0) = 1/2^w$. Die Wahrscheinlichkeit,

dass P als Affiner Punkt aufaddiert wird, ist ebenso groß, beziehungsweise klein: $P((k_- \dots k_-)_2 = 1) = 1/2^w$.

Die Wahrscheinlichkeit, dass ein Punkt vom Typ A aufaddiert wird ist $P((k_- \dots k_-)_2 = \text{even}) = (2^{w-1}-1)/2^w$, was die gleiche Wahrscheinlichkeit dafür ist, dass ein Punkt vom Typ B aufaddiert wird:$P((k_- \dots k_-)_2 = \text{odd}) = (2^{w-1}-1)/2^w$.

2. Art: Von den $n(2^w-1)/(w \cdot 2^w)$ Additionen ist die erste, $R^E \leftarrow R^D + P_-^{A/B}$, auch hier eine Zuweisung, da R bei der ersten Addition noch der Nullpunkt ist: $R^E \leftarrow \mathcal{O} + P_-^{A/B} = R^E \leftarrow P_-^{A/B}$. In diesem Fall gibt es aber nur einen Punkt, der vom Typ A ist, nämlich der Punkt P_1. P_0 liegt affin vor, alle verbleibenden Punkte liegen im Koordinatensystem B vor. Daher ergeben sich folgende Wahrscheinlichkeiten: $P((k_- \dots k_-)_2 = 0) = 1/2^w$, $P((k_- \dots k_-)_2 = 1) = 1/2^w$, $P((k_- \dots k_-)_2 = 2) = 1/2^w$ und $P((k_- \dots k_-)_2 > 2) = (2^w-3)/2^w$.

Aufeinanderfolgende Punktverdopplungen werden in ein und demselben Koordinatensystem berechnet: $R^C \leftarrow 2 \cdot R^C$. Vor jeder echten Punktaddition wird eine Punktverdopplung der Form $R^D \leftarrow 2 \cdot R^C$, und nach jeder Punktaddition wird eine Punktverdopplung der Form $R^C \leftarrow 2 \cdot R^E$ ausgeführt. In $1/2^w$ Fällen ist die letzte Operation eine Punktverdopplung, sonst eine Punktaddition. Eine genauere Aussage ist nicht möglich. Ist die letzte Operation eine Verdopplung, hat die Normierung des Ergebnispunktes die Form $R^{\mathcal{A}} \leftarrow R^D$, sonst $R^{\mathcal{A}} \leftarrow R^E$. Es ergeben sich für die zwei unterschiedlichen Vorberechnungsarten folgende Berechnungen:

1. Art: Vorberechnung:

1	$\times$	$P_1^A \leftarrow 2 \cdot P_0^{\mathcal{A}}$	$+$
$(2^{w-2}-1)$	$\times$	$P_{j-1}^A \leftarrow 2 \cdot P_{i-1}^A$	$+$
$(2^{w-2}-1)$	$\times$	$P_{j-1}^A \leftarrow 2 \cdot P_{i-1}^B$	$+$
$(2^{w-1}-1)$	$\times$	$P_j^B \leftarrow P_\cdot^{\mathcal{A}} + P_\cdot^{\mathcal{A}}$.	

Hauptrechnung:

$1/(2^w-1)$	$\times$	$R^E \leftarrow P_{(\dots)_2}^{\mathcal{A}}$	$+$
$(2^{w-1}-1)/(2^w-1)$	$\times$	$R^E \leftarrow P_{(\dots)_2}^{\mathcal{A}}$	$+$
$(2^{w-1}-1)/(2^w-1)$	$\times$	$R^E \leftarrow P_{(\dots)_2}^{B}$	$+$
$1/(2^w-1) \cdot (\frac{n(2^w-1)}{w \cdot 2^w} - 1)$	$\times$	$R^E \leftarrow R^D + P_-^{\mathcal{A}}$	$+$
$(2^{w-1}-1)/(2^w-1) \cdot (\frac{n(2^w-1)}{w \cdot 2^w} - 1)$	$\times$	$R^E \leftarrow R^D + P_-^{A}$	$+$
$(2^{w-1}-1)/(2^w-1) \cdot (\frac{n(2^w-1)}{w \cdot 2^w} - 1)$	$\times$	$R^E \leftarrow R^D + P_-^{B}$	$+$

$$
\begin{array}{lcll}
(\lfloor \frac{n-1}{w} \rfloor \cdot w - 2 \cdot (\frac{n(2^w-1)}{w \cdot 2^w} - 1)) & \times & R^C \leftarrow 2 \cdot R^C & + \\
(n(2^w-1)/(w \cdot 2^w) - 1) & \times & R^D \leftarrow 2 \cdot R^C & + \\
(n(2^w-1)/(w \cdot 2^w) - 1) & \times & R^C \leftarrow 2 \cdot R^E & + \\
(2^w-1)/2^w & \times & R^{\mathcal{A}} \leftarrow R^E & + \\
1/2^w & \times & R^{\mathcal{A}} \leftarrow R^C. &
\end{array}
$$

2. Art: Vorberechnung:

$$
\begin{array}{lcll}
1 & \times & P_1^{\mathcal{A}} \leftarrow 2 \cdot P_0^{\mathcal{A}} & + \\
(2^w-3) & \times & P_{j-1}^B \leftarrow P_{\cdot}^{\mathcal{A}} + P_{\cdot}^{\mathcal{A}}. &
\end{array}
$$

Hauptrechnung:

$$
\begin{array}{lcll}
1/(2^w-1) & \times & R^E \leftarrow P_{(\ldots)_2}^{\mathcal{A}} & + \\
1/(2^w-1) & \times & R^E \leftarrow P_{(\ldots)_2}^{\mathcal{A}} & + \\
(2^w-3)/(2^w-1) & \times & R^E \leftarrow P_{(\ldots)_2}^{B} & + \\
1/(2^w-1) \cdot (\frac{n(2^w-1)}{w \cdot 2^w} - 1) & \times & R^E \leftarrow R^D + P_{-}^{\mathcal{A}} & + \\
(1/(2^w-1) \cdot (\frac{n(2^w-1)}{w \cdot 2^w} - 1) & \times & R^E \leftarrow R^D + P_{-}^{A} & + \\
(2^w-3)/(2^w-1) \cdot (\frac{n(2^w-1)}{w \cdot 2^w} - 1) & \times & R^E \leftarrow R^D + P_{-}^{B} & + \\
(\lfloor \frac{n-1}{w} \rfloor \cdot w - 2 \cdot (\frac{n(2^w-1)}{w \cdot 2^w} - 1)) & \times & R^C \leftarrow 2 \cdot R^C & + \\
(n(2^w-1)/(w \cdot 2^w) - 1) & \times & R^D \leftarrow 2 \cdot R^C & + \\
(n(2^w-1)/(w \cdot 2^w) - 1) & \times & R^C \leftarrow 2 \cdot R^E & + \\
(2^w-1)/2^w & \times & R^{\mathcal{A}} \leftarrow R^E & + \\
1/2^w & \times & R^{\mathcal{A}} \leftarrow R^C. &
\end{array}
$$

Es werden die Platzhalter der Formel wiederverwendet und Auszüge der Ergebnisse für die fünf Bitgrößen 140, 160, 192, 197 und 272 in den Tabelle 9.2 und 9.3 dargestellt.

Im ersten Teil der Tabelle 9.2 ist deutlich zu erkennen, dass die Modifiziert Jacobischen Koordinaten mit 14.28 ms das beste Ergebnis für 140 Bit liefern, wenn keine gemischten Koordinaten verwendet werden. Das bestätigt sich auch bei den anderen Bitlängen. Folgende Beobachtungen sind zu machen:

Mit reinen Koordinaten sind bei jeder untersuchten Bitlänge die Modifiziert Jacobischen Koordinaten die beste Wahl, gefolgt von den Jacobischen und dann den Chudnowsky Jacobischen Koordinaten. Die Projektiven Koordinaten kommen an Stelle vier, während die Affinen Koordinaten den letzten Platz einnehmen. Die beste Fenstergröße in allen diesen Fällen ist $w = 4$.

Bei den gemischten Koordinaten gibt es jeweils vier Lösungen, die für alle untersuchten Bitlängen das beste Resultat liefern:

Fixed-size-sliding-window ovP, 140 Bit							
Koordinaten					Art	w	hochg. in ms
A	B	C	D	E			
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	1	4	14.29
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	1	4	15.36
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	2	4	15.93
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2	4	17.35
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	2	4	17.85
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	4	**13.15**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	4	13.15
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	4	13.15
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	4	13.15
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	1	4	13.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	1	4	13.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	1	4	13.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	1	4	13.2

Tabelle 9.2: Hochgerechnete Laufzeit Fixed-size-sliding-window (ALG 6.1.2) ohne Verwendung vorberechneter Punkte, 140 Bit, in ms

Fixed-size-sliding-window ovP, 160 - 272 Bit										
Koordinaten					Art	w	hochgerechnet in ms			
A	B	C	D	E			160	192	197	272
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	1	4	15.65	21.33	26.92	49.56
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	1	4	16.74	22.93	29.22	53.85
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	2	4	17.33	23.75	30.37	55.93
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2	4	18.93	25.87	33.05	60.87
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	2	4	22.43	32.96	36.48	73.62
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	4	**14.63**	**19.94**	**24.71**	**45.49**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	4	14.63	19.94	24.71	45.49
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	4	14.63	19.94	24.71	45.49
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	4	14.63	19.94	24.71	45.49
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	1	4	14.68	20.0	24.78	45.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	1	4	14.68	20.0	24.78	45.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	1	4	14.68	20.0	24.78	45.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	1	4	14.68	20.0	24.78	45.58

Tabelle 9.3: Hochgerechnete Laufzeit Fixed-size-sliding-window (ALG 6.1.2) ohne Verwendung vorberechneter Punkte, 160 - 272 Bit, in ms

Die Vorberechnung wird mit Affinen Punkten berechnet. Aufeinanderfolgende Punktverdopplungen werden in der Hauptrechnung mit Modifiziert Jacobischen Koordinaten ausgeführt. Im Übergang zur Punktaddition wird mit Jacobischen oder Chudnowsky Jacobischen Koordinaten gerechnet: Modifizierte Koordinaten sind in der Punktaddition teuer, es können aber Körperoperationen eingespart werden, legt man das Ergebnis in Jacobischen Koordinaten ab. Der Mehraufwand, der entsteht, um Chudnowsky Jacobische Koordinaten zu erzeugen, wird in der Punktaddition wieder eingespart.

Kürzere Laufzeiten erreicht man auf jeden Fall mit gemischten Koordinaten, nicht mit reinen. Die Differenz zwischen der Laufzeit im besten reinen Fall gegenüber der im besten gemischten, wächst mit der Bitlänge: Bei 140 Bit spart man mit den gemischten Koordinaten gegenüber den Modifiziert Jacobischen 1.12 ms ein, bei 160 Bit 1.02 ms, bei 192 Bit spart man 1.08 ms, bei 197 Bit schon 2.21 ms und bei 272 Bit ganze 4.07 ms. Bei hohen Effizienzanforderungen bedeuten 3 - 5 ms mehr oder weniger einen großen Unterschied.

Die Unterschiede der Laufzeiten zwischen den aufeinanderfolgenden Lösungen mit gemischten Koordinaten sind sehr klein. Es handelt sich zum größten Teil um hundertstel Millisekunden.

Echte Unterschiede innerhalb der ersten 10 Kombinationen können daher sicher erst nach sehr vielen Durchläufen festgestellt werden. Bei kleinen Bitlängen ist vielleicht selbst dann nichts zu erkennen.

Laufzeitverbesserungen durch unterschiedliche Wahl von Koordinaten und Fenstergröße sind abhängig von der Rechenleistung: Je mehr Zeit eine Rechner für eine Körpermultiplikation und -invertierung benötigt, desto größer wird der Unterschied zwischen den einzelnen Ausprägungen sein. Je langesamer der Rechner ist, desto mehr sollte darauf geachtet werden, dass die kostengünstigste Variation verwendet wird.

Diese Beobachtungen lassen sich zum größten Teil auf die folgenden Algorithmen übertragen.

9.1.3 Sliding-window Exponentiation ohne Verwendung vorberechneter Punkte

Die Formel zur Berechnung der zu erwartenden Laufzeit des Algorithmus 6.1.3 lässt sich ganz ähnlich herleiten wie im vorherigen Abschnitt:

Die Vorberechnung besteht in $2^{w-1}-1$ Punktadditionen und einer Punktverdopplung:

Vorberechnung:
INPUT: w, P
OUTPUT: $P_i = (2\cdot i+1)\cdot P$ mit $0 \le i < 2^{w-1}$

$P_0 \leftarrow P, t \leftarrow 2\cdot P$
`for` $i = 1$ `to` $2^{w-1}-1$
  $P_i \leftarrow P_{i-1} + t$

Dabei wird der Punkt t nur zur Vorberechnung benötigt. Er kann mit anderen Koordinaten abgelegt werden, als die berechneten P_i. In der ersten Addition ist der erste Summand P_1 affin. Die erste Addition ist daher vom Typ $P_2^B \leftarrow P_1^A + t^A$, die restlichen $2^{w-1}-2$ Additionen sind vom Typ $P_-^B \leftarrow P_-^B + t^A$:

Vorberechnung:

1	$\times$	$t^A \leftarrow 2\cdot P^A$	$+$
1	$\times$	$P_1^B \leftarrow P_0^A + t^A$	
$2^{w-1}-2$	$\times$	$P_-^B \leftarrow P_-^B + t^A$.	

Die Hauptrechnung benötigt im Schnitt $n/(w+1)$ Additionen und zwischen $n-w$ und $n-1$ Verdopplungen:

Hauptrechnung:
INPUT: k, P, P_i
OUTPUT: $k\cdot P$

$R \leftarrow \mathcal{O}, i \leftarrow n-1$
`while` $i \ge 0$
  `if` $k_i = 0$ `then`
a $R \leftarrow 2\cdot R, i \leftarrow i-1$
  `else`
    $m \leftarrow i-w$, `if` $(m < -1)$ `then` $m \leftarrow -1$
    $j \leftarrow m+1$
    `while` $(k_j = 0)$
      $j{+}{+}$
    $s \leftarrow (k_i k_{i-1} \ldots k_j)_2$
    `while` $(i \ge j)$
b $R \leftarrow 2\cdot R, i \leftarrow i-1$

```
c       R ← R + P_{(s-1)/2}
        while (i > m)
d           R ← 2 · R, i ← i − 1
  return R
```

Die erste der Additionen in Schritt c ist nur eine Zuweisung. Es gebe jede Addition einen Punkt im Koordinatensystem E zurück, aufeinandefolgende Verdopplungen werden im Koordinatensystem C berechnet. Für jede echte Addition gibt eine anführende Verdopplung, die den Summanden der Art C nimmt und einen Punkt vom Typ D zurückgibt. Und nach jeder Addition existiert eine Punktverdopplung, die den Summanden des Typs E nimmt und einen Punkt vom Typ B zurück gibt. Für die Addtion ist noch wichtig zu wissen, dass die Wahrscheinlichkeit, dass der Basispunkt P aufaddiert wird, $P((k_i k_{i-1} \dots k_j)_2 = 2^p, p = 0, \dots, w-1) = w/(2^w - 1)$ ist. Und P ist affin!

Die letzte Operation ist in dieser Methode mit der Wahrscheinlichkeit $1/2$ eine Addition, bzw. eine Verdopplung. Auch hier bildet eine Normierung des Ergebnispunktes den Abschluss.

Wieviele Verdopplungen tatsächlich benötigt werden, hängt vom Exponenten k ab: Es wird innerhalb des Fensters mit Größe w von hinten her das letzte Bit gesucht, das gesetzt ist. Beispiel 10 illustriert dies:

Beispiel 10: $n = 15$**,** $w = 4$

$1000..._2 \Rightarrow n-1$
$1100..._2 \Rightarrow n-2$
$1010..._2 \Rightarrow n-3$
$1110..._2 \Rightarrow n-3$
$1001..._2 \Rightarrow n-4$
$1011..._2 \Rightarrow n-4$
$1101..._2 \Rightarrow n-4$
$1111..._2 \Rightarrow n-4$

Die einzelnen Fälle *Anzahl Verdopplungen* (#DBL) treten mit unterschiedlichen Wahrscheinlichkeiten auf: $P(\#\text{DBL} = n-1) = 1/2^{w-1}$, $P(\#\text{DBL} = n-2) = 1/2^{w-1}$, $P(\#\text{DBL} = n-3) = 2^{3-2}/2^{w-1}$, $\dots$, $P(\#\text{DBL} = n-w) = 2^{w-2}/2^{w-1}$, für $w \geq 2$. Daher gewichten wir die Verdopplungen wie folgt:

$$\frac{1}{2^{w-1}} \cdot ((n-1) + (n-2) + \sum_{W=3}^{w} 2^{W-2} \cdot (n-W))$$
$$\Leftrightarrow \quad \frac{1}{2^{w-1}} \cdot ((2n-3) + \sum_{W=3}^{w} 2^{W-2} \cdot (n-W)) =: t$$

So ergibt sich folgende Berechnung:

Vorberechnung:

1	$\times$	$t^A \leftarrow 2 \cdot P^{\mathcal{A}}$	$+$
1	$\times$	$P_1^B \leftarrow P_0^{\mathcal{A}} + t^A$	
$2^{w-1} - 2$	$\times$	$P_-^B \leftarrow P_-^B + t^A.$	

Hauptrechnung:

$\frac{w}{2^w-1}$	$\times$	$R^E \leftarrow P_-^{\mathcal{A}}$	$+$
$\frac{2^w-1-w}{2^w-1}$	$\times$	$R^E \leftarrow P_-^B$	$+$
$\frac{w}{2^w-1} \cdot (\frac{n}{w+1} - 1)$	$\times$	$R^E \leftarrow R^D + P_-^{\mathcal{A}}$	$+$
$\frac{2^w-1-w}{2^w-1} \cdot (\frac{n}{w+1} - 1)$	$\times$	$R^E \leftarrow R^D + P_-^B$	$+$
$(t - 2 \cdot (\frac{n}{w+1} - 1))$	$\times$	$R^C \leftarrow 2 \cdot R^C$	$+$
$(\frac{n}{w+1} - 1)$	$\times$	$R^D \leftarrow 2 \cdot R^C$	$+$
$(\frac{n}{w+1} - 1)$	$\times$	$R^C \leftarrow 2 \cdot R^E$	$+$
$1/2$	$\times$	$R^{\mathcal{A}} \leftarrow R^E$	$+$
$1/2$	$\times$	$R^{\mathcal{A}} \leftarrow R^C.$	

Es werden die Platzhalter der Formel wiederverwendet. Tabellen 9.4 bis 9.8 fassen Auszüge der Ergebnisse zusammen.

9.1.4 Exponentiation mit Non-adjacent-forms ohne Verwendung vorberechneter Punkte

Auch für den Algorithmus 6.1.4 ist die Vorgehensweise dieselbe. Für die Vorberechnung gilt das Gleiche wie im vorhergehenden Fall:

Vorberechnung:

Sliding-window ovP, 140 Bit						
Koordinaten					w	hochg.
A	B	C	D	E		in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	13.33
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	14.44
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	15.05
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	16.4
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	16.94
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	**12.2**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	12.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	12.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	12.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	4	12.39
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	12.39
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	12.41
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	4	12.41

Tabelle 9.4: Hochgerechnete Laufzeit Sliding-window (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 140 Bit, in ms

Sliding-window ovP, 160 Bit						
Koordinaten					w	hochg.
A	B	C	D	E		in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	14.63
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	15.76
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	16.42
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	17.09
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	21.34
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	**13.58**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	13.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	13.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	13.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	4	13.78
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	13.78
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	13.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	4	13.81

Tabelle 9.5: Hochgerechnete Laufzeit Sliding-window (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 160 Bit, in ms

Sliding-window ovP, 192 Bit						
Koordinaten					w	hochg. in ms
A	B	C	D	E		
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	19.95
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	21.62
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	22.54
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	24.59
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	31.48
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	**18.57**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	18.57
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	18.57
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	18.57
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	5	18.84
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	5	18.84
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	18.85
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	4	18.85

Tabelle 9.6: Hochgerechnete Laufzeit Sliding-window (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 192 Bit, in ms

Sliding-window ovP, 197 Bit						
Koordinaten					w	hochg. in ms
A	B	C	D	E		
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	24.88
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	27.23
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	28.47
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	31.01
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	34.45
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	**22.84**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	22.84
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	22.84
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	22.84
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	22.98
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	5	22.98
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	5	22.98
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	5	22.98

Tabelle 9.7: Hochgerechnete Laufzeit Sliding-window (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 197 Bit, in ms

Sliding-window ovP, 272 Bit						
Koordinaten					w	hochg. in ms
A	B	C	D	E		
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	46.29
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	50.81
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	5	53.14
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	5	57.87
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	5	70.29
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	**42.63**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	5	42.63
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	5	42.63
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	5	42.63
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	42.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	42.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	42.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	42.73

Tabelle 9.8: Hochgerechnete Laufzeit Sliding-window (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 272 Bit, in ms

INPUT: w, P
OUTPUT: $P_i = (2i+1) \cdot P$ mit $0 \leq i < 2^{w-1}$

$P_0 = P$
$t = 2 \cdot P$
`for` $i = 1$ `to` $2^{w-1} - 1$
  $P_i \leftarrow P_{i-1} + t$

Daher gilt auch hier:

Vorberechnung:

$$\begin{array}{lll} 1 & \times & t^A \leftarrow 2 \cdot P^{\mathcal{A}} \quad + \\ 1 & \times & P_1^B \leftarrow P_0^{\mathcal{A}} + t^A \\ 2^{w-1}-2 & \times & P_-^B \leftarrow P_-^B + t^A. \end{array}$$

Die Hauptrechnung dieses Algorithmus unterscheidet sich ein wenig vom Algorithmus 6.1.3. Sie benötigt im Schnitt $n/(w+2)$ Punktadditionen und ebenso wie bei der vorangehenden $n - w$ bis $n - 1$ Verdopplungen:

Hauptrechnung:

INPUT: k, P, P_i
OUTPUT: $k \cdot$

```
R ← O
for i = N.length; i ≥ 0; i ← i − 1
   R ← 2 · R
   if N[i] ≠ 0 then
      R ← R + P_{N[i]}
return R
```

Auch die Komplexität unterscheidet sich - ebenfalls nur wenig - von der des vorhergehenden Algorithmus: Es gibt im Schnitt etwas weniger Punktadditionen, nämlich nur $n/(w+2)$, und die Wahrscheinlichkeit, dass die letzte Operation eine Addition ist, ist hier $1/(w+2)$, da im Schnitt nur jede $w+2$-te Stelle der Non-adjacent-Form gesetzt ist. Daher ist die Wahrscheinlichkeit, dass die letzte Operation eine Verdopplung ist, $(w+1)/(w+2)$. Auch die Wahrscheinlichkeit, dass P als Affiner Punkt addiert oder subtrahiert wird, ist anders: Wenn $N[i] \neq 0$, dann ist die Wahrscheinlichkeit den Basispunkt zu addieren oder zu subtrahieren $P(N[i] = \pm 1) = 2/2^w = 1/2^{w-1}$.

Da wir davon ausgehen, dass die höchstwertige Stelle der non-adjacent-form des Exponenten ungleich Null ist, und höchstens eine von $w+1$ aufeinander folgenden Stellen ungleich null ist, folgt, dass die Anzahl der Verdopplungen höchstens $n-w-1$ ist. Es ergibt sich folgende Komplexität:

Vorberechnung:

1	$\times$	$t^A \leftarrow 2 \cdot P^{\mathcal{A}}$	$+$
1	$\times$	$P_1^B \leftarrow P_0^{\mathcal{A}} + t^A$	
$2^{w-1}-2$	$\times$	$P_-^B \leftarrow P_-^B + t^A$.	

Hauptrechnung:

$\frac{1}{2^{w-1}}$	$\times$	$R^E \leftarrow P_-^{\mathcal{A}}$	$+$
$\frac{2^{w-1}-1}{2^{w-1}}$	$\times$	$R^E \leftarrow P_-^B$	$+$
$\frac{1}{2^{w-1}} \cdot (\frac{n}{w+2} - 1)$	$\times$	$R^E \leftarrow R^D + P_-^{\mathcal{A}}$	$+$
$\frac{2^{w-1}-1}{2^{w-1}} \cdot (\frac{n}{w+2} - 1)$	$\times$	$R^E \leftarrow R^D + P_-^B$	$+$
$(n-w-1-2 \cdot (\frac{n}{w+2} - 1))$	$\times$	$R^C \leftarrow 2 \cdot R^C$	$+$
$(\frac{n}{w+2} - 1)$	$\times$	$R^D \leftarrow 2 \cdot R^C$	$+$
$(\frac{n}{w+2} - 1)$	$\times$	$R^C \leftarrow 2 \cdot R^E$	$+$

NAF ovP, 140, 160 und 192 Bit								
Koordinaten					w	hochg. in ms		
A	B	C	D	E		140	160	192
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	12.73	13.99	19.09
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	13.91	15.2	20.88
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	14.5	15.85	21.8
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	15.81	17.34	23.78
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	16.33	20.59	30.41
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	**11.64**	**12.99**	**17.78**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	11.64	12.99	17.78
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	11.64	12.99	17.78
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	11.64	12.99	17.78
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	3	11.78	13.12	18.0
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	3	11.78	13.12	18.0
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	3	11.78	13.12	18.0
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	3	11.78	13.12	18.0

Tabelle 9.9: Hochgerechnete Laufzeit NAF (ALG 6.1.4) ohne Verwendung vorberechneter Punkte, 140, 160 und 192 Bit, in ms

$$\begin{array}{lll} 1/(w+2) & \times & R^{\mathcal{A}} \leftarrow R^{E} \qquad + \\ (w+1)/(w+2) & \times & R^{\mathcal{A}} \leftarrow R^{C}. \end{array}$$

Tabellen 9.9 bis 9.10 fassen Auszüge der Ergebnisse zusammen.

9.1.5 Vergleich der hochgerechneten Laufzeiten der Methoden für $k \cdot P$ ohne Verwendung vorberechneter Punkte

Die jeweils besten, hochgerechneten Zeiten können nun untereinander verglichen werden, was in Abbildung 9.1 geschieht. Das Verfahren Square-and-Multiply ist mit Abstand das schlechteste. Die restlichen vier Verfahren liegen dicht nebeneinander, das beste ist die Exponentiation mit non-adjacent-forms.

NAF ovP, 197 Bit						
Koordinaten					w	hochg. in ms
A	B	C	D	E		
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	23.78
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	26.28
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	27.52
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	29.97
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	33.29
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	**21.82**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	21.82
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	21.82
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	21.82
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	4	22.13
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	22.13
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	22.15
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	4	22.15

Tabelle 9.10: Hochgerechnete Laufzeit NAF (ALG 6.1.4) ohne Verwendung vorberechneter Punkte, 197 Bit, in ms

NAF ovP, 272 Bit						
Koordinaten					w	hochg. in ms
A	B	C	D	E		
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	43.31
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	49.15
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	51.53
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	56.14
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	68.47
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	**40.9**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	40.9
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	40.9
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	40.9
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	41.25
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	5	41.25
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	5	41.25
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	5	41.25

Tabelle 9.11: Hochgerechnete Laufzeit NAF (ALG 6.1.4) ohne Verwendung vorberechneter Punkte, 272 Bit, in ms

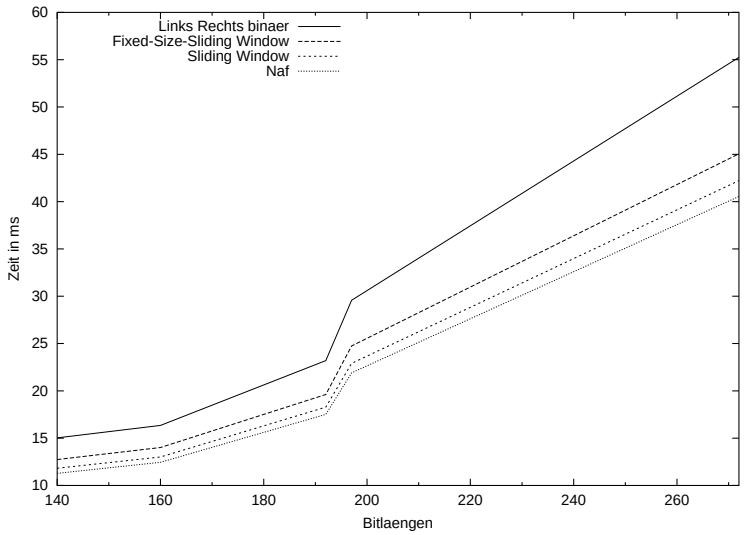


Abbildung 9.1: Vergleich der hochgerechneten Laufzeit von $k \cdot P$ ohne Verwendung vorberechneter Punkte

9.2 Einfache Punktmultiplikationsmethoden unter Verwendung vorberechneter Punkte

In diesem Abschnitt erkennt man, welchen großen Einfluss die Vorberechnung im vorherigen Abschnitt auf die Komplexität ausübt. Die gleichen Algorithmen, die vorher unterschiedliche optimale Kombinationen besaßen, haben nun die gleichen Kombinationen. Daher werden die Ergebnisauszüge für alle fünf Bitlängen gemeinsam in einer Tabelle präsentiert.

Die Formeln der Komplexitätsberechnung werden nicht erneut hergeleitet, da sie nur eine Wiederholung jener der vorherigen Abschnitte sind. Der einzige Unterschied besteht darin, dass nun davon ausgegangen wird, dass die vorberechneten Punkte affin vorliegen.

Für alle Methoden wird ein maximaler Speicherplatz von 50 KB vorgegeben, der zum Speichern der Punkte zur Verfügung stehen soll. Ebenso wie die Plattform kann diese Zahl für unterschiedliche Anwendungen neu gewählt werden. Ziel ist es, für alle Methoden eine gemeinsame Ausgangsbasis zu haben, um sie besser

vergleichen zu können. Jede Methode muss daher die Ungleichung

$$\begin{aligned} &\#\text{vorberechnete Punkte } \cdot 2 \cdot n \text{ Bits } \leq 50 \cdot 2^{13} \text{ Bits} \\ &\Leftrightarrow \\ &\#\text{vorberechnete Punkte } \leq 50/n \cdot 2^{12} \end{aligned} \tag{9.1}$$

erfüllen. Die Anzahl der vorberechneten Punkte ist abhängig von den Parametern w bei den einfachen Window- und NAF-Methoden, von w_1 und w_2 bei den Interleaving-Methoden, von h und v bei der LimLee-Methode, sowie der Bitlänge n. Die oberen Parametergrenzen, w_{max}, $w_{1\,max}$, $w_{2\,max}$, h_{max} und v_{max} werden bei jeder Methode in Abhängigkeit der Bitlänge n neu gewählt.

9.2.1 Fixed-size-sliding-window Exponentiation unter Verwendung vorberechneter Punkte

Die Formeln zur Komplexitätsberechnung sind direkt aus Abschnitt 9.1.2 ersichtlich. Dabei sind die Platzhalter A und B direkt durch $\mathcal{A}$ zu ersetzen, da die vorberechneten Punkte affin abgespeichert sind.

Hauptrechnung:

1	$\times$	$R^E \leftarrow P^{\mathcal{A}}_{(\ldots)_2}$	$+$
$(\frac{n(2^w-1)}{w \cdot 2^w} - 1)$	$\times$	$R^E \leftarrow R^D + P^{\mathcal{A}}_{-}$	$+$
$(\lfloor \frac{n-1}{w} \rfloor \cdot w - 2 \cdot (\frac{n(2^w-1)}{w \cdot 2^w} - 1))$	$\times$	$R^C \leftarrow 2 \cdot R^C$	$+$
$(n(2^w-1)/(w \cdot 2^w) - 1)$	$\times$	$R^D \leftarrow 2 \cdot R^C$	$+$
$(n(2^w-1)/(w \cdot 2^w) - 1)$	$\times$	$R^C \leftarrow 2 \cdot R^E$	$+$
$(2^w-1)/2^w$	$\times$	$R^{\mathcal{A}} \leftarrow R^E$	$+$
$1/2^w$	$\times$	$R^{\mathcal{A}} \leftarrow R^C$.	

Es werden die Platzhalter der Formel wiederverwendet. Tabellen 9.12 bis 9.14 fassen Auszüge der Ergebnisse zusammen. Da diese Methode $2^w - 1$ vorberechnete Punkte benötigt, ist für 140 bis 197 Bit $w_{max} = 10$ und für 272 Bit $w_{max} = 9$, wie man durch Einsetzen in die Formel (9.1) sieht:

$$\begin{aligned} 140: &\quad 2^w - 1 = 2^{10} - 1 = 1023 \leq 50/140 \cdot 2^{12} < 1463 \\ 160: &\quad 2^w - 1 = 2^{10} - 1 = 1023 \leq 50/160 \cdot 2^{12} = 1280 \\ 192: &\quad 2^w - 1 = 2^{10} - 1 = 1023 \leq 50/192 \cdot 2^{12} = 1066.\overline{6} \\ 197: &\quad 2^w - 1 = 2^{10} - 1 = 1023 \leq 50/197 \cdot 2^{12} < 1040 \\ 272: &\quad 2^w - 1 = 2^{9} - 1 = 511 \leq 50/272 \cdot 2^{12} < 753 \end{aligned}$$

Fixed-size-sliding-window uvP, 140, 160 und 197 Bit						
Koordinaten			w	hochg. in ms		
C	D	E		140	160	197
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	9.79	10.55	19.04
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	11.25	12.51	22.11
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	12.16	13.47	23.97
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	10	13.3	14.79	26.19
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	10	13.32	17.15	28.29
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	**9.69**	**10.9**	**18.85**
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	10	9.69	10.9	18.85
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	10	9.69	10.9	18.85
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	10	9.69	10.9	18.85
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	10	9.78	11.0	19.03
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	9.78	11.0	19.03
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	9.79	11.01	19.04
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	10	9.79	11.01	19.04

Tabelle 9.12: Hochgerechnete Laufzeit Fixed-size-sliding-window (ALG 6.1.2) unter Verwendung vorberechneter Punkte, 140, 160 und 197 Bit, in ms

Den Tabellen kann man entnehmen, dass es jeweils vier Ausprägungen mit dem besten Ergebnis in allen Bitlängen gibt: Die Verdopplungen werden mit Modifiziert Jacobischen Koordinaten durchführt und für Punktadditionen werden Jacobische oder Chudnowsky Jacobische Koordinaten verwendet. Nur sehr wenig langsamer ist die Anwendung von Modifiziert Jacobischen Koordinaten alleine. Das lässt sich damit begründen, dass es bei Windowgröße 10 nur $n/11$ Additionen gibt. Verdopplungen werden dagegen bis zu $n-1$ benötigt. Ein weiterer wichtiger Aspekt ist, dass eine Addition der Form $R^{\mathcal{J}^{\mathcal{M}}} \leftarrow R^{\mathcal{J}^{\mathcal{M}}} + P^{\mathcal{A}}$ nur um 3 Multiplikationen teurer ist als eine der Form $R^{\mathcal{J}^{\mathcal{M}}} \leftarrow R^{\mathcal{J}^{\mathcal{C}}} + P^{\mathcal{A}}$, wogegen eine Punktverdopplung der Form $R^{\mathcal{J}^{\mathcal{M}}} \leftarrow 2 \cdot R^{\mathcal{J}^{\mathcal{C}}}$ um 2 Multiplikationen teurer ist als eine der Form $R^{\mathcal{J}^{\mathcal{M}}} \leftarrow 2 \cdot R^{\mathcal{J}^{\mathcal{M}}}$. Eine Punktmultiplikation mit Fixed-size-Sliding-Window-Exponentiation ist mit nur Modifiziert Jacobischen Koordinaten nur um eine Multiplikation teurer als mit einer Jacobischen oder Chudnowsky Jacobischen Koordinaten.

Fixed-size-sliding-window uvP, 192 Bit				
Koordinaten			w	hochg.
C	D	E		in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	15.81
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	18.0
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	8	19.37
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	8	21.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	8	26.58
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	**15.66**
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	10	15.66
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	10	15.66
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	10	15.66
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	8	15.71
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	15.71
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	15.71
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	15.71

Tabelle 9.13: Hochgerechnete Laufzeit Fixed-size-sliding-window (ALG 6.1.2) unter Verwendung vorberechneter Punkte, 192 Bit, in ms

Fixed-size-sliding-window uvP, 272 Bit				
Koordinaten			w	hochg.
C	D	E		in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	9	37.34
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	42.86
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	8	46.37
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	8	50.67
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	8	60.37
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	**36.89**
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	8	36.89
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	36.89
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	36.89
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	9	36.93
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	9	36.93
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	9	36.93
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	9	36.93

Tabelle 9.14: Hochgerechnete Laufzeit Fixed-size-sliding-window (ALG 6.1.2) unter Verwendung vorberechneter Punkte, 272 Bit, in ms

9.2.2 Sliding-window Exponentiation unter Verwendung vorberechneter Punkte

Die Formeln zur Berechnung der zu erwartenden Laufzeit des Algorithmus 6.1.3 mit vorweggenommener Vorberechnung lassen sich direkt von denen aus Abschnitt 9.1.3 herleiten. Wie in der vorangegangenen Methode werden dabei wieder die Platzhalter A und B durch $\mathcal{A}$ ersetzt, und t ist wieder die Abschätzung der Anzahl der benötigten Verdopplungen mit $t := \frac{1}{2^{w-1}} \cdot ((2n-3) + \sum_{W=3}^{w} 2^{W-2} \cdot (n - W))$:

Hauptrechnung:

$$
\begin{array}{lllll}
1 & \times & R^E \leftarrow P_{-}^{\mathcal{A}} & + \\
(\frac{n}{w+1} - 1) & \times & R^E \leftarrow R^D + P_{-}^{\mathcal{A}} & + \\
(t - 2 \cdot (\frac{n}{w+1} - 1)) & \times & R^C \leftarrow 2 \cdot R^C & + \\
(\frac{n}{w+1} - 1) & \times & R^D \leftarrow 2 \cdot R^C & + \\
(\frac{n}{w+1} - 1) & \times & R^C \leftarrow 2 \cdot R^E & + \\
1/2 & \times & R^{\mathcal{A}} \leftarrow R^E & + \\
1/2 & \times & R^{\mathcal{A}} \leftarrow R^C. &
\end{array}
$$

Es werden die Platzhalter der Formel wiederverwendet. Tabellen 9.15 bis 9.16 fassen Auszüge der Ergebnisse zusammen. Da diese Methode $2^w - 1$ vorberechnete Punkte benötigt, ist für 140 bis 197 Bit $w_{max} = 11$ und für 272 Bit $w_{max} = 10$, wie man durch Einsetzen in die Formel (9.1) sieht:

$$
\begin{array}{ll}
140: & 2^{w-1} - 1 = 2^{10} - 1 = 1023 \leq 50/140 \cdot 2^{12} < 1463 \\
160: & 2^{w-1} - 1 = 2^{10} - 1 = 1023 \leq 50/160 \cdot 2^{12} = 1280 \\
192: & 2^{w-1} - 1 = 2^{10} - 1 = 1023 \leq 50/192 \cdot 2^{12} = 1066.\overline{6} \\
197: & 2^{w-1} - 1 = 2^{10} - 1 = 1023 \leq 50/197 \cdot 2^{12} < 1040 \\
272: & 2^{w-1} - 1 = 2^{9} - 1 = 511 \leq 50/272 \cdot 2^{12} < 753
\end{array}
$$

Das Ergebnis ist dem des vorangegangen Algorithmus sehr ähnlich: Es gibt auch hier Vierer-Gruppen, die jeweils dieselbe Laufzeit besitzen. Auch hier werden die Punktverdopplungen in Modifiziert Jacobischen Koordinaten ausgeführt und die Punktadditionen mit Chudnowsky- oder Jacobischen Koordinaten. Die Begründung ist ebenfalls die Gleiche. Ebenso ist auch hier die maximale Fenstergröße nicht immer die optimale.

Sliding-window uvP, 140, 160, 192 und 197 Bit							
Koordinaten			w	hochg. in ms			
C	D	E		140	160	192	197
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	9.56	10.76	14.88	18.33
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	11.07	12.3	17.13	21.43
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	11.97	13.27	18.48	23.97
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	13.1	14.57	20.24	25.42
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	13.1	16.88	25.32	27.44
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	**9.48**	**10.67**	**14.75**	**18.17**
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	11	9.48	10.67	14.75	18.17
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	11	9.48	10.67	14.75	18.17
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	11	9.48	10.67	14.75	18.17
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	11	9.55	10.75	14.87	18.32
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	9.55	10.75	14.87	18.32
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	9.56	10.76	14.88	18.33
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	11	9.56	10.76	14.88	18.33

Tabelle 9.15: Hochgerechnete Laufzeit Sliding-window (ALG 6.1.3) unter Verwendung vorberechneter Punkte, 140, 160, 192 und 197 Bit, in ms

Sliding-window uvP, 272 Bit				
Koordinaten			w	hochg. in ms
C	D	E		
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	35.49
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	41.34
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	44.83
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	10	48.99
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	10	58.32
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	**35.15**
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	10	35.15
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	10	35.15
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	10	35.15
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	10	35.47
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	35.47
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	35.49
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	10	35.49

Tabelle 9.16: Hochgerechnete Laufzeit Sliding-window (ALG 6.1.3) unter Verwendung vorberechneter Punkte, 272 Bit, in ms

9.2.3 Exponentiation mit Non-adjacent-forms unter Verwendung vorberechneter Punkte

Die Formeln für die Komplexität des Algorithmus 6.1.4, wenn die Vorberechnung vorweg genommen wurde, lässt sich direkt aus Abschnitt 9.1.4 ablesen, wobei wieder die Platzhalter A und B durch $\mathcal{A}$ ersetzt wurden:

Hauptrechnung:

$$\begin{array}{lcll} 1 & \times & R^E \leftarrow P_-^{\mathcal{A}} & + \\ (\frac{n}{w+2}-1) & \times & R^E \leftarrow R^D + P_-^{\mathcal{A}} & + \\ (n-1-2\cdot(\frac{n}{w+2}-1)) & \times & R^C \leftarrow 2\cdot R^C & + \\ (\frac{n}{w+2}-1) & \times & R^D \leftarrow 2\cdot R^C & + \\ (\frac{n}{w+2}-1) & \times & R^C \leftarrow 2\cdot R^E & + \\ 1/(w+2) & \times & R^{\mathcal{A}} \leftarrow R^E & + \\ (w+1)/(w+2) & \times & R^{\mathcal{A}} \leftarrow R^C. & \end{array}$$

Tabellen 9.17 bis 9.18 fassen Auszüge der Ergebnisse zusammen. Da diese Methode ebenso viele vorberechnete Punkte benötigt wie die Sliding Window Exponentiation, nämlich $2^w - 1$, gelten hier die gleichen maximalen Fenstergrößen: Für 140 bis 197 Bit $w_{max} = 11$ und für 272 Bit $w_{max} = 10$.

Die Ergebnisse sind zu denen der vorangegangenen Methoden nahezu identisch, nur die Laufzeit hat sich weiter verbessert.

9.2.4 Exponentiation mit ± 1-Additionsketten unter Verwendung vorberechneter Punkte

Die Hauptrechnung dieses Algorithmus unterscheidet sich fast gar nicht vom Algorithmus 6.1.4. Sie benötigt aber keine Verdopplungen, dafür $n/2$ Punktadditionen:

Hauptrechnung:
INPUT: k, P, P_i, $N(k)$
OUTPUT: $k \cdot P$

$R \leftarrow \mathcal{O}$
`for` $i = N.length; i \geq 0; i \leftarrow i-1$
  `if` $N[i] == -1$ `then`

NAF uvP, 140, 160, 192 und 197 Bit							
Koordinaten			w	hochg. in ms			
C	D	E		140	160	192	197
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	9.34	10.53	14.6	17.98
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	10.84	12.97	16.85	21.09
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	11.73	13.03	18.19	22.89
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	12.83	14.31	19.92	25.02
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	12.83	16.56	24.9	27.0
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	**9.26**	**10.45**	**14.48**	**17.83**
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	11	9.26	10.45	14.48	17.83
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	11	9.26	10.45	14.48	17.83
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	11	9.26	10.45	14.48	17.83
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	11	9.33	10.52	14.59	17.97
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	9.33	10.52	14.59	17.97
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	9.34	10.53	14.6	17.98
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	11	9.34	10.53	14.6	17.98

Tabelle 9.17: Hochgerechnete Laufzeit NAF (ALG 6.1.4) unter Verwendung vorberechneter Punkte, 140, 160, 192 und 197 Bit, in ms

NAF uvP, 272 Bit				
Koordinaten			w	hochg. in ms
C	D	E		
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	34.86
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	40.74
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	44.2
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	10	48.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	10	57.5
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	**34.55**
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	10	34.55
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	10	34.55
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	10	34.55
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	10	34.84
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	34.84
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	34.86
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	10	34.86

Tabelle 9.18: Hochgerechnete Laufzeit NAF (ALG 6.1.4) unter Verwendung vorberechneter Punkte, 272 Bit, in ms

±1-Additionsketten uvP, alle Bitlängen					
Koordinate	hochg. Zeit in ms je Bitlänge				
B	140	160	192	197	272
$\mathcal{J}^{\mathcal{C}}$	**5.69**	**6.16**	**8.52**	**11.0**	**20.65**
$\mathcal{J}$	5.69	6.16	8.52	11.0	20.65
$\mathcal{P}$	6.15	6.67	9.21	11.93	22.41
$\mathcal{J}^{\mathcal{M}}$	6.35	7.68	10.66	13.13	26.03
$\mathcal{A}$	7.13	8.1	12.06	13.86	27.27

Tabelle 9.19: Hochgerechnete Laufzeit ±1-Additionsketten (ALG 6.1.5) unter Verwendung vorberechneter Punkte, alle Bitlängen, in ms

$R \leftarrow R - P_i$
`else` bif $N[i] == 1$ `then`
$R \leftarrow R + P_i$
`return` R

Diese Methode benötigt $n - 1$ vorberechnete Punkte. Sie benötigen für jeder der fünf Bitlängen weniger als 50 KB, daher kann die Methode im gleichen Rahmen gestetet werden.

Hauptrechnung:

1	$\times$	$R^B \leftarrow P_i^{\mathcal{A}}$	$+$
$(n/2-1)$	$\times$	$R^B \leftarrow R^B \pm P_i^{\mathcal{A}}$	$+$
1	$\times$	$R^{\mathcal{A}} \leftarrow R^B$.	

Die Ergebnisse werden in der Tabelle 9.19 zusammengefasst. Bei dieser Methode schneiden die Chudnowsky Jacobischen zusammen mit den Jacobischen Koordinaten am besten ab. Das liegt daran, dass nur Additionen benötigt werden und immer einer der Summanden affin ist. Laut Tabelle 6.5 sind diese beien Additionsarten die schnellsten. Da das Modifiziert Jacobische System für Additionen eher ungeeignet ist, wird es vom Projektiven System geschlagen. Das Affine System ist weit abgeschlagen, da es im Schnitt $n/2$ Invertierungen pro Punktmultiplikation benötigt.

9.2.5 Exponentiation mit LimLee unter Verwendung vorberechneter Punkte

Die Hauptrechnung benötigt $\frac{n}{h}$ Additionen und $\frac{n}{hv} - 1$ Verdopplungen:

Hauptrechnung:

$R \leftarrow \mathcal{O}$
`for` $i = b-1$ `to` 0
  $R \leftarrow 2 \cdot R$
  `for` $j = v-1$ `to` 0
    // μ ist hier die Binärdarstellung von $k_{h-1,j,0} \ldots k_{0,j,0}$
    $R \leftarrow R + P_{K_\mu,j}$
`return` R

Bei dieser Methode werden $v-1$ Punktadditionen hintereinander ausgeführt. Es gibt zwei verschiedene Fälle: Ist $v = 1$, werden Verdopplung und Addition abwechselnd ausgeführt. In diesem Falle gebe die Verdopplung das Ergebnis in Koordinaten der Art C wieder, die Additon in Koordinaten der Art E.

Im zweiten Fall ist $v > 1$ und mindestens zwei Additionen werden hintereinander ausgeführt. Jede Verdopplung gebe das Ergebnis in den Koordinaten der Art C zurück. Die daran anschließende Addition gebe das Ergebnis in den Koordinaten D wieder und jede Addition vor einer Verdopplung in Koordinaten der Art E. Es ergibt sich folgende Komplexität:

Hauptrechnung:

$v = 1$:

1	$\times$	$R^E \leftarrow P^{\mathcal{A}}_{K_{-,-}}$	$+$
$(\frac{n}{h} - 1)$	$\times$	$R^E \leftarrow R^C + P^{\mathcal{A}}_{K_{-,-}}$	$+$

$v > 1$:

1	$\times$	$R^E \leftarrow P^{E}_{K_{-,-}}$	$+$
$(\frac{n}{h} - 2 \cdot (\frac{n}{hv} - 1))$	$\times$	$R^D \leftarrow R^D + P^{\mathcal{A}}_{K_{-,-}}$	$+$
$(\frac{n}{hv} - 1)$	$\times$	$R^D \leftarrow R^C + P^{\mathcal{A}}_{K_{-,-}}$	$+$
$(\frac{n}{hv} - 1)$	$\times$	$R^E \leftarrow R^D + P^{\mathcal{A}}_{K_{-,-}}$	$+$

$v \geq 1$:

$\frac{n}{hv} - 1$	$\times$	$R^C \leftarrow 2 \cdot R^E$.	

Wie in den vorangegangenen Abschnitten wird wieder vorausgesetzt, dass die vorberechneten Punkte nicht mehr als 50 KB belegen. Es gilt somit, dass $2 \cdot n \cdot v(2^h - 1) \leq 50 \cdot 2^{13}$. Die Anzahl der möglichen Parameterkombinationen von h und v ist sehr groß. Durch die Grundkomplexität der Hauptrechnung $\lceil \frac{n}{h} \rceil$ ADD und $\lceil \frac{n}{hv} \rceil - 1$ DBL ist es am günstigsten, h und v maximal zu wählen.

LimLee uvP, 140 Bit					
Koordinaten			h	v	hochg. in ms
C	D	E			
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	2	1.6
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	2	1.65
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	10	2	1.73
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	2	1.82
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	10	2	1.85
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	2	**1.6**
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	2	1.6
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	10	2	1.6
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	10	2	1.6
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	2	1.6
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}$	10	2	1.6
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	10	2	1.6
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	10	2	1.6

Tabelle 9.20: Hochgerechnete Laufzeit LimLee (ALG 6.1.6) unter Verwendung vorberechneter Punkte, 140 Bit, in ms

Tabellen 9.20 bis 9.24 fassen Auszüge der Ergebnisse zusammen. Auffällig ist bei diesen Ergebnissen, dass viele unterschiedliche Ausprägungen scheinbar die gleiche Laufzeit haben. Das stimmt natürlich nur bedingt. Es handelt sich bei dieser Methode um eine sehr kleine Anzahl von Punktadditionen und vor allem Punktverdopplungen. Daher fallen Unterschiede in der Hochrechnung noch viel geringer aus als in den vorangegangenen Methoden. Und da diese Unterschiede so klein sind, werden sie in der Berechnung weggerundet.

Zu beobachten ist jedoch, dass Punktverdopplungen entweder mit Jacobischen oder Modifiziert Jacobischen Koordinaten ausgeführt werden, wogegen Additionen mit Jacobischen oder Chudnowsky Jacobischen verwendet werden.

9.2.6 Vergleich der hochgerechneten Laufzeiten der Methoden für $k \cdot P$ unter Verwendung vorberechneter Punkte

Die jeweils besten, hochgerechneten Zeiten können nun wieder untereinander verglichen werden, was in Abbildung 9.2 geschieht. Es ist deutlich, dass das LimLee-Verfahren das mit Abstand beste ist. Zwar ist die Vorberechnung bei dieser Metho-

LimLee uvP, 160 Bit					
Koordinaten			h	v	hochg. in ms
C	D	E			
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	7	8	1.91
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	7	8	1.92
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	7	8	2.09
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	7	8	2.33
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	7	8	2.56
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	7	8	**1.91**
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	7	8	1.91
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	7	8	1.91
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	7	8	1.91
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	7	8	1.91
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}$	7	8	1.91
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	7	8	1.91
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	7	8	1.91

Tabelle 9.21: Hochgerechnete Laufzeit LimLee (ALG 6.1.6) unter Verwendung vorberechneter Punkte, 160 Bit, in ms

LimLee uvP, 192 Bit					
Koordinaten			h	v	hochg. in ms
C	D	E			
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	6	16	2.92
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	6	16	2.92
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	6	16	3.17
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	6	16	3.62
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	6	16	4.19
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	6	16	**2.92**
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	6	16	2.92
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	6	16	2.92
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	6	16	2.92
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	6	16	2.92
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	6	16	2.92
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	6	16	2.92
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	6	16	2.92

Tabelle 9.22: Hochgerechnete Laufzeit LimLee (ALG 6.1.6) unter Verwendung vorberechneter Punkte, 192 Bit, in ms

LimLee uvP, 197 Bit					
Koordinaten			h	v	hochg. in ms
C	D	E			
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	4	3.34
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	8	4	3.39
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	8	4	3.69
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	9	2	3.95
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	8	4	4.08
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	4	**3.34**
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	4	3.34
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	4	3.34
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	4	3.34
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	4	3.34
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}$	8	4	3.34
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	4	3.34
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	4	3.34

Tabelle 9.23: Hochgerechnete Laufzeit LimLee (ALG 6.1.6) unter Verwendung vorberechneter Punkte, 197 Bit, in ms

LimLee uvP, 272 Bit					
Koordinaten			h	v	hochg. in ms
C	D	E			
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	3	6.63
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	8	3	6.76
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	8	3	7.37
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	8	3	7.71
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	8	3	8.97
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	3	**6.63**
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	3	6.63
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	3	6.63
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	3	6.63
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	3	6.63
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	3	6.63
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}$	8	3	6.63
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	3	6.63

Tabelle 9.24: Hochgerechnete Laufzeit LimLee (ALG 6.1.6) unter Verwendung vorberechneter Punkte, 272 Bit, in ms

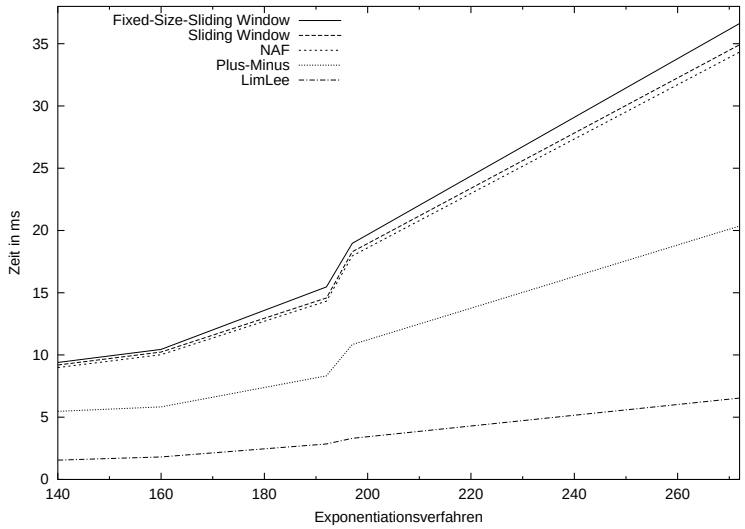


Abbildung 9.2: Vergleich der hochgerechneten Laufzeit von $k \cdot P$ unter Verwendung vorberechneter Punkte

de das aufwendigste, aber insgesamt das effizienteste.

9.3 Mehrfache Punktmultiplikation ohne Verwendung vorberechneter Punkte

In diesem Abschnitt werden die besten Kombinationen für mehrfache Punktmultiplikationen aus Abschnitt 6.2 berechnet, wenn die Verwendung vorberechneter Punkte nicht möglich ist.

9.3.1 Simultane Exponentiation ohne Verwendung vorberechneter Punkte

Auch im Fall der mehrfachen Punktmultiplikation lässt sich die hochgerechnete Laufzeit so herleiten wie für eine einfache Punktmultiplikation. Die Vorberechnung besteht aus genau einer Punktaddition:

Vorberechnung:
$A = P_1 + P_2$

P_1 und P_2 liegen affin vor. Daher ergibt sich nur ein Term zur Vorberechnug:

Vorberechnung:
$1 \times$ $R^A \leftarrow P_1^A + P_2^A$.

Die Hauptrechnung benötigt $3/4\ n$ Punktadditionen und $n-1$ Punktverdopplungen.

Hauptrechnung:

```
R ← O
for i = n − 1 down to 0
   R ← 2 · R
   if k1[i] = 1 then
      if k2i] = 1 then
         R ← R + A
      else
         R ← R + P1
   else
      if k2[i] = 1then
         R ← R + P2
return R
```

Bei $1/3$ aller $3/4 \cdot n$ Additionen ist der zweite Summand von Typ A, beim Rest affin. Jede Addition bekomme den ersten Summanden in Koordinaten der Art *C* und gebe das Ergebnis in Koordinaten der Art *D* zurück (M1). Die erste Addition ist nur eine Zuweisung. Aufeinanderfolgende Punktverdopplungen (M4) seien von der Art $R^B \leftarrow 2 \cdot R^B$. Vor jeder echten Punktaddition, nach einer Verdopplung, gibt es eine Punktverdopplung der Art $R^C \leftarrow 2 \cdot R^B$ (M5) und danach eine der Art $R^B \leftarrow 2 \cdot R^D$ (M3). M2 nehme einen Punkt der Art *D* und gebe einen der Art *C* wieder.

Wie oft jede einzelne *Maschine* verwendet wird lässt sich folgender Aufstellung entnehmen:

M1: Addition											$\frac{3}{4} \cdot n$
M2:	11_2	10_2	01_2	11_2	11_2	10_2	00_2	00_2	11_2		
	11_2	11_2	11_2	01_2	10_2	01_2	11_2	10_2	00_2	$\rightarrow$	$\#\frac{9}{16} \cdot (n-1)$
M3:	10_2	10_2	00_2								
	10_2	00_2	10_2							$\rightarrow$	$\#\frac{3}{16} \cdot (n-1)$

M4: 00_2
00_2 $\rightarrow$ $\#\frac{1}{16} \cdot (n-1)$

M5: 01_2 00_2 01_2
01_2 00_2 01_2 $\rightarrow$ $\#\frac{3}{16} \cdot (n-1)$

Es ergibt sich folgende Komplexitätsberechnung:

Vorberechnung:

$1 \times R^A \leftarrow P_1^{\mathcal{A}} + P_2^{\mathcal{A}}.$

Hauptrechnung:

$\frac{1}{3} \times R^D \leftarrow A^A +$
$\frac{2}{3} \times R^D \leftarrow A^{\mathcal{A}} +$
$\frac{1}{3} \cdot (\frac{3}{4}n - 1) \times R^D \leftarrow R^C + A^{\mathcal{A}} +$
$\frac{2}{3} \cdot (\frac{3}{4}n - 1) \times R^D \leftarrow R^C + A^{\mathcal{A}} +$
$\frac{9}{16}(n-1) \times R^C \leftarrow 2 \cdot R^D +$
$\frac{3}{16}(n-1) \times R^B \leftarrow 2 \cdot R^D +$
$\frac{1}{16}(n-1) \times R^B \leftarrow 2 \cdot R^B +$
$\frac{3}{16}(n-1) \times R^C \leftarrow 2 \cdot R^B.$

In Tabelle 9.25 werden die Ergebnisse zusammengefasst. Bei allen Bitlängen ist die Ausprägung die beste, die die einzelnen vorberechneten Punkt affin abspeichert, die Punktverdopplungen mit Modifiziert Jacobischen Koordinaten berechnet und für die Additionen Jacobische oder Chudnowsky Jacobische Koordinaten verwendet. Im Vergleich zu den bisher untersuchten Methoden schneidet die Berechnung dieser mit Jacobischen Koordinaten besser ab, als die mit Modifiziert Jacobischen. Das lässt sich damit begründen, dass diese Methode nur $n-1$ Punktverdopplungen braucht, aber mehr als $n/2$ Additionen. Für Punktadditionen sind Modifiziert Jacobische ungünstig.

9.3.2 Simultane 2^w-ary Exponentiation ohne Verwendung vorberechneter Punkte

Diese Methode ist im Wesentlichen die Fixed-Size-Sliding-Window Methode für zwei Exponenten. Daher lässt sich die Vorberechnung des Algorithmus 6.2.2 ebenfalls auf zwei Arten durchführen:

Simultane Exponentiation, ovP, alle Bitlängen								
Koordinaten				hochg. in ms				
A	B	C	D	140	160	192	197	272
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	20.58	22.47	31.03	39.45	74.19
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	20.92	22.89	31.56	40.05	75.32
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	21.07	23.0	31.76	40.42	76.02
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	22.91	25.07	34.54	43.95	82.66
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	22.62	28.84	42.84	46.47	96.36
$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	**18.95**	**20.78**	**28.64**	**36.18**	**68.05**
$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	18.95	20.78	28.65	36.19	68.06
$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	18.95	20.78	28.65	36.19	68.06
$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	18.95	20.79	28.65	36.2	68.07
$\mathcal{A}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	19.41	21.25	29.33	37.13	69.82
$\mathcal{A}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	19.41	21.25	29.32	37.12	69.81
$\mathcal{A}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	19.41	21.26	29.32	37.12	69.81
$\mathcal{A}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	19.42	21.26	29.33	37.13	69.82

Tabelle 9.25: Hochgerechnete Laufzeit Simultane Exponentiation (ALG 6.2.1) ohne Verwendung vorberechneter Punkte, alle Bitlängen, in ms

1. *Vorberechnung:*
   INPUT: w, P
   OUTPUT: $P_{i,j} = i \cdot P_1 + j \cdot P_2$ mit $0 \leq i, j < 2^w$
   $P_{0,0} \leftarrow \mathcal{O}$
   `for` $i = 0$ `to` $2^{w-1} - 1$
   $j = 2i$
   `for` $o = 0$ `to` $2^{w-1} - 1$
   $p = 2o$
   a $P_{j,p} \leftarrow 2 \cdot P_{i,o}$
   b $P_{j,p+1} \leftarrow P_{j,p} + P_1$
   $o = 2i + 1$
   `for` $p = 0$ `to` $2^w - 1$
   c $P_{o,p} \leftarrow P_{o-1,p} + P_2$

2. *Vorberechnung:*
   INPUT: w, P
   OUTPUT: $P_{i,j} = i \cdot P_1 + j \cdot P_2$ mit $0 \leq i, j < 2^w$
   $P_{0,0} \leftarrow \mathcal{O}$
   `for` $i = 0$ `to` $2^w - 1$

```
        for j = 1 to 2^w − 1
a           P_{i,j} ← P_{i,j−1} + P_1
        if (i < 2^w − 1)
b           P_{i+1,0} ← P_{i,0} + P_2
```

Die erste Art benötigt $2^{2(w-1)} - 1$ Punktverdopplungen und $2^{2w} - 2^{2(w-1)} - 2$ Punktadditionen, die zweite Art dagegen $(2^w - 1)^2 + 2^w - 1 = 2^{2w} - 2^w$ Punktadditionen. Bei jeder dieser Additionen liegt einer der Summanden, P_1 oder P_2, affin vor. Das kann gegenüber der ersten Art ein Vorteil sein, da hier ein großer Teil der Operationen Punktverdopplungen mit nicht-Affinen Koordinaten ist. Das wurde bereits anhand des Fixed-Size-Sliding-Windows Algorithmus in Abschnitt 9.1.2 diskutiert.

In den folgenden zwei Abschnitten werden die Komplexitäten der 2 Arten hergeleitet.

Vorberechnung nach Art 1:

Wir lassen zwei verschiedene Koordinatensysteme zu: Die erste Punktaddition ist nur eine Zuweisung. Die Punktverdopplung in Schritt a gebe das Ergebnis in der Form A zurück, die Punktadditionen in den Schritten b und c in der Form B. Die erste Addition in Schritt b ist von der Form $P^{\mathcal{A}}_{0,1} \leftarrow \mathcal{O} + P^{\mathcal{A}}_1$, die restlichen $2^{2w-2} - 1$ Additionen in Schritt b sind von der Form $P^{B}_{j,p+1} \leftarrow P^{\mathcal{A}}_{j,p} + P^{\mathcal{A}}_1$.

Die erste Punktverdopplung in Schritt a ist von der Form $\mathcal{O} \leftarrow 2 \cdot \mathcal{O}$, die für $i = 0,\ o = 1$ sind von der Form $P^{\mathcal{A}}_{0,2} \leftarrow 2 \cdot P^{\mathcal{A}}_{0,1}$ und die restlichen $2^{2w-2} - 1$ Punktverdopplungen teilen sich auf in die der Form $P^{\mathcal{A}}_{j,p} \leftarrow 2 \cdot P^{\mathcal{A}}_{i,o}$, wenn i gerade, und $P^{\mathcal{A}}_{j,p} \leftarrow 2 \cdot P^{B}_{i,o}$, wenn i ungerade.

Die erste Addition in Schritt c ist von der Form $P^{\mathcal{A}}_{1,0} \leftarrow \mathcal{O} + P^{\mathcal{A}}_2$ und die zweite Addition ist von der Form $P^{B}_{1,1} \leftarrow P^{\mathcal{A}}_{0,1} + P^{\mathcal{A}}_2$. Die restlichen $2^{2w-1} - 2$ Additionen teilen sich in die Form $P^{B}_{o,p} \leftarrow P^{\mathcal{A}}_{o-1,p} + P^{\mathcal{A}}_2$ und in die Form $P^{B}_{o,p} \leftarrow P^{B}_{o-1,p} + P^{\mathcal{A}}_2$. Es ergeben sich folgende Formeln:

Vorberechnung:

1	$\times$	$\mathcal{O} \leftarrow 2 \cdot \mathcal{O}$	$+$
2	$\times$	$P^{\mathcal{A}}_{-,-} \leftarrow 2 \cdot P^{\mathcal{A}}_{-,-}$	$+$
2	$\times$	$P^{\mathcal{A}}_{-,-} \leftarrow \mathcal{O} + P^{\mathcal{A}}_{-}$	$+$
1	$\times$	$P^{B}_{-,-} \leftarrow P^{\mathcal{A}}_{-,-} + P^{\mathcal{A}}_2$	$+$

$$
\begin{array}{lll}
(2^{2w-3}-2) & \times \; P^{\mathcal{A}}_{-,-} \leftarrow 2 \cdot P^{\mathcal{A}}_{-,-} & + \\
(2^{2w-3}-1) & \times \; P^{\mathcal{A}}_{-,-} \leftarrow 2 \cdot P^{\mathcal{B}}_{-,-} & + \\
(2^{2w-1}-2) & \times \; P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{A}}_{-,-} + P^{\mathcal{A}}_{2} & + \\
(2^{2w-2}-1) & \times \; P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{B}}_{-,-} + P^{\mathcal{A}}_{2}. &
\end{array}
$$

Am Ende liegen 2 Punkte in Affinen Koordinaten vor, nämlich $P_{0,\,1}$ und $P_{1,\,0}$, $2^{2w-2}-1$ Punkte in Koordinaten der Art A und $3 \cdot 2^{2w-2}-2$ Punkte in Koordinaten der Art B.

Vorberechnung nach Art 2:

Diese Art besteht fast ausschließlich aus Punktadditionen. Die einzigen Ausnahmen sind bei $i=0,\ j=2$, $P_{0,2} \leftarrow P_{0,\,1}(=P_1)+P_1 = 2 \cdot P_1$ (a) und bei $i=1$, $P_{2,\,0} \leftarrow P_{1,\,0}(=P_2)+P_2 = 2 \cdot P_2$ (b). Diese beiden Operationen sollen das Ergebnis in Koordinaten der Art A zurück geben, alle Additionen in Koordinaten der Form B. Es ergibt sich folgende Komplexität:

Vorberechnung:

$$
\begin{array}{lll}
2 & \times \; P^{\mathcal{A}}_{-,-} \leftarrow \mathcal{O} + P^{\mathcal{A}}_{-} & + \\
2 & \times \; P^{\mathcal{A}}_{-,-} \leftarrow 2 \cdot P^{\mathcal{A}}_{-} & + \\
3 & \times \; P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{A}}_{-,-} + P^{\mathcal{A}}_{1} & + \\
1 & \times \; P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{A}}_{-,-} + P^{\mathcal{A}}_{1} & + \\
(2^{2w}-9) & \times \; P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{B}}_{-,-} + P^{\mathcal{A}}_{-}. &
\end{array}
$$

Am Ende liegen 2 Punkte in Affinen Koordinaten vor, nämlich $P_{0,\,1}$ und $P_{1,\,0}$, 2 Punkte in Koordinaten der Art A, $P_{0,\,2}$ und $P_{2,\,0}$, und $2^{2w}-5$ Punkte der Art B.

Hauptrechnung

Die Hauptrechnung benötigt $\lfloor \frac{n-1}{w} \rfloor \cdot w$ Punktverdopplungen und im Schnitt $n \cdot \frac{1-\frac{1}{2^{2w}}}{w}$ Punktadditionen:

Hauptrechnung:
INPUT: k_1, P_1, k_2, P_2, w, $P_{i,\,j} = i \cdot P_1 + j \cdot P_2$ mit $0 \leq i,\ j < 2^w$
OUTPUT: $k_1 \cdot P_1 + k_2 \cdot P_2$

```
R ← O
for j = ⌊(n − 1)/w⌋w down to 0
   for i = 1 to w
      R ← 2 · R
   if (k1[j + w − 1 . . . j], (k2[j + w − 1 . . . j]) ≠ (0, 0) then
      R ← R + P_{k1[j+w−1...j],(k2[j+w−1...j])}
return R
```

Mehrfach hintereinander ausgeführte Punktverdopplungen seien vom Typ $D \leftarrow 2 \cdot D$. Die Verdopplungen vor einer Addition seien vom Typ $E \leftarrow 2 \cdot D$ und die nach einer Addition vom Typ $D \leftarrow 2 \cdot F$. Von den insgesamt $\lfloor \frac{n-1}{w} \rfloor \cdot w$ Punktverdopplungen sind also nur $\lfloor \frac{n-1}{w} \rfloor \cdot w - 2 \cdot n \cdot \frac{1-\frac{1}{2^{2w}}}{w}$ Punktverdopplungen vom Typ $D \leftarrow 2 \cdot D$, der Rest teilt sich in solche vom Typ $E \leftarrow 2 \cdot D$ und vom Typ $D \leftarrow 2 \cdot F$ auf. Die erste Punktaddition ist wiederum nur eine Zuweisung. Je nach Art der Vorberechnung setzen sich die Additionen wie folgt zusammen:

1. Art: Von den $2^w - 1$ vorberechneten Punkten $P_{i,\,j}$, $1 \leq i,\ j < 2^w$ sind 2 affin. Weitere $2^{2w-2} - 1$ Punkte liegen in Koordinaten der Art A vor und $3 \cdot 2^{2w-2} - 2$ Punkte in Koordinaten der Art B. Die Wahrscheinlichkeit, dass ein Affiner Punkt aufaddiert wird, ist somit $P(P^{\mathcal{A}}_{-,\,-}) = 2/(2^{2w} - 1)$. Die Wahrscheinlichkeit, dass ein Punkt der Art A aufaddiert wird, ist $P(P^{A}_{-,\,-}) = (2^{2w-2} - 1)/(2^{2w} - 1)$. Die Wahrscheinlichkeit eines Aufaddierens eines Punktes der Art B ist $P(P^{B}_{-,\,-}) = (3 \cdot 2^{2w-2} - 2)/(2^{2w} - 1)$.

2. Art: 2 Punkte liegen affin vor, 2 in Koordinaten der Art A, $2^{2w} - 5$ Punkte liegen in Koordinaten der Art B vor. Die Wahrscheinlichkeit des Aufaddierens eines Affinen Punktes ist also die gleiche wie die des Aufaddierens eines Punktes der Art A: $P(P^{\mathcal{A}}_{-,\,-}) = P(P^{A}_{-,\,-}) = 2/(2^{2w} - 1)$. Die Wahrscheinlichkeit, dass ein Punkt der Art B aufaddiert wird, ist $P(P^{B}_{-,\,-}) = (2^{2w} - 5)/(2^{2w} - 1)$.

Zusammen mit diesen Berechnungen ergibt sich folgende Komplexität:

1. Vorberechnung:

1	×	$\mathcal{O} \leftarrow 2 \cdot \mathcal{O}$	+
2	×	$P^{\mathcal{A}}_{-,\,-} \leftarrow 2 \cdot P^{\mathcal{A}}_{-,\,-}$	+
2	×	$P^{\mathcal{A}}_{-,\,-} \leftarrow \mathcal{O} + P^{\mathcal{A}}_{-}$	+
1	×	$P^{B}_{-,\,-} \leftarrow P^{\mathcal{A}}_{-,\,-} + P^{\mathcal{A}}_{2}$	+

$$
\begin{array}{lcll}
(2^{2w-3}-2) & \times & P^{\mathcal{A}}_{-,-} \leftarrow 2 \cdot P^{\mathcal{A}}_{-,-} & + \\
(2^{2w-3}-1) & \times & P^{\mathcal{A}}_{-,-} \leftarrow 2 \cdot P^{\mathcal{B}}_{-,-} & + \\
(2^{2w-1}-2) & \times & P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{A}}_{-,-} + P^{\mathcal{A}}_{2} & + \\
(2^{2w-2}-1) & \times & P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{B}}_{-,-} + P^{\mathcal{A}}_{2}. &
\end{array}
$$

Hauptrechnung:

$$
\begin{array}{lcll}
(\lfloor \frac{n-1}{w} \rfloor \cdot w - 2 \cdot (n \frac{1-\frac{1}{2^{2w}}}{w} - 1)) & \times & R^{C} \leftarrow 2 \cdot R^{C} & + \\
(n \cdot \frac{1-\frac{1}{2^{2w}}}{w} - 1) & \times & R^{D} \leftarrow 2 \cdot R^{C} & + \\
(n \cdot \frac{1-\frac{1}{2^{2w}}}{w} - 1) & \times & R^{C} \leftarrow 2 \cdot R^{E} & + \\
\frac{2}{2^{2w}-1} \cdot (n \frac{1-\frac{1}{2^{2w}}}{w} - 1) & \times & R^{E} \leftarrow R^{D} + P^{\mathcal{A}}_{-} & + \\
\frac{2^{2w-2}-1}{2^{2w}-1} \cdot (n \frac{1-\frac{1}{2^{2w}}}{w} - 1) & \times & R^{E} \leftarrow R^{D} + P^{\mathcal{A}}_{-,-} & + \\
\frac{3 \cdot 2^{2w-2}-2}{2^{2w}-1} \cdot (n \frac{1-\frac{1}{2^{2w}}}{w} - 1) & \times & R^{E} \leftarrow R^{D} + P^{\mathcal{B}}_{-,-} & + \\
\frac{1}{2^{2w}} & \times & R^{\mathcal{A}} \leftarrow R^{C} & + \\
\frac{2^{2w}-1}{2^{2w}} & \times & R^{\mathcal{A}} \leftarrow R^{E}. &
\end{array}
$$

2. Vorberechnung:

$$
\begin{array}{lcll}
2 & \times & P^{\mathcal{A}}_{-,-} \leftarrow \mathcal{O} + P^{\mathcal{A}}_{-} & + \\
2 & \times & P^{\mathcal{A}}_{-,-} \leftarrow 2 \cdot P^{\mathcal{A}}_{-} & + \\
3 & \times & P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{A}}_{-,-} + P^{\mathcal{A}}_{1} & + \\
1 & \times & P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{A}}_{-,-} + P^{\mathcal{A}}_{1} & + \\
(2^{2w}-9) & \times & P^{\mathcal{B}}_{-,-} \leftarrow P^{\mathcal{B}}_{-,-} + P^{\mathcal{A}}_{-}. &
\end{array}
$$

Hauptrechnung:

$$
\begin{array}{lcll}
(\lfloor \frac{n-1}{w} \rfloor \cdot w - 2 \cdot (n \frac{1-\frac{1}{2^{2w}}}{w} - 1)) & \times & R^{C} \leftarrow 2 \cdot R^{C} & + \\
n \cdot \frac{1-\frac{1}{2^{2w}}}{w} & \times & R^{D} \leftarrow 2 \cdot R^{C} & + \\
n \cdot \frac{1-\frac{1}{2^{2w}}}{w} & \times & R^{C} \leftarrow 2 \cdot R^{E} & + \\
\frac{2}{2^{2w}-1} \cdot (n \frac{1-\frac{1}{2^{2w}}}{w} - 1) & \times & R^{E} \leftarrow R^{D} + P^{\mathcal{A}}_{-} & + \\
\frac{2}{2^{2w}-1} \cdot (n \frac{1-\frac{1}{2^{2w}}}{w} - 1) & \times & R^{E} \leftarrow R^{D} + P^{\mathcal{A}}_{-,-} & + \\
\frac{2^{2w}-5}{2^{2w}-1} \cdot (n \frac{1-\frac{1}{2^{2w}}}{w+1} - 1) & \times & R^{E} \leftarrow R^{D} + P^{\mathcal{B}}_{-,-} & +
\end{array}
$$

$\frac{1}{2^{2w}}$	$\times$	$R^{\mathcal{A}} \leftarrow R^C$	$+$
$\frac{2^{2w}-1}{2^{2w}}$	$\times$	$R^{\mathcal{A}} \leftarrow R^E$.	

Die Additionen sind ihrem Vorkommen nach gewichtet: Es gibt je $(2^{2w-2}-1)$ vorberechnete Punkte vom Typ A und B und $(2^{2w-1}-1)$ vom Typ C. Weiter liegen die Punkte P_1 und P_2 *vorberechnet* in Affiner Darstellung vor.

Tabelle 9.26 fasst die wichtigsten Ergebnisse zusammen.

9.3.3 Simultane sliding window Exponentiation ohne Verwendung vorberechneter Punkte

Diese Methode benötigt zur Vorberechnung $2^{2w} + 2^{2w-2} - 2$ Punktadditionen und 2 Punktverdopplungen:

Vorberechnung:
INPUT: w, P
OUTPUT: $P_{i,\,j} = i \cdot P_1 + j \cdot P_2$ mit $0 \leq i, j < 2^w$,
wo i oder j ungerade ist.

a $P_{12} \leftarrow 2 \cdot P_1$, $P_{22} \leftarrow 2 \cdot P_2$
$P_{1,\,0} \leftarrow P_1$, $P_{0,\,1} \leftarrow P_2$
`for` $i = 3$ `to` $2^w - 1$ `step` 2
b $P_{0,\,i} \leftarrow P_{0,\,i-2} + P_{22}$, $P_{i,\,0} \leftarrow P_{i-2,\,0} + P_{12}$
`for` $i = 1$ `to` $2^w - 1$ `step` 2
`for` $j = 1$ `to` $2^w - 1$
c $P_{i,\,j} \leftarrow P_{i,\,j-1} + P_2$
`for` $i = 2$ `to` $2^w - 1$ `step` 2
`for` $j = 1$ `to` $2^w - 1$ `step` 2
d $P_{i,\,j} \leftarrow P_{i-1,\,j} + P_1$

Die 2 Punktverdopplungen in Schritt a sollen die Ergebnispunkte im Koordinatensystem A zurückgeben, also $P^{\mathcal{A}}_{--} \leftarrow 2 \cdot P^{\mathcal{A}}_{-}$.

Die Punktadditionen in den Schritten b, c und d sollen die Punkte im Koordinatensystem B wiedergeben. Für $i = 3$ in Schritt b haben die beiden Additionen die Form $P^{B}_{-,\,-} \leftarrow P^{\mathcal{A}}_{-,\,-} + P^{A}_{--}$. Die restlichen $(2 \cdot (2^{w-1} - 2))$ Additionen sind von der Form $P^{B}_{-,\,-} \leftarrow P^{B}_{-,\,-} + P^{A}_{--}$.$P_{0,\,i} \leftarrow P_{0,\,i-2} + P_{22}$

Für $i = j = 1$ in Schritt c gilt $P^{B}_{1,\,1} \leftarrow P^{\mathcal{A}}_{1,\,0} + P^{\mathcal{A}}_{2}$. Es folgen $(2^{2w-1} - 2^{w-1} - 1)$ weitere Additionen der Form $P^{B}_{-,\,-} \leftarrow P^{B}_{-,\,-} + P^{\mathcal{A}}_{2}$ in Schritt c.

Simultaneous 2^w-ary, ovP, alle Bitlängen											
Koordinaten					Art	w	hochg. in ms				
A	B	C	D	E			140	160	192	197	272
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	1	2 3	18.75	20.4	27.88	35.24	 64.77
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	1	2 3	19.04	20.68	28.36	36.0	 66.71
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	1, 2	2	19.27	20.92	28.68	36.43	67.85
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2	2	20.92	22.78	31.15	39.53	73.65
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	1	2	20.1	25.47	37.52	40.63	83.33
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	1, 2	2	**16.34**	**18.03**	**24.64**	**30.58**	**56.93**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	1, 2	2	16.34	18.03	24.64	30.58	56.93
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	1, 2	2	16.34	18.03	24.64	30.58	56.93
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	1, 2	2	16.34	18.03	24.64	30.58	56.93
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	2	16.67	18.33	25.02	31.13	57.83
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	2	16.67	18.33	25.02	31.13	57.83
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	2	16.67	18.33	25.02	31.13	57.83
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	2	16.67	18.33	25.02	31.13	57.83

Tabelle 9.26: Hochgerechnete Laufzeit 2^w-ary Exponentiation (ALG 6.2.2) ohne Verwendung vorberechneter Punkte, alle Bitlängen, in ms

Schritt d besteht aus $(2^{2w-2} - 2^{w-1})$ Punktadditionen, ebenfalls der Form $P^B_{-,-} \leftarrow P^B_{-,-} + P^{\mathcal{A}}_1$.

Damit ergibt sich folgende Komplexität:

Vorberechnung:

2	$\times$	$P^{\mathcal{A}}_{--} \leftarrow 2 \cdot P^{\mathcal{A}}_{-}$	$+$
2	$\times$	$P^B_{-,-} \leftarrow P^{\mathcal{A}}_{-,-} + P^{\mathcal{A}}_{--}$	$+$
1	$\times$	$P^B_{1,1} \leftarrow P^{\mathcal{A}}_{1,0} + P^{\mathcal{A}}_{2}$	$+$
$2 \cdot (2^{w-1} - 2)$	$\times$	$P^B_{-,-} \leftarrow P^B_{-,-} + P^{\mathcal{A}}_{--}$	$+$
$(3 \cdot 2^{2w-2} - 2^w - 1)$	$\times$	$P^B_{-,1} \leftarrow P^B_{-,0} + P^{\mathcal{A}}_{2}$.	

Am Ende sind von den insgesamt $(3 \cdot 2^{2w-2})$ Punkten 2 affin und $(3 \cdot 2^{2w-2} - 2)$ Punkte sind von der Art B.

Die Hauptrechnung besteht im Schnitt aus $(n/(w+1/3))$ Punktadditionen und wie bei den anderen Window-Algortihmen auch aus $(n-1)$ oder $(n-(n\%w))$ Punktverdopplungen:

Hauptrechnung:
INPUT: w, P_1, P_2, k_1, k_2,
$P_{i,j} = i \cdot P_1 + j \cdot P_2$ mit $0 \le i, j < 2^w$,
wo i oder j ungerade ist.
OUTPUT: $k_1 \cdot P_1 + k_2 \cdot P_2$

$R \leftarrow \mathcal{O}$
$j \leftarrow n-1$
`while` $j \ge 0$
  `if` $k_1[j] = 0$ `and` $k_2[j] = 0$
    $R \leftarrow 2 \cdot R$, $j \leftarrow j - 1$
  `else`
    $j_{\text{new}} \leftarrow \max(j - w,\ -1)$
    $h \leftarrow j_{\text{new}} + 1$
    `while` $k_1[h] = 0$ `and` $k_2[h] = 0$
      $h \leftarrow h + 1 \quad \rightarrow \quad j \ge h > j_{\text{new}}$
    $K_1 \leftarrow k_1[j \ldots h]$, $K_2 \leftarrow K_2[j \ldots h]$
    `while` $j \ge h$
      $R \leftarrow 2 \cdot R$, $j \leftarrow j - 1$
    $R \leftarrow R + P_{K,L}$
    `while` $j > j_{\text{new}}$

```
            R ← 2 · R, j ← j − 1
    return R
```

Die erste Punktaddition ist eine Zuweisung. Hintereinander ausgeführte Verdopplungen sollen das Ergebnis in Koordinaten der Art C zurückgeben. Verdopplungen direkt vor einer Addition sollen das Ergebnis in Koordinaten der Art D zurückgeben, Additionen in Koordinaten der Form E.

Die Anzahl der benötigten Verdopplungen muss auch hier wieder abgeschätzt werden, das geschieht analog zu der Abschätzung der einfachen Sliding-Window-Methode. Die Anzahl der benötigten Verdopplungen ist

$$t := \frac{3}{2^{2w} - 2^{2(w-1)}} \cdot (n-1) + \frac{9}{2^{2w} - 2^{2(w-1)}} \cdot (n-2) + \sum_{W=3}^{w} \frac{9 \cdot 2^{2(W-2)}}{2^{2w} - 2^{2(w-1)}} (n-W)$$

Es ergibt sich folgende Komplexität:

Vorberechnung:

2	$\times$	$P^{\mathcal{A}}_{--} \leftarrow 2 \cdot P^{\mathcal{A}}_{-}$	$+$
2	$\times$	$P^{B}_{-,-} \leftarrow P^{\mathcal{A}}_{-,-} + P^{A}_{--}$	$+$
1	$\times$	$P^{B}_{1,1} \leftarrow P^{\mathcal{A}}_{1,0} + P^{\mathcal{A}}_{2}$	$+$
$2 \cdot (2^{w-1} - 2)$	$\times$	$P^{B}_{-,-} \leftarrow P^{B}_{-,-} + P^{A}_{--}$	$+$
$(3 \cdot 2^{2w-2} - 2^{w} - 1)$	$\times$	$P^{B}_{-,1} \leftarrow P^{B}_{-,0} + P^{\mathcal{A}}_{2}$.	

Hauptrechnung:

$(t - 2 \cdot (\frac{n}{w+\frac{1}{3}} - 1))$	$\times$	$R^{C} \leftarrow 2 \cdot R^{C}$	$+$
$(\frac{n}{w+\frac{1}{3}} - 1)$	$\times$	$R^{D} \leftarrow 2 \cdot R^{C}$	$+$
$(\frac{n}{w+\frac{1}{3}} - 1)$	$\times$	$R^{C} \leftarrow 2 \cdot R^{E}$	$+$
$\frac{2}{3 \cdot 2^{2w-2}} \cdot (\frac{n}{w+\frac{1}{3}} - 1)$	$\times$	$R^{E} \leftarrow R^{D} + P^{\mathcal{A}}_{-}$	$+$
$\frac{3 \cdot 2^{2w-2} - 2}{3 \cdot 2^{2w-2}} \cdot (\frac{n}{w+\frac{1}{3}} - 1)$	$\times$	$R^{E} \leftarrow R^{D} + P^{B}_{-,-}$	$+$
$\frac{1}{4}$	$\times$	$R^{\mathcal{A}} \leftarrow R^{C}$	$+$
$\frac{3}{4}$	$\times$	$R^{\mathcal{A}} \leftarrow R^{E}$.	

Simultaneous Sliding Window, ovP, alle Bitlängen										
Koordinaten					w	hochg. in ms				
A	B	C	D	E		140	160	192	197	272
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	2	17.82	19.42	26.55	33.51	
					3					61.43
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	2	18.21	19.8	27.18	34.46	
					3					63.32
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	2	18.53	20.14	27.63	35.07	
					3					64.81
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2	20.13	21.94	30.03	38.08	
					3					70.44
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	2	19.42	24.63	36.31	39.33	80.77
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	**15.67**	**17.31**	**23.67**	**29.35**	**54.7**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	15.67	17.31	23.67	29.35	54.7
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	15.67	17.31	23.67	29.35	54.7
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	15.67	17.31	23.67	29.35	54.7
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	15.75	17.36	23.7	29.45	54.79
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	15.75	17.36	23.7	29.45	54.79
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	15.75	17.36	23.7	29.45	54.79
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	15.75	17.36	23.7	29.45	54.79

Tabelle 9.27: Hochgerechnete Laufzeit Simultaneous Sliding Window Exponentiation (ALG 6.2.3) ohne Verwendung vorberechneter Punkte, alle Bitlängen, in ms

9.3.4 Basic Interleaving Exponentiation ohne Verwendung vorberechneter Punkte

Die Vorberechnung dieser Methode benötigt $(2^{w_1-1}+2^{w_2-1}-2)$ Punktadditionen und 2 Punktverdopplungen, für $w_1, w_2 > 1$, also genau doppelt soviel Operationen wie die Exponentiation mit Non-adjacent-forms. Es ist dieselbe Vorberechnung, nur für zwei Punkte P_1 und P_2 und zwei Windowgrößen w_1 und w_2:

Vorberechnung:
INPUT: w_1, w_2, P_1, P_2
OUTPUT: $P_{1j} = j \cdot P_1$ und $P_{2j} = j \cdot P_2$ mit $0 \leq j < 2^w_{1,2}$,
j ungerade
$P_{12} \leftarrow 2 \cdot P_1$, $P_{22} \leftarrow 2 \cdot P_2$
`for` $i = 3$ `to` 2^{w_1} `step` 2
$P_{1i} \leftarrow P_{1i-1} + P_{12}$
`for` $i = 3$ `to` 2^{w_2} `step` 2
$P_{2i} \leftarrow P_{2i-1} + P_{22}$

Die Komplexität wird direkt von der des Algorithmus 6.1.4 abgeleitet:

Vorberechnung:

2	$\times$	$t^A \leftarrow 2 \cdot P^A$	$+$
2	$\times$	$P_1^B \leftarrow P_0^A + t^A$	$+$
$2^{w_1-1} + 2^{w_2-1} - 4$	$\times$	$P_-^B \leftarrow P_-^B + t^A$.	

Die Hauptrechnung benötigt $n/(w_1+1)$ und $n/(w_2+1)$ Punktadditionen und - so wie beim Sliding-Window-Verfahren - aus $n-w$ bis $n-1$ Verdopplungen:

Hauptrechnung:
INPUT: w_1, w_2, P_1, P_2, $P_{1j} = j \cdot P_1$ und
$P_{2j} = j \cdot P_2$ mit $0 \leq j < 2^w_{1,2}$, j ungerade
OUTPUT: $k_1 \cdot P_1 + k_2 \cdot P_2$
$R \leftarrow \mathcal{O}$
`for` $i = 1$ `to` 2
$window\_handle_i \leftarrow$ `null`
`for` $j = n-1$ `down to` 0
$R \leftarrow 2 \cdot R$
`for` $i = 1$ `to` 2

```
        if window_handle_i = null and k_i[j] = 1
            J ← j − w_i + 1
            while k_i[J] = 0
                J ← J + 1     → j ≥ J > j − w und J ≥ 0
            window_handle_i ← J
            K_i ← K_i[j ... J]
        if window_handle_i = j
            R ← R + (E_i · P_i)     (mittels Tabelleneintrag)
            window_handle_i ← null
    return R
```

Bei dieser Methode kann nicht abgeschätzt werden, wie Additionen und Verdopplungen verteilt sind: Bei allen bisherigen Methoden folgt jeder Punktaddition eine Punktverdopplung, ausser man ist am Ende des Algorithmus angelangt oder es handelt sich um die Methode LimLee. Das ist bei dieser sowie der nächsten Methode nicht unbedingt der Fall. Die Additionen werden für jeden Exponenten wie mit der Sliding Window Methode getrennt ausgeführt, die Verdopplung dagegen geschieht simultan. Weiters gibt es nicht eine feste Fenstergrößen wie bei der Fixed Size Sliding Window Methode, sondern die zwei Fenstergrößen können jeden Wert zwischen 0 und w_i, $i = 1, 2$, annehmen. Die beste Abschätzung, die man machen kann, ist, die Vorberechnung wie bereits gezeigt zu optimieren und für die Hauptrechnung nur ein Koordinatensystem zuzulassen und hierfür die beste Wahl zu treffen.

Die Anzahl der Verdopplungen wird genauso wie bei der Simultanen Sliding-Window hergeleitet, wobei nur die kleinere der beiden Fenstergrößen w_1 und w_2 in die Rechnung einfließt. Sei $w = \min(w_1,\ w_2)$. Dann wird die Anzahl der Verdopplungen wie folgt gewichtet:

$$t := \frac{3}{2^{2w} - 2^{2(w-1)}} \cdot (n-1) + \frac{9}{2^{2w} - 2^{2(w-1)}} \cdot (n-2) + \sum_{W=3}^{w} \frac{9 \cdot 2^{2(W-2)}}{2^{2w} - 2^{2(w-1)}} (n-W)$$

Die Komplexitätsformeln setzen sich wie folgt zusammen:

Vorberechnung:

2	$\times$	$t^A \leftarrow 2 \cdot P^A$	$+$
2	$\times$	$P_1^B \leftarrow P_0^A + t^A$	$+$
$2^{w_1-1} + 2^{w_2-1} - 4$	$\times$	$P_-^B \leftarrow P_-^B + t^A$.	

Basic Interleaving Exponentiation, ovP, 140 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	18.12
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	18.48
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	18.61
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	19.24
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	20.21
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	**15.95**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	16.21
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	16.21
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	16.36
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	16.36
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	16.41
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	16.62
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}$	4	4	16.62

Tabelle 9.28: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 140 Bit, in ms

Hauptrechnung:

$\frac{2}{2^{w_1-1}+2^{w_2-1}}$	$\times$	$R^C \leftarrow P^{\mathcal{A}}$	$+$	
$\frac{2^{w_1-1}+2^{w_2-1}-2}{2^{w_1-1}+2^{w_2-1}}$	$\times$	$R^C \leftarrow P^B$	$+$	
$\frac{2}{2^{w_1-1}+2^{w_2-1}}(\frac{n}{w_1+1}+\frac{n}{w_2+1}-1)$	$\times$	$R^C \leftarrow R^C + P^{\mathcal{A}}$	$+$	
$\frac{2^{w_1-1}+2^{w_2-1}-2}{2^{w_1-1}+2^{w_2-1}}(\frac{n}{w_1+1}+\frac{n}{w_2+1}-1)$	$\times$	$R^C \leftarrow R^C + P^B$	$+$	
t	$\times$	$R^C \leftarrow 2 \cdot R^C$	$+$	
1	$\times$	$R^{\mathcal{A}} \leftarrow R^C$.		

Tabellen 9.28 bis 9.32 fassen Auszüge der Ergebnisse zusammen.

Bei allen fünf Bitlängen wird das beste Ergebnis erzielt, wenn die vorberechneten Punkte affin gespeichert werden und die Operationen in der Hauptrechnung in Modifiziert Jacobischen Koordinaten berechnet werden. Wird nur ein Koordinatensystem verwendet, es ist am günstigsten, das Modifiziert Jacobische zu verwenden.

Basic Interleaving Exponentiation, ovP, 160 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	19.63
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	19.99
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	20.17
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	21.96
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	24.37
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	**17.65**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	17.96
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	17.96
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	18.07
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	18.07
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	18.21
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}$	4	4	18.3
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	18.33

Tabelle 9.29: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 160 Bit, in ms

Basic Interleaving Exponentiation, ovP, 192 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	26.68
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	27.31
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	27.58
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	29.96
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	35.88
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	**24.1**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	24.41
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	24.41
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	24.64
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	24.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	24.73
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	5	24.78
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	4	24.96

Tabelle 9.30: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 192 Bit, in ms

Basic Interleaving Exponentiation, ovP, 197 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	33.68
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	34.64
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	35.01
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	38.0
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	38.85
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	**29.8**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	29.95
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	29.95
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	30.02
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	30.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	30.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	5	30.88
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	30.88

Tabelle 9.31: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 197 Bit, in ms

Basic Interleaving Exponentiation, ovP, 272 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	5	5	61.99
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	5	5	64.1
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	5	5	64.72
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	5	5	70.25
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	5	5	78.99
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	**55.07**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	55.3
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	55.3
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	55.42
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	5	56.84
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	6	5	57.06
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	6	57.06
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	57.11

Tabelle 9.32: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 272 Bit, in ms

9.3.5 w-NAF-based Interleaving Exponentiation ohne Verwendung vorberechneter Punkte

Dieser Algorithmus benötigt die gleiche Vorberechnung wie der vorangegangene:

Vorberechnung:
INPUT: w_1, w_2, P_1, P_2
OUTPUT: $P_{1j} = j \cdot P_1$ und $P_{2j} = j \cdot P_2$ mit $0 \leq j < 2^w_{1,\,2}$,
j ungerade
$P_{12} \leftarrow 2 \cdot P_1$, $P_{22} \leftarrow 2 \cdot P_2$
`for` $i = 3$ `to` 2^{w_1} `step` 2
$P_{1i} \leftarrow P_{1i-1} + P_{12}$
`for` $i = 3$ `to` 2^{w_2} `step` 2
$P_{2i} \leftarrow P_{2i-1} + P_{22}$

Daher ist auch die Komplexität dieselbe:

Vorberechnung:

2	$\times$	$t^A \leftarrow 2 \cdot P^A$	$+$
2	$\times$	$P_1^B \leftarrow P_0^A + t^A$	$+$
$2^{w_1-1} + 2^{w_2-1} - 4$	$\times$	$P_-^B \leftarrow P_-^B + t^A$.	

Dieser Algorithmus benötigt nur $n/(w_1 + w_2 + 4)$ Additionen und $n - (n\%w)$ Verdopplungen wenn $n\%w \neq 0$ und $n - w$ Verdopplungen, sonst:

Hauptrechnung:
INPUT: w_1, w_2, P_1, P_2, $P_{1j} = j \cdot P_1$ und
$P_{2j} = j \cdot P_2$ mit $0 \leq j < 2^w_{1,\,2}$, j ungerade
OUTPUT: $k_1 \cdot P_1 + k_2 \cdot P_2$
$R \leftarrow \mathcal{O}$
`for` $i = 1$ `to` 2
$N_i[n], \ldots, N_i[n_i + 1] \leftarrow 0, \ldots, 0$
$N_i[n_i], \ldots, N_i[0] \leftarrow w$-NAF von k_i
`for` $j = n$ `down to` 0
$R \leftarrow 2 \cdot R$
`for` $i = 1$ `to` 2
`if` $N_i[j] = 0$
$R \leftarrow R + N_i[j] \cdot P_i$
`return` R

Auch bei diesem Algorithmus kann keine Aussage über die Verteilung der Additionen und Verdopplungen gemacht werden. Es wird daher wie im vorherigen Algorithmus verfahren. Es werden aber dagegen - wie bei der Exponentiation mit non-adjacent-forms - höchstens $n - w - 1$ mit $w = \min(w_1,\ w_2)$ Verdopplungen benötigt. Es ergibt sich folgende Komplexität:

Vorberechnung:

2		$\times$	$t^A \leftarrow 2 \cdot P^A$	$+$
2		$\times$	$P_1^B \leftarrow P_0^A + t^A$	$+$
$2^{w_1-1} + 2^{w_2-1} - 4$		$\times$	$P_-^B \leftarrow P_-^B + t^A$.	

Hauptrechnung:

$\frac{2}{2^{w_1-1}+2^{w_2-1}}$		$\times$	$R^C \leftarrow P^A$	$+$
$\frac{2^{w_1-1}+2^{w_2-1}-2}{2^{w_1-1}+2^{w_2-1}}$		$\times$	$R^C \leftarrow P^B$	$+$
$\frac{2}{2^{w_1-1}+2^{w_2-1}}(\frac{n}{w_1+2} + \frac{n}{w_2+2} - 1)$		$\times$	$R^C \leftarrow R^C + P^A$	$+$
$\frac{2^{w_1-1}+2^{w_2-1}-2}{2^{w_1-1}+2^{w_2-1}}(\frac{n}{w_1+2} + \frac{n}{w_2+2} - 1)$		$\times$	$R^C \leftarrow R^C + P^B$	$+$
t		$\times$	$R^C \leftarrow 2 \cdot R^C$	$+$
1		$\times$	$R^A \leftarrow R^C$.	

Tabellen 9.33 bis 9.37 fassen Auszüge der Ergebnisse zusammen.

Auch bei dieser Methode wird bei allen fünf Bitlängen das beste Ergebnis erzielt, wenn die vorberechneten Punkte affin gespeichert werden und die Operationen in der Hauptrechnung in Modifiziert Jacobischen Koordinaten berechnet werden. Auch ist wieder das Modifiziert Jacobische System die erste Wahl für die Berechnung mit nur einem Koordinatensystem.

9.4 Mehrfache Punktmultiplikation mit Vorberechnung

In diesem Abschnitt werden die Punktmultiplikationsmethoden für die Operation $k_1 \cdot P_1 + k_2 \cdot P_2$ betrachtet, die sich zur Signaturverifikation eignen: Die Basic Interleaving und die w-NAF-based Interleaving Exponentiations Methode. Nur sie bieten die Möglichkeit, vorweg berechnete Vielfache des Punktes P_1 in die Rechnung zu integrieren. P_1 ist im Falle der Signaturverifikation der Basispunkt G. P_2 ist vorher nicht bekannt, es können für diesen Punkt daher vor der Verifikation keine Vorberechnung angestellt werden. Die Vorberechnung reduziert sich mit beiden

w-NAF-based Interleaving Exponentiation, ovP, 140 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	3	3	16.76
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	3	3	17.28
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	17.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	18.26
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	19.1
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	**14.92**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	15.09
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	15.09
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	15.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	15.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	15.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	15.49
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	5	15.49

Tabelle 9.33: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 140 Bit, in ms

w-NAF-based Interleaving Exponentiation, ovP, 160 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	18.22
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	18.77
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	19.06
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	20.77
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	23.14
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	**16.55**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	16.71
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	16.71
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	16.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	17.0
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	17.0
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	17.05
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	17.12

Tabelle 9.34: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 160 Bit, in ms

w-NAF-based Interleaving Exponentiation, ovP, 192 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	24.75
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	25.65
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	26.07
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	28.35
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	34.08
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	**22.59**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	22.86
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	22.86
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	23.07
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	23.1
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	23.1
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	23.25
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	23.37

Tabelle 9.35: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 192 Bit, in ms

w-NAF-based Interleaving Exponentiation, ovP, 197 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	31.15
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	32.48
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	33.06
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	35.91
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	36.9
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	**27.83**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	28.25
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	28.25
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	28.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	28.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	28.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	28.69
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	28.72

Tabelle 9.36: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 197 Bit, in ms

w-NAF-based Interleaving Exponentiation, ovP, 272 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	57.45
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	60.19
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	61.39
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	66.7
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	75.69
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	**51.8**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	52.17
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	52.17
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	52.43
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	52.84
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	52.84
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	53.22
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	5	53.22

Tabelle 9.37: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 272 Bit, in ms

Methoden immerhin um die Hälfte.

9.4.1 Basic Interleaving Exponentiation unter Verwendung vorberechneter Punkte

Die Komplexität kann direkt aus Abschnitt 9.3.4 übernommen werden, mit $t := \frac{3}{2^{2w}-2^{2(w-1)}} \cdot (n-1) + \frac{9}{2^{2w}-2^{2(w-1)}} \cdot (n-2) + \sum_{W=3}^{w} \frac{9 \cdot 2^{2(W-2)}}{2^{2w}-2^{2(w-1)}}(n-W)$:

Vorberechnung:

$1 \quad \times \quad t^A \leftarrow 2 \cdot P^{\mathcal{A}} \quad +$

$1 \quad \times \quad P_1^B \leftarrow P_0^{\mathcal{A}} + t^A \quad +$

$2^{w_2-1} - 2 \quad \times \quad P_-^B \leftarrow P_-^B + t^A.$

Hauptrechnung:

$\frac{2^{w_1-1}+1}{2^{w_1-1}+2^{w_2-1}} \quad \times \quad R^C \leftarrow P^{\mathcal{A}} \quad +$

$\frac{2^{w_2-1}-1}{2^{w_1-1}+2^{w_2-1}} \quad \times \quad R^C \leftarrow P^B \quad +$

Basic Interleaving Exponentiation, uvP, 140 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	14.17
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	14.93
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	15.43
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	16.8
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	16.35
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	**13.1**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	13.27
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	13.36
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	13.45
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	13.47
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	13.5
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	13.54
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	13.57

Tabelle 9.38: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 140 Bit, in ms

$$
\begin{array}{ll}
\frac{n}{(w_1+1)} & \times \quad R^C \leftarrow R^C + P^{\mathcal{A}} + \\
\frac{1}{2^{w_2-1}} \cdot (\frac{n}{w_2+1} - 1) & \times \quad R^C \leftarrow R^C + P^{\mathcal{A}} + \\
\frac{2^{w_2-1}-1}{2^{w_2-1}} \cdot (\frac{n}{w_2+1} - 1) & \times \quad R^C \leftarrow R^C + P^{B} + \\
t & \times \quad R^C \leftarrow 2 \cdot R^C \quad + \\
1 & \times \quad R^{\mathcal{A}} \leftarrow R^C.
\end{array}
$$

Tabellen 9.38 bis 9.42 fassen die besten Kombinationen zusammen.

9.4.2 w-NAF-based, Interleaving Exponentiation unter Verwendung vorberechneter Punkte

Die Komplexität kann direkt aus Abschnitt 9.3.5 übernommen werden:

Vorberechnung:

$$
\begin{array}{ll}
1 & \times \quad t^A \leftarrow 2 \cdot P^{\mathcal{A}} \quad + \\
1 & \times \quad P_1^B \leftarrow P_0^{\mathcal{A}} + t^A \quad +
\end{array}
$$

Basic Interleaving Exponentiation, uvP, 160 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	15.58
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	16.36
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	16.91
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	18.47
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	20.91
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	**14.6**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	14.78
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	14.91
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	14.93
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.96
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	5	14.97
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	14.99
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	15.02

Tabelle 9.39: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 160 Bit, in ms

Basic Interleaving Exponentiation, uvP, 192 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	21.37
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	22.58
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	23.37
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	25.46
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	31.12
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	**20.1**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	20.34
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	20.41
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	5	20.46
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	20.54
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	20.57
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	20.6
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	20.64

Tabelle 9.40: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 192 Bit, in ms

Basic Interleaving Exponentiation, uvP, 197 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	26.81
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	28.53
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	29.6
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	32.21
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	33.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	**24.89**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	25.04
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	25.18
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	25.34
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	25.52
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	25.52
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	5	25.54
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	25.6

Tabelle 9.41: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 197 Bit, in ms

Basic Interleaving Exponentiation, uvP, 272 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	5	50.74
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	4	54.13
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	5	56.18
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	10	5	61.16
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	10	5	70.57
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	**47.31**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	47.43
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	5	47.9
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	48.02
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	5	48.17
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	48.28
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	48.38
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	48.43

Tabelle 9.42: Hochgerechnete Laufzeit Basic Interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 272 Bit, in ms

w-NAF-based Interleaving Exponentiation, uvP, 140 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	3	13.33
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	3	14.17
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	3	14.77
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	15.72
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	3	16.09
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	**12.45**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	12.56
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	12.61
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	3	12.71
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	12.74
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	12.76
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	12.77
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	12.77

Tabelle 9.43: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 140 Bit, in ms

$2^{w_2-1}-2$	$\times$	$P_-^B \leftarrow P_-^B + t^A.$

Hauptrechnung:

$\frac{2^{w_1-1}+1}{2^{w_1-1}+2^{w_2-1}}$	$\times$	$R^C \leftarrow P^{\mathcal{A}} \quad +$
$\frac{2^{w_2-1}-1}{2^{w_1-1}+2^{w_2-1}}$	$\times$	$R^C \leftarrow P^B \quad +$
$\frac{n}{(w_1+2)}$	$\times$	$R^C \leftarrow R^C + P^{\mathcal{A}} +$
$\frac{1}{2^{w_2-1}} \cdot (\frac{n}{w_2+2} - 1)$	$\times$	$R^C \leftarrow R^C + P^{\mathcal{A}} +$
$\frac{2^{w_2-1}-1}{2^{w_2-1}} \cdot (\frac{n}{w_2+2} - 1)$	$\times$	$R^C \leftarrow R^C + P^B +$
t	$\times$	$R^C \leftarrow 2 \cdot R^C \quad +$
1	$\times$	$R^{\mathcal{A}} \leftarrow R^C.$

Tabellen 9.43 bis 9.47 fassen die besten Kombinationen zusammen.

w-NAF-based Interleaving Exponentiation, uvP, 160 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	14.73
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	3, 4	15.62
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	16.22
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	17.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	20.12
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	**13.91**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.01
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	14.07
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	14.14
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.16
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	3	14.17
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.19
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.19

Tabelle 9.44: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 160 Bit, in ms

w-NAF-based Interleaving Exponentiation, uvP, 192 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	20.22
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	21.58
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	22.45
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	24.47
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	29.98
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	**19.16**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	19.36
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	19.37
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	19.48
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	19.53
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	19.56
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	19.56
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	19.57

Tabelle 9.45: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 192 Bit, in ms

w-NAF-based Interleaving Exponentiation, uvP, 197 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	25.31
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	27.24
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	28.4
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	30.93
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	32.5
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	**23.67**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	23.93
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	24.06
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	24.09
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	24.23
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	24.3
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	24.3
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	24.31

Tabelle 9.46: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 197 Bit, in ms

w-NAF-based Interleaving Exponentiation, uvP, 272 Bit					
Koordinaten			w_1	w_2	hochg.
A	B	C			in ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	4	47.94
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	4	51.72
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	4	54.03
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	10	4	58.84
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	10	4	68.27
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	**45.14**
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	45.52
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	45.66
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	45.99
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	5	46.03
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	3	46.07
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	46.1
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	5	46.14

Tabelle 9.47: Hochgerechnete Laufzeit w-NAF-based Interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 272 Bit, in ms

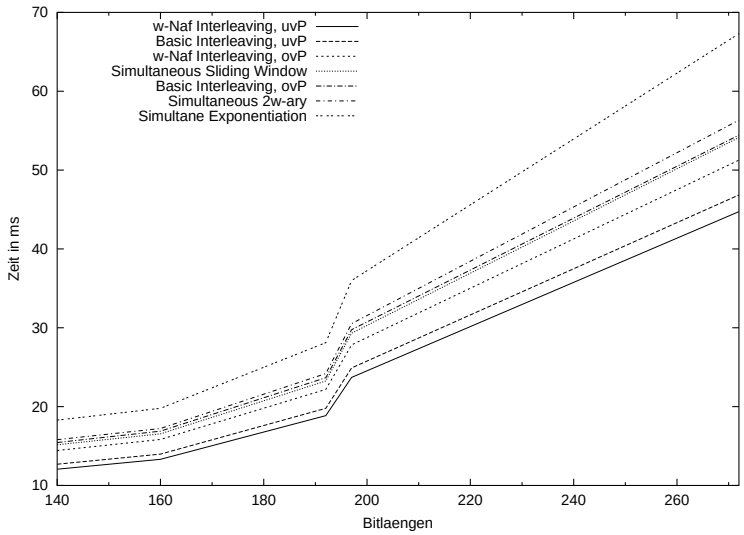


Abbildung 9.3: Vergleich der hochgerechneten Laufzeit von $k_1 \cdot P_1 + k_2 \cdot P_2$ ohne Verwendung vorberechneter Punkte

9.4.3 Vergleich der hochgerechneten Laufzeiten der Methoden für $k_1 \cdot P_1 + k_2 \cdot P_2$

Die jeweils besten, hochgerechneten Zeiten können nun wieder untereinander verglichen werden. In Abbildung 9.3 werden alle hochgerechneten Laufzeiten der Methoden für $k_1 \cdot P_1 + k_2 \cdot P_2$ verglichen. Es sind sieben verschiedene, da die Laufzeiten aus Abschnitt 9.3 sowie 9.4 abgebildet sind.

Die beste Methode, die Operation $k_1 \cdot P_1 + k_2 \cdot P_2$ zu berechnen, ist die w-NAF-based Interleaving Exponentiation. Dabei ist es egal, ob die Verwendung vorberechneter Punkte möglich ist, oder nicht.

9.5 Analyse der Laufzeituntersuchung

In diesem Abschnitt soll entschieden werden, welcher Algorithmus mit welchen Koordinaten für welche Anwendung die beste Wahl ist. Wir wiederholen die Ausführungen des Kapitels 6.3:

1. Für die Signaturerzeugung sind vor allem die Methoden für $s \cdot G$ geeig-

net, die vorberechnete Punkte verwenden. Sie sind für diese Anwendung die schnellsten und deshalb einsetzbar, weil der Basispunkt im Voraus bekannt ist: Fixed-size-sliding-window Exponentiation (ALG 6.1.2), Sliding window Exponentiation (ALG 6.1.3), Exponentiation mit Non-adjacent-forms (ALG 6.1.4), ± 1-Ketten (ALG 6.1.5) und LimLee Exponentiation (ALG 6.1.6).

2. Beim Schlüsselaustausch ist der Basispunkt nicht unbedingt im Voraus bekannt. Dann sind die Methoden für $s \cdot G$ die besseren, die keine oder nur wenig vorberechneten Punkte benötigen. Dies sind die Methoden Square-and-Multiply (ALG 6.1.1), Fixed-size-sliding-window Exponentiation (ALG 6.1.2), Sliding window Exponentiation (ALG 6.1.3) und die Exponentiation mit Non-adjacent-forms (ALG 6.1.4).

3. Die beste Wahl für die Signaturverifikation ist hängt davon ab, ob erstens der Basispunkt des Signierers im Voraus bekannt ist und ob zweitens ob die Vorberechnung von Punkten und deren Abspeichern sich lohnt, weil sie öfters verwendet werden. Ist dies der Fall, gibt es zwei grundsätzliche Möglichkeiten:

   (a) Die Signaturverifikation benötigt die Berechnung von $h_1 \cdot G + h_2 \cdot W$. Der erste Teil kann wie bei der Signaturerzeugung mit den Methoden Fixed-size-sliding-window Exponentiation (ALG 6.1.2), Sliding window Exponentiation (ALG 6.1.3), Exponentiation mit Non-adjacent-forms (ALG 6.1.4), ± 1-Ketten (ALG 6.1.5) und LimLee Exponentiation (ALG 6.1.6) berechnet werden. Sie sind vorzuziehen, da sie vorberechnete Punkte verwenden können.

   Der zweite Teil muss dagegen mit Methoden berechnet werden, die keine oder nur wenige vorberechnete Punkte benötigen. Dies sind die Methoden Square-and-Multiply (ALG 6.1.1), Fixed-size-sliding-window Exponentiation (ALG 6.1.2), Sliding window Exponentiation (ALG 6.1.3) und die Exponentiation mit Non-adjacent-forms (ALG 6.1.4). Anschließend müssen die Einzelergebnisse addiert werden.

   (b) Die andere Möglichkeit ist eine der Methoden zu verwenden, die $h_1 \cdot G$ und $h_2 \cdot W$ simultan berechnet. Es können von dieser Gruppe die Methoden verwendet werden, die vorberechnete Punkte verwenden. Das sind die Methoden Basic Interleaving Exponentiation (ALG 6.2.4) und w-NAF-based interleaving Exponentiation (ALG 6.2.5).

Andernfalls müssen Methoden verwendet werden, die keine oder wenig vorberechneten Punkte benötigen. Und auch dann gibt es die zwei verschiedenen Möglichkeiten:

(a) $h_1 \cdot G$ und $h_2 \cdot W$ werden getrennt berechnet und deren Ergebnisse dann addiert. Das sollte mit Methoden geschehen, die keine oder nur wenige vorberechnete Punkte benötigen: Square-and-Multiply (ALG 6.1.1), Fixed-size-sliding-window Exponentiation (ALG 6.1.2), Sliding window Exponentiation (ALG 6.1.3) und die Exponentiation mit Non-adjacent-forms (ALG 6.1.4).

(b) Oder es werden die Methoden verwendet, die $h_1 \cdot G + h_2 \cdot W$ in einem berechnet. Auch hier dürfen die Methoden keine oder nur wenige vorberechnete Punkte benötigen: Simultane Exponentiation (ALG 6.2.1), Simultane 2^w-ary Exponentiation (ALG 6.2.2), Simultane sliding window Exponentiation (ALG 6.2.3), Basic Interleaving Exponentiation Exponentiation (ALG 6.2.4) und w-NAF-based interleaving Exponentiation (ALG 6.2.5).

Innerhalb dieser Einteilung kann nun die beste Wahl für die einzelnen Anwendungen getroffen werden. Dafür muss nur die jeweils beste Zeit pro Bitlänge aus der zusammengestellten Tabelle herausgesucht werden.

Die Signaturerzeugung wird am besten mit der Methode LimLee ausgeführt, auch gut zu sehen in Abbildung 9.2 auf Seite 147. Die vorberechneten Punkte sind dabei affin. Durch die sehr kleine Anzahl an Punktadditionen und -verdopplungen können alle acobischen Kombinationen verwendet werden. Der zeitliche Unterschied bewegt sich dabei im Tausendstel-Millisekunden-Bereich. Da die Gesamtlaufzeit aber bei 140 Bit bei 1.6 ms und bei 272 Bit bei immerhin nur 6.54 ms beträgt, fällt dies nicht allzu schwer ins Gewicht. Die Parameter h und v sind für die verschiedenen Bitlängen unterschiedlich gewählt, um die Obergrenze des Speicherplatzes für die vorberechneten Punkte nicht zu überschreiten. Die beste Parameterwahl ist für 140 Bit $h = 10$ und $v = 2$, für 160 Bit $h = 7$ und $v = 10$, für 192 Bit $h = 6$ und $v = 16$, für 197 Bit $h = 8$ und $v = 4$ und für 272 Bit $h = 8$ und $v = 3$.

Der Schlüsselaustausch (Fall 2.) wird am besten mit der Exponentiation mit w-NAF-based ausgeführt. Bei allen fünf untersuchten Bitlängen wird die Vorberechnung in diesem Fall am Besten mit affinen Koordinaten ausgeführt. Die hintereinander ausgeführten Punktverdopplungen werden am besten mit Modifiziert Jacobischen Koordinaten ausgeführt, wobei die letzte vor eine Addition Jacobische oder Chudnowsky Jacobische Koordinaten zurückgibt. Die Addition gibt das

Ergebnis wieder in Modifiziert Jacobischen Koordinaten zurück.

Bei der Signaturverifikation ist die beste Methode nicht ganz so einfach den Tabellen zu entnehmen, da es -wie bereits besprochen - verschiedene Möglichkeiten gibt, die miteinander verglichen werden müssen. Darüber hinaus sind die beiden Fälle zu unterscheiden, in denen einmal die Verwendung vorberechneter Punkte möglich ist und einmal nicht.

Im ersten Fall gibt es die Möglichkeit die beste der Methoden zu verwenden, die die Operation $k_1 \cdot P_1 + k_2 \cdot P_2$ auf einmal lösen. Ein Vergleich der Tabellen ergibt, dass bei allen 4 Bitlängen die w-NAF-based Interleaving Exponentiation die beste Lösung biete. Dabei werden die Punkte in der Vorberechnungsphase Jacobisch abgespeichert, die einzige Punktverdopplung gibt dazu das Ergebnis affin zurück, die Operationen in der Hauptrechnung werden mit Modifiziert Jacobsichen Koordinaten ausgeführt. Dabei sind für die ersten vier untersuchten Bitlängen $n = 140, 160, 192$ und 197 die Fenstergrößen $w_1 = 11$ und $w_2 = 4$ und für $n = 272$ die Fenstergrößen $w_1 = 10$ und $w_2 = 5$ zu verwenden.

Die andere Möglichkeit ist, die beste Methode unter Verwendung vorberechneter Punkte mit der besten ohne Verwendung vorberechneter Punkte zu kombinieren und deren Ergebnis zu addieren. Das ist erstens die LimLee Methode und zweitens die Exponentiation mit NAFs. Laut den hochgerechneten Zeiten in den Tabellen kommen folgende Zeiten zustande:

Methode	140	160	192	197	272
LimLee uvP +	1.6	1.91	2.92	3.34	6.63
NAF ovP	11.64	12.99	17.78	21.82	40.9
=	13.24	14.9	20.7	25.16	47.53
w-NAF-based Interleaving uvP	12.45	13.91	19.16	23.67	45.14

Die w-NAF-based Interleaving Exponentiation ist hochgerechnet ein wenig schneller als die Kombination der anderen beiden Methoden und daher vorzuziehen. Das wird in Abbildung 9.4 illustriert.

Es bleibt zu untersuchen, welches die beste Wahl ist eine Signatur zu verifizieren, wenn keine vorberechneten Punkte verwendet werden können. Auch in diesem Fall schneidet die w-NAF-based Interleaving Exponentiation besser ab als die anderen Methoden, die die Operation $k_1 \cdot P_1 + k_2 \cdot P_2$ auf einmal berechnen. Die schnellste Methode zur einfachen Punktmultiplikation ohne Verwendung vorberechneter Punkt ist wieder die Exponentiation mit NAFs. Zu erwarten ist, dass die w-NAF-based Interleaving Exponentiation schneller ist, als letztere Methode zweimal angewandt, da die verschachtelte Methode nur halb soviele Punktverdopplun-

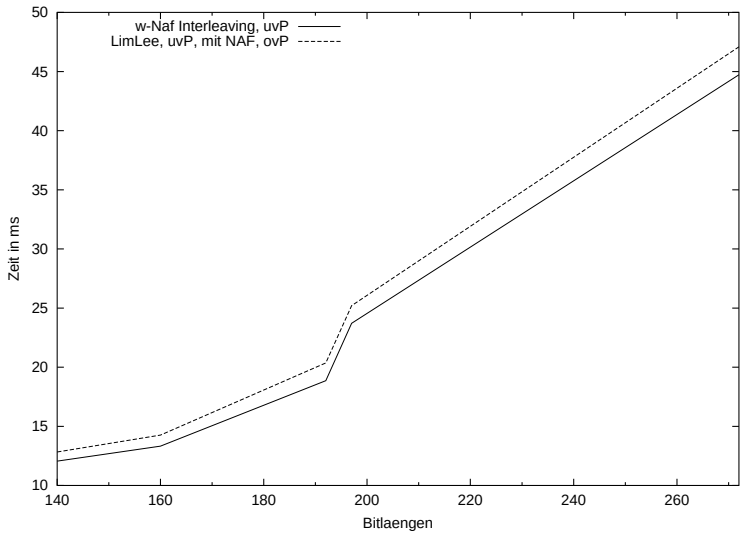


Abbildung 9.4: Vergleich der hochgerechneten Laufzeiten von LimLee, uvP, mit Naf, ovP, und w-Naf-based Interleaving, uvP

gen benötigt. Dies bestätigt sich im direkten Vergleich:

Methode	140	160	192	197	272
NAF ovP	11.64	12.99	17.78	21.82	40.9
2· NAF ovP	23.28	25.98	35.56	43.64	81.8
w-NAF-based Interleaving ovP	14.92	16.55	22.59	27.83	51.8

Auch in diesem Fall ist also die w-NAF-based Interleaving Exponentiation zu verwenden, um das beste Ergebnis zu erzielen. Die große Differenz der zu erwartenden Laufzeiten ist gut in Abbildung 9.5 zu erkennen.

9.6 Aussagekraft der hochgerechnete Laufzeiten

Die tatsächliche Laufzeit einer Methode zur Punktmultiplikation ist stark abhängig von der Anzahl der Einsen (Hamminggewicht) und deren Verteilung in der binären Repräsentation des Exponenten k, da sie die Anzahl der Punktadditionen und die Anzahl der Übergänge von Punktadditionen zu Verdopplungen bestimmen.

Für die Hochrechnung der Laufzeit wurde für diese Anzahl jeweils der Erwartungswert verwendet, da man in den seltensten Fällen den Exponenten im Voraus

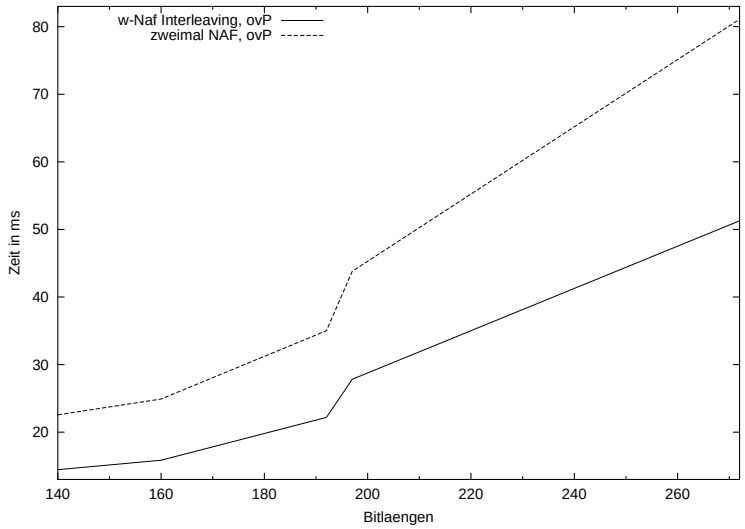


Abbildung 9.5: Vergleich der doppelten hochgerechneten Laufzeiten von NAF, ovP, und w-Naf-based Interleaving Exponentiation, ovP

kennt und der Erwartungswert zumindest in den meisten Fällen angenähert wird. So liegt zum Beispiel der Erwartungswert für das Hamminggewicht einer zufälligen n-Bit langen Zahl bei $n/2$. Der Erwartungswert der jeweiligen Anzahl der Punktadditionen je Algorithmus wurde den zitierten Arbeiten entnommen.

In der Praxis treten Abweichungen vom Erwartungswert auf. Das führt dazu, dass die Laufzeiten, die experimentell gemessen werden, entweder unterhalb oder oberhalb der hochgerechneten Zeiten liegen.

Zu beachten ist zusätzlich, dass die Zeit, die für den Aufruf der verschieden Punktoperationsmethoden benötigt wird, nicht in die Hochrechnung einfließen. Zusätzlich sind wegen der geforderten Erweiterbarkeit die Methoden der Punktaddition und Verdopplung (`PointGFP` oder `PointMixed`) in einer anderen Klasse liegen als die Methoden zur Punktmultiplikation (`Point`). Es wäre interessant zu untersuchen, inwiefern die Integrierung ein Verzicht dieses Features Effizienzvorteile bringen könnte. So ging auch die Zeit für Objekt-Castings nicht in die Berechnung ein. Es sind daher durchgehend leicht längere Zeiten zu erwarten. Die wird sich im nächsten Kapitel bestätigen.

Doch auch für spezielle Exponenten können die vorgestellte Strategie und die Formeln aus diesem Kapitel dazu verwendet werden, die optimale Ausprägung zu finden: Dabei wird der betreffende Exponent darauf untersucht, wieviele Punkt-

additionen benötigt werden, wieviele Verdopplungen, und wieviele Wechsel zwischen Verdopplungen und Additionen benötigt werden. Dies hängt natürlich vom gewählten Algorithmus ab. Diese Werte müssen in die Formeln an den Stellen eingesetzt werden, an denen momentan der Erwartungwert steht.

Beispiel 11: Berechnung der optimalen Ausprägung für einen speziellen Exponenten

Es soll $379351 \cdot P$ mit Sliding Window, Fenstergröße $w = 3$ berechnet werden. Die Punkte $P_i = (2 \cdot i + 1) \cdot P$, $i = 0 \ldots 3$, seien bereits vorberechnet. Die Binärdarstellung des Exponenten ist 1011100100111010111_2. Durch die Fenstergröße $w = 3$ ergibt sich die Einteilung $101|11|00|1|00|101|0|101|0|1$ und somit der Ablauf

$101 \rightarrow$	ADD
$11 \rightarrow$	DBL DBL ADD
$00\|1 \rightarrow$	DBL DBL DBL ADD
$00\|111 \rightarrow$	DBL DBL DBL DBL DBL ADD
$0\|101 \rightarrow$	DBL DBL DBL DBL ADD
$0\|1 \rightarrow$	DBL DBL ADD.

Die erste Addition ist von Typ $R^E \leftarrow P^{\mathcal{A}}$. Die restlichen 5 Additionen sind vom Typ $R^E \leftarrow R^D + P_-^{\mathcal{A}}$ (siehe die Formelsammlung auf Seite 138). Ebenfalls werden 5 Verdopplungen vom Typ $R^D \leftarrow 2 \cdot R^C$ und 5 vom Typ $R^C \leftarrow 2 \cdot R^E$ benötigt. Von den 16 Verdopplungen $(n - w)$ insgesamt werden so noch 6 vom Typ $R^C \leftarrow 2 \cdot R^C$ ausgeführt. Die Berechnung endet mit einer Normierung des Punktes R mit $R^{\mathcal{A}} \leftarrow R^E$. Aus der Abschätzung

$$
\begin{array}{lcll}
1 & \times & R^E \leftarrow P_-^{\mathcal{A}} & + \\
(\frac{n}{w+1} - 1) & \times & R^E \leftarrow R^D + P_-^{\mathcal{A}} & + \\
(t - 2 \cdot (\frac{n}{w+1} - 1)) & \times & R^C \leftarrow 2 \cdot R^C & + \\
(\frac{n}{w+1} - 1) & \times & R^D \leftarrow 2 \cdot R^C & + \\
(\frac{n}{w+1} - 1) & \times & R^C \leftarrow 2 \cdot R^E & + \\
1/2 & \times & R^{\mathcal{A}} \leftarrow R^E & + \\
1/2 & \times & R^{\mathcal{A}} \leftarrow R^C. &
\end{array}
$$

mit den Werten

$$
\begin{array}{lcll}
1 & \times & R^E \leftarrow P_-^{\mathcal{A}} & + \\
(\frac{19}{4} - 1) = 3.75 & \times & R^E \leftarrow R^D + P_-^{\mathcal{A}} & + \\
(16 - 2 \cdot 3.75) = 8.5 & \times & R^C \leftarrow 2 \cdot R^C & + \\
3.75 & \times & R^D \leftarrow 2 \cdot R^C & + \\
3.75 & \times & R^C \leftarrow 2 \cdot R^E & + \\
1/2 & \times & R^{\mathcal{A}} \leftarrow R^E & + \\
1/2 & \times & R^{\mathcal{A}} \leftarrow R^C. &
\end{array}
$$

wird so

$$
\begin{array}{lcll}
1 & \times & R^E \leftarrow P_-^{\mathcal{A}} & + \\
5 & \times & R^E \leftarrow R^D + P_-^{\mathcal{A}} & + \\
((19 - 3) - 2 \cdot 5) = 6 & \times & R^C \leftarrow 2 \cdot R^C & + \\
5 & \times & R^D \leftarrow 2 \cdot R^C & + \\
5 & \times & R^C \leftarrow 2 \cdot R^E & + \\
1 & \times & R^{\mathcal{A}} \leftarrow R^E & + \\
0 & \times & R^{\mathcal{A}} \leftarrow R^C. &
\end{array}
$$

Wir haben keine Laufzeituntersuchung der Körperoperationen über einem Körper $\mathbb{F}_p$ mit $\log_2 p = 19$, daher ist es in diesem Beispiel nicht möglich, die beste Ausprägung zu berechnen.

Legt man die gemessenen Laufzeiten für die Operationen für 140 Bit zu Grunde (keine gute Abschätzung, zugegeben), sind auch hier die Ausprägungen die besten, die für aufeinander folgende Verdopplungen Modifiziert Jacobische Koordinaten und für die Additionen Jacobische oder Chudnowsky Jacobische Koordinaten verwenden.

Kapitel 10

Experimentelle Ergebnisse

In diesem Kapitel wird die Brücke zur Praxis geschlagen: Die berechneten Ausprägungen der einzelnen Algorithmen wurden mit Hilfe des FlexiProviders getestet, die Ergebnisse in Tabellen zusammengefasst.

Eine Zusammenfassung der wichtigsten Ergebnisse der experimentellen Tests und der hochgerechneten Komplexitäten auf der gewählten Plattform schließt den folgenden Abschnitt ab. Darin wird gezeigt, dass die beste Methode zur einfachen Punktmultiplikation ohne Verwendung vorberechneter Punkte die Exponentiation mit Non-adjacent-forms ist, mit einer der besten acht Ausprägungen aus Kapitel 9. Keine der anderen Methoden kann dagegen die Methode LimLee schlagen, wenn die Verwendung vorberechneter Punkte möglich ist. Dabei werden ebenfalls die berechneten Ausprägungen des letzten Kapitels bestätigt.

Bei der mehrfachen Punktmultiplikation ohne Verwendung vorberechneter Punkte weichen die Ergebnisse der experimentellen Tests von denen der Hochrechnung ab: Während letztere die Methode w-Naf-based Interleaving Exponentiation empfehlen, zeigt dieses Kapitel, dass die Simultaneous Sliding Window Methode die bessere Wahl ist. Der Erwartungswert der Anzahl der Punktaddition weicht bei der Interleaving Methode stark von den tatsächlichen Werten ab.

Ist die Verwendung vorberechneter Punkte dagegen möglich, gewinnt auch in diesem Kapitel die w-Naf-based Interleaving Exponentiation. Alle Tests bestätigen im Großen und Ganzen die berechneten Ausprägungen des vorangegangenen Kapitels.

Die Zeiten, die bis zur Fertigstellung dieser Arbeit mit Hilfe des Kryptoprozessors gemessen wurden, schließen dieses Kapitels ab (Seite 213 ff). Darin

Tabellenübersicht der experimentellen Ergebnissen	
Algorithmus	Tabellennummer
Square-And-Multiply,ovP	Tab. 10.2
Fixed-size-sliding-window	Tab. 10.3- 10.4
Sliding-window Exponentiation,ovP	Tab. 10.5 - 10.9
Exponentiation mit Non-adjacent-forms,ovP	Tab. 10.10 - 10.12
Fixed-size-sliding-window, uvP	Tab. 10.13 - 10.15
Sliding-window Exponentiation, uvP	Tab. 10.16
Exponentiation mit Non-adjacent-forms, uvP	Tab. 10.17
± 1-Additionsketten	Tab. 6.1.5
LimLee	Tab. 10.19 - 10.20
Simultane Exponentiation, ovP	Tab. 10.21
Simultane 2^w-ary Exponentiation, ovP	Tab. 10.22 - 10.23
Simultane sliding window Exponentiation, ovP	Tab. 10.24 - 10.25
Basic interleaving Exponentiation, ovP	Tab. 10.26 - 10.30
wNAF-based interleaving Exponentiation, ovP	Tab. 10.31 - 10.35
Basic interleaving Exponentiation, uvP	Tab. 10.36 - 10.40
wNAF-based interleaving Exponentiation, uvP	Tab. 10.41 - 10.45

Tabelle 10.1: Tabellenübersicht der experimentellen Ergebnissen

wird gezeigt, dass auch ohne Verwendung vorberechneter Punkte kurze Laufzeiten möglich sind.

10.1 Laufzeitmessungen der Software-Implementierung

Eine Übersicht, in der die Zeiten für welchen Algorithmus zu finden sind, erhält man in die Tabelle 10.1. Die Experimente zur Softwareimplementierung wurden auf der selben Plattform ausgeführt, auf der auch die Laufzeiten der Körperberechnungen gemessen wurden[1]. Dabei wurden je Tausend Punktmultiplikationen mit verschiedenen, zufällig gewählten Exponenten ausgeführt. Für die Tests wurden für jede der fünf Bitgrößen, 140, 160, 192, 197 und 272, ein Satz EC Parameter mit der Bibliothek LiDIA [LiDIA98] erzeugt, deren Körper, Kurve und Basispunkt verwendet werden. Die Parameter sind im Anhang, Seite I ff, aufgelistet.

Wir werten die experimentellen Ergebnisse in ihrer Gesamtheit aus und gehen

[1] AMD Athlon mit 1.3 GHz und 256 MB, Linux, Java Version „1.4.1_02“

nur auf einzelne spezielle Fälle ein:

Beim direkten Vergleich der Zeiten der Tabellen dieses Kapitels mit den hochgerechneten Zeiten des Kapitels 9 fällt als erstes auf, dass viele experimentellen Zeiten höher liegen als die hochgerechneten. Das tritt hauptsächlich bei den Methoden auf, die eine große Komplexität haben, wie um Beispiel Square-and-Multiply. Dies wurde bereits im vorangegangen Kapitel angesprochen: In die hochgerechneten Zeiten wurde nicht die Zeit mit eingerechnet, die benötigt wird, Methoden aufzurufen, Castings durchzuführen und auf Array-Felder zuzugreifen. Diese Zeiten fließen nun in die Ergebnisse der experimentellen Tests ein. Da sie aber unabhängig vom Algorithmus sind, werden sie in dieser Arbeit nicht näher untersucht.

Bei einer Aufteilung der Algorithmen wie im Kapitel 9 kann beim Vergleich der Zeiten der einzelnen Algorithmen beobachtet werden, dass von Algorithmus zu Algorithmus eine Verbesserung stattfindet. Dies deckt sich mit den Komplexitäten der einzelnen Methoden (siehe Tabellen 6.1 und 6.2 auf den Seiten 77 und 84), sowie mit der tatsächlich benötigten Anzahl an Additionen bei diesem Experiment:

Die Exponenten wurden zufällig gewählt. Eine Untersuchung ergibt, dass bei den 1000 verschiedenen Exponenten die Anzahl der gesetzten Einsen ($\#1$) in der jeweiligen Binärdarstellung knapp größer ist als $n/2$:

$$\begin{array}{llll} n = 140 & \rightarrow & 140/2 = 70 & \#1\text{: } 70.763 \\ n = 160 & \rightarrow & 160/2 = 80 & \#1\text{: } 80.515 \\ n = 192 & \rightarrow & 192/2 = 96 & \#1\text{: } 97.407 \\ n = 197 & \rightarrow & 197/2 = 98.5 & \#1\text{: } 98.407 \\ n = 272 & \rightarrow & 272/2 = 136 & \#1\text{: } 136.275 \end{array}$$

Die jeweilige Anzahl der Einsen liegt also tatsächlich ganz dicht am entsprechenden Erwartungswert. Die Ergebnisse der Tests der Methode Square-and-Multiply spiegeln daher die hochgerechneten Zeiten ziemlich genau wieder (vergleiche die Tabellen 10.2 und 9.1). Die Anzahl der Einsen entspricht in dieser Methode der Anzahl der benötigten Punktaddtionen.

Doch schon bei der nächsten Methode kann man beobachten, dass dies nicht immer der Fall ist: Bei der Bitlänge $n = 140$ und $w = 4$ ist der Erwartungswert der Anzahl der benötigten Additionen

$$2^{4-1} + \frac{140 \cdot (2^4 - 1)}{4 \cdot 2^4} = 40.8125.$$

Tatsächlich sind es in diesem Experiment mit den 1000 zufällig gewählten Exponenten aber im Schnitt 45.884 Additionen. In Prozent ausgedrückt werden über

12% mehr Additionen benötigt als der Erwartungswert vermuten lässt. Diese zusätzlichen fünf Additionen ziehen sich auch durch die vier größeren Bitlängen. Doch das Verhältnis ändert sich: Schon bei 192 Bit handelt es sich nur noch um 9.7%, bei 272 Bit nur noch um 6% Abweichung vom Erwartungswert.

Auch bei der Sliding Window Methode gibt es Abweichungen, jedoch viel geringere: Bei $n = 140$ mit $w = 4$ sind 36 Additionen zu erwarten, tatsächlich werden aber nur 35.238 im Schnitt ausgeführt. Das macht deutlich, dass die hochgerechneten Zeiten nicht direkt mit denen der Experimente verglichen werden dürfen.

Die schnellste Methode unter denen, die $k \cdot P$ ohne Verwendung vorberechneter Punkte berechnen, ist auch bei den experimentellen Tests die Exponentiation mit Non-adjacent-forms (Tabellen 10.10 bis 10.10). Die Schwankungen, denen die experimentellen Ergebnisse unterliegen sind gering. So sind in den meisten Fällen sogar die feinen Unterschiede zwischen den Ausprägungen zu erkennen. Auch dass die Verwendung von ausschließlich einer Koordinatenart nicht die beste Wahl ist, wird deutlich.

Die Ergebnisse der Experimente mit den Methoden, die vorberechnete Punkte verwenden (Tabellen 10.13 bis 10.20), sagen deutlich aus, dass das LimLee-Verfahren das mit Abstand Schnellste ist - vorausgesetzt, jeder Methode steht der gleiche Speicherplatz zur Verfügung. Mit unter 2 ms bei 140 Bit und unter 8 ms bei 272 Bit kommt keine andere Methode an diese Zeiten heran. Auffällig ist hier, dass die erste der besten acht Ausprägungen ($\mathcal{J}^{\mathcal{C}}\,\mathcal{J}^{\mathcal{C}}\,\mathcal{J}^{\mathcal{M}}$) bei 140 und 192 Bit eine deutlich höhere Laufzeit besitzt als die übrigen sieben. Dies muss damit zusammenhängen, dass sie Chudnowsky Jacobische Koordinaten zur Addition verwenden, diese im Verhältnis zur Verdopplung jedoch nur wenig benötigt werden. So kostet die Berechnung von Chudnowsky Jacobischen Koordinaten in diesen Fällen mehr, als deren Verwendung einsparen.

Ist die Anforderung nicht so hoch und muss nur die menschliche Wahrnehmung befriedigt werden, genügen auch bereits die anderen Algorithmen. Selbst für 272 Bit liegen die besten Werte weit unterhalb der geforderten 100 Millisekunden.

Es bleiben die Algorithmen zur mehrfachen Punktberechnung, deren Laufzeiten in den Tabellen 10.21 bis 10.45 festgehalten sind. Es fällt sofort auf, dass die Zeiten nicht von Methode zu Methode besser werden. Aus diesem Grund untersuchen wir, wieviele Punktadditionen bei den Tests tatsächlich pro Punktmultiplikation im Schnitt benötigt wurden. Folgende Tabelle vergleicht die Anzahl der zu erwartenden Additionen nach Tabelle 6.2 mit der tatsächlichen anhand der der Bitlänge $n = 140$:

	erwartete #ADD	tatsächliche #ADD
Sim. Expo.	105	103,28
Sim. 2^w-ary	75.625	77.814
Sim. Slid.	70	69.146
Basic Interl., ovP	72	70.884
w-Naf Interl., ovP	62,667	61.204
Basic Interl., uvP	39.667	47.532
w-Naf Interl., uvP	34.1	41.733

Die mittlere Spalte bestätigt zunächst die experimentellen Ergebnisse insofern, als unter den ersten drei Methoden die Laufzeit leicht abnimmt und sie bei Basic Interleaving wieder etwas ansteigen sollte, um zur nächsten Methode wieder abzufallen.

Die rechte Spalte zeigt, dass das Abfallen der Laufzeit von der ersten zur zweiten Methode schwächer ausfallen muss, als durch die mittlere Spalte vorgegeben. Dies wird auch von den experimentellen Ergebnissen bestätigt. Allgemein weicht die tatsächliche Anzahl der Punktadditionen von dem Erwartungswert nur leicht ab, wenn keine vorberechneten Punkte verwendet werden. Doch in den letzten zwei Zeilen ist zu sehen, dass das nicht für die beiden Methoden gilt, die vorberechnete Punkte verwenden. Ihre tatsächliche Komplexität liegt um knappe acht Additionen höher als erwartet. Dies schlägt sich selbstverständlich in der Laufzeit nieder.

Ein weiteres Phänomen ist, dass die beste Laufzeit der Simultanen Sliding Window Methode besser ist als die der w-Naf-based Interleaving Exponentiation. Vergleicht man jedoch diese Zeiten mit den hochgerechneten aus dem vorherigen Kapitel, sieht man, dass die experimentelle Laufzeit der simultanen Methode leicht besser ist als die hochgerechnete (hochgerechnet: 15.67 ms, experimentell: 14.31), während es bei der Interleaving Methode genau umgekehrt ist (hochgerechnet: 14.92, experimentell: 15.68). So lässt sich auch dieses Phänomen erklären.

10.1.1 Zusammenfassung

Dieser Abschnitt hat gezeigt, dass die hochgerechneten Komplexitäten im Großen und Ganzen durch experimentelle Tests bestätigt werden. Es gibt kleine Abweichungen in den Laufzeiten der einzelnen Methoden, die sich durch eine Untersuchung der Exponenten erklären lassen.

So haben die Zeiten der experimentellen Tests und die hochgerechneten Komplexitäten für diese Plattform folgendes ergeben:

- Für einfache Punktmultiplikationen sollte die Exponentiation mit Non-adjacent-forms eingesetzt werden, wenn keine vorberechneten Punkte zur Verfügung stehen. Sie bietet in dieser Algorithmen-Gruppe die kürzeste Komplexität und auch die kleinste Laufzeit. Dabei ist zu vermeiden, ausschließlich eines der fünf Koordinatensysteme einzusetzten, da diese um mindestens eine Millisekunde vom besten Wert abweichen. Die Vorberechnung soll mit Affinen Koordinaten berechnet werden, die Hauptrechnung mit Modifiziert Jacobischen und Jacobischen oder Chudnowsky Jacobischen Koordinaten. Dabei sind letztere für Hintereinanderausführungen von Punktadditionen einzusetzten, Modifiziert Jacobische Koordinaten zu Hintereinanderausführungen von Punktverdopplungen.

- Wenn vorberechnete Punkte zur einfachen Punktmultiplikation verwendet werden können, soll die Methode von Lim und Lee verwendet werden. Sie ist mit Abstand die beste Wahl. Dabei sind die Parameter h und v - wie im letzten Kapitel berechnet - zu verwenden. Da die Anzahl der benötigten Punktoperationen bei dieser Methode so klein ist, unterscheiden sich die besten acht Ausprägungen nur wenig voneinander. Insbesondere können die Punktoperationen alle mit Jacobischen Koordinaten ausgeführt werden. Ausschließlich Projektive, Modifiziert Jacobische oder Affine Koordinaten sollten vermieden werden.

- Eine Aussage über die mehrfache Punktmultiplikationen ohne Verwendung vorberechneter Punkte lässt sich nicht ganz so eindeutig treffen: Die Hochrechnung empfiehlt die Verwendung der w-Naf-based Interleaving Exponentiation, die eine leicht bessere Komplexität besitzt als die Simultaneous Sliding Window Exponentiation. Doch bei den Experimenten konnte dies nicht bestätigt werden. Das lag an den unterschiedlichen Abweichungen vom Erwartungswert der Anzahl der Punktadditionen. Dieser Fall zeigt, dass der Erwartungswert gerade bei dieser Art von Punktmultiplikation kein absolut verlässliches Maß ist. Die Hochrechnungen aus Kapitel 9 empfehlen so die Verwendung der w-Naf-based Interleaving Exponentiation, die experimentellen Tests dagegen die Simultaneous Sliding Window Exponentiation. Für jede der beiden Methoden sind die berechneten besten Ausprägungen jedoch im Großen und Ganzen bestätigt worden.

- Trotz der beschriebenen Abweichungen der Anzahl der Punktadditionen vom Erwartungswert, wird die w-Naf-based Interleaving Exponentiation unter Verwendung vorberechneter Punkte mitsamt der ersten acht Ausprägungen von den experimentellen Ergebnissen bestätigt.

Square-and-Multiply ovP, alle Bitlängen							
Koordinaten			exp. ms				
A	B	C	140	160	192	197	272
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	19.92	20.94	27.21	32.66	53.29
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	20.49	20.77	27.19	33.04	54.37
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	21.45	21.7	28.27	34.97	57.27
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	22.09	22.82	30.6	36.7	63.01
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	23.75	28.3	38.76	42.44	79.87
$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	19.82	20.32	26.8	31.62	51.86
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	19.94	20.11	26.29	31.28	51.89
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	19.39	20.15	26.32	31.61	52.33
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	19.49	20.46	26.42	31.09	51.7
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	20.28	20.85	27.51	32.52	53.47
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	20.39	20.86	26.88	32.17	53.3
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	20.14	20.85	27.07	32.47	53.23
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	21.04	21.01	27.41	32.35	53.38

Tabelle 10.2: Experimentelle Laufzeit Square-and-Multiply (ALG 6.1.1) ohne Verwendung vorberechneter Punkte, alle Bitlängen, in ms

Fixed-size-sliding-window ovP, 140 Bit							
Koordinaten					Art	w	exp. ms
A	B	C	D	E			
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	1	4	14.39
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	1	4	15.37
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	2	4	15.98
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	2	4	18.31
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2	4	17.35
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	4	13.29
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	4	13.21
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	4	13.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	4	13.22
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	1	4	13.43
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	1	4	13.45
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	1	4	13.38
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	1	4	13.39

Tabelle 10.3: Experimentelle Laufzeit Fixed-size-sliding-window (ALG 6.1.2) ohne Verwendung vorberechneter Punkte, 140 Bits, in ms

Fixed-size-sliding-window ovP, 160 - 272 Bit										
Koordinaten					Art	w	exp. ms			
A	B	C	D	E			160	192	197	272
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	1	4	15.5	21.77	27.55	49.71
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	1	4	16.44	23.1	29.53	53.82
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	2	4	16.97	23.9	30.73	56.19
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2	4	18.62	26.18	33.67	61.51
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	2	4	22.59	32.83	38.05	74.49
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	4	14.37	20.53	25.53	45.9
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	4	14.55	20.62	25.5	45.83
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	4	14.51	20.55	25.43	45.86
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	4	14.46	20.55	25.5	46.05
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	1	4	14.49	20.47	25.71	45.77
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	1	4	14.45	20.4	25.66	45.87
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	1	4	14.41	20.48	25.65	45.98
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	1	4	14.53	20.45	25.58	45.88

Tabelle 10.4: Experimentelle Laufzeit Fixed-size-sliding-window (ALG 6.1.2) ohne Verwendung vorberechneter Punkte, 160 - 272 Bits, in ms

Sliding-window Exponentiation ovP, 140 Bit						
Koordinaten					w	exp. ms
A	B	C	D	E		
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	13.53
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	14.56
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	15.13
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	16.68
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	17.26
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	12.41
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	12.36
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	12.42
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	12.49
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	4	12.67
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	12.64
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	12.6
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	4	12.51

Tabelle 10.5: Experimentelle Laufzeit Sliding-window Exponentiation (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 140 Bits, in ms

Sliding-window Exponentiation ovP, 160 Bit						
Koordinaten					w	exp.
A	B	C	D	E		ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	14.57
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	15.53
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	16.09
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	17.69
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	21.57
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	13.51
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	13.52
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	13.47
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	13.5
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	4	13.53
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	13.6
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	13.83
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	13.91

Tabelle 10.6: Experimentelle Laufzeit Sliding-window Exponentiation (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 160 Bits, in ms

Sliding-window Exponentiation ovP, 192 Bit						
Koordinaten					w	exp.
A	B	C	D	E		ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	20.5
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	21.79
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	22.62
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	24.93
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	31.5
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	18.96
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	18.93
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	19.02
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	18.96
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	18.9
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	19.23
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	4	19.27
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	18.94

Tabelle 10.7: Experimentelle Laufzeit Sliding-window Exponentiation (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 192 Bits, in ms

Sliding-window Exponentiation ovP, 197 Bit						
Koordinaten					w	exp.
A	B	C	D	E		ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	25.69
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	27.66
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	5	29.23
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	5	31.88
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	5	35.71
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	23.85
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	5	23.85
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	5	23.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	5	23.86
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	23.79
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	23.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	23.76
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	23.75

Tabelle 10.8: Experimentelle Laufzeit Sliding-window Exponentiation (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 197 Bits, in ms

Sliding-window Exponentiation ovP, 272 Bit						
Koordinaten					w	exp.
A	B	C	D	E		ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	47.0
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	51.25
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	5	53.49
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	5	58.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	5	70.24
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	43.11
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	5	43.05
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	5	43.03
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	5	43.06
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	43.21
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	43.24
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	43.33
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	43.27

Tabelle 10.9: Experimentelle Laufzeit Sliding-window Exponentiation (ALG 6.1.3) ohne Verwendung vorberechneter Punkte, 272 Bits, in ms

Exponentiation mit Non-adjacent-forms ovP, 140, 160 und 192 Bit								
Koordinaten					w	exp. ms		
A	B	C	D	E		140	160	192
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	13.2	14.18	19.88
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	14.24	15.19	21.46
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	14.9	15.92	22.33
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	17.6	18.59	26.22
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	17.85	21.93	32.08
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	12.22	13.25	18.57
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	12.23	13.26	18.55
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	12.22	13.31	18.51
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	12.3	13.35	18.62
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	3	12.32	13.29	18.7
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	3	12.3	13.27	18.68
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	3	12.32	13.33	18.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	3	12.28	13.21	18.77

Tabelle 10.10: Experimentelle Laufzeit Exponentiation mit Non-adjacent-forms (ALG 6.1.4) ohne Verwendung vorberechneter Punkte, 140, 160 und 192 Bits, in ms

Exponentiation mit Non-adjacent-forms ovP, 197 Bit						
Koordinaten					w	exp.
A	B	C	D	E		ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	24.88
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	27.18
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	28.24
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	33.25
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	36.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	23.26
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	23.23
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	23.26
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	23.23
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	4	23.5
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	23.53
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	23.46
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	4	23.56

Tabelle 10.11: Experimentelle Laufzeit Exponentiation mit Non-adjacent-forms (ALG 6.1.4) ohne Verwendung vorberechneter Punkte, 197 Bits, in ms

Exponentiation mit Non-adjacent-forms ovP, 272 Bit						
Koordinaten					w	exp.
A	B	C	D	E		ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	45.15
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	49.89
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	52.37
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	61.55
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	71.54
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	41.99
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	4	41.97
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	4	42.08
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	4	42.13
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	42.47
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	5	42.4
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	5	42.49
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	5	42.58

Tabelle 10.12: Experimentelle Laufzeit Exponentiation mit Non-adjacent-forms (ALG 6.1.4) ohne Verwendung vorberechneter Punkte, 272 Bits, in ms

Fixed-size-sliding-window uvP, 140, 160 und 197 Bit						
Koordinaten			w	exp. ms		
C	D	E		140	160	192
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	9.87	10.89	19.46
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	11.21	12.16	22.22
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	11.97	13.12	23.98
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	10	13.23	14.46	26.63
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	10	14.55	18.05	31.2
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	9.75	10.74	19.24
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	10	9.78	10.75	19.33
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	10	9.76	10.73	19.3
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	10	9.8	10.81	19.33
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	10	9.85	10.83	19.45
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	9.8	10.83	19.48
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	9.89	10.82	19.46
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	10	9.8	10.85	19.48

Tabelle 10.13: Experimentelle Laufzeit Fixed-size-sliding-window (ALG 6.1.2) unter Verwendung vorberechneter Punkte, 140, 160 und 197 Bits, in ms

Fixed-size-sliding-window uvP, 192 Bit				
Koordinaten			w	exp.
C	D	E		ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	16.15
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	17.95
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	8	19.26
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	8	21.3
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	8	27.95
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	15.88
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	10	15.97
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	10	15.99
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	10	15.93
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	8	15.91
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	15.94
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	15.93
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	15.89

Tabelle 10.14: Experimentelle Laufzeit Fixed-size-sliding-window (ALG 6.1.2) unter Verwendung vorberechneter Punkte, 192 Bits, in ms

Fixed-size-sliding-window uvP, 272 Bit				
Koordinaten			w	exp.
C	D	E		ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	9	37.56
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	42.56
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	8	45.91
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	8	50.96
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	8	64.34
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	36.92
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	8	36.89
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	36.89
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	36.87
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	9	37.08
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	9	37.13
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	9	37.09
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	9	37.07

Tabelle 10.15: Experimentelle Laufzeit Fixed-size-sliding-window (ALG 6.1.2) unter Verwendung vorberechneter Punkte, 272 Bits, in ms

Sliding-window Exponentiation uvP, alle Bitlängen								
Koordinaten			w	exp. ms				
C	D	E		140	160	192	197	272
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	9.58	10.63	15.14	18.86	35.59 $w = 10$
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	11.03	11.97	17.03	21.48	41.04 $w = 10$
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	11.9	12.92	18.5	23.31	44.42 $w = 10$
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	13.24	14.27	20.52	25.97	49.38 $w = 10$
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	14.31	17.8	26.8	30.04	61.68 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	9.6	10.53	14.99	18.72	35.3 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	11	9.59	10.59	15.07	18.79	35.25 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	11	9.59	10.63	15.06	18.62	35.33 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	11	9.62	10.7	15.04	18.84	35.24 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	11	9.65	10.69	15.12	18.85	35.57 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	9.65	10.62	15.13	18.87	35.55 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	9.64	10.65	15.14	18.85	35.58 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	11	9.56	10.71	15.15	18.85	35.54 $w = 10$

Tabelle 10.16: Experimentelle Laufzeit Sliding-window Exponentiation (ALG 6.1.3) unter Verwendung vorberechneter Punkte, alle Bitlängen, in ms

Exponentiation mit Non-adjacent-forms uvP, alle Bitlängen								
Koordinaten			w	exp. ms				
C	D	E		140	160	192	197	272
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	10.09	11.12	15.55	19.24	35.9 $w = 10$
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	11.6	12.52	17.63	22.2	41.68 $w = 10$
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	12.5	13.39	18.92	23.93	45.06 $w = 10$
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	13.67	14.97	21.15	26.58	50.27 $w = 10$
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	15.09	18.55	27.84	31.28	64.21 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	9.94	11.02	15.43	19.14	35.8 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	11	9.45	10.96	15.46	19.18	35.85 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	11	9.98	11.03	15.48	19.15	35.66 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	11	9.98	10.98	15.45	19.22	35.72 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	11	10.06	11.11	15.64	19.33	36.04 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	10.0	11.07	15.62	19.3	36.09 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	10.01	11.03	15.66	19.33	36.02 $w = 10$
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	11	10.02	11.08	15.62	19.4	36.09$w = 10$

Tabelle 10.17: Experimentelle Laufzeit Exponentiation mit Non-adjacent-forms (ALG 6.1.4) unter Verwendung vorberechneter Punkte, alle Bitlängen, in ms

±1-Additionsketten uvP, 140 Bit					
Koordinaten	exp. ms				
C	140	160	192	197	272
$\mathcal{J}^{\mathcal{C}}$	4.71	4.88	6.89	8.92	16.21
$\mathcal{J}$	4.78	5.0	6.93	8.9	16.11
$\mathcal{P}$	5.23	5.35	7.48	9.62	17.61
$\mathcal{J}^{\mathcal{M}}$	5.81	5.97	8.45	10.89	20.06
$\mathcal{A}$	5.61	6.71	9.97	11.44	22.4

Tabelle 10.18: Experimentelle Laufzeit ±1-Additionsketten (ALG 6.1.5) unter Verwendung vorberechneter Punkte, 140 Bits, in ms

LimLee uvP, 140, 160, 192 und 272 Bit												
Koordinaten			140, h, v		160, h, v		192, h, v		272, h, v			
C	D	E	8	5	7	8	6	16	7	5		
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	1.85		2.19		3.26		7.41			
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	1.83		2.23		3.29		7.38			
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2.04		2.43		3.43		8.01			
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	2.08		2.58		3.86		8.59			
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	1.96		2.72		4.26		9.47			
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	3.29		2.19		6.44		7.33			
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	1.86		2.25		3.2		7.34			
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	1.88		2.34		3.27		7.43			
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	1.86		2.25		3.25		7.36			
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	1.89		2.21		3.31		7.44			
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}$	1.9		2.26		3.26		7.24			
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	1.9		2.22		3.33		7.29			
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	1.93		2.2		3.28		7.37			

Tabelle 10.19: Experimentelle Laufzeit LimLee (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 140, 160, 192 und 272 Bits, in ms

LimLee uvP, 197 Bit					
Koordinaten			h	v	exp. ms
C	D	E			
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	4	3.81
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	8	4	3.912
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	8	4	4.2
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	9	2	4.39
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	8	4	4.48
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	4	3.89
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	8	4	3.86
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	4	3.88
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	4	3.87
$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	8	4	3.91
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}$	8	4	3.82
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	8	4	3.8
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	8	4	3.79

Tabelle 10.20: Experimentelle Laufzeit LimLee (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 197 Bits, in ms

Simultane Exponentiation ovP, alle Bitlängen								
Koordinaten				exp. ms				
A	B	C	D	140	160	192	197	272
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	20.63	21.7	31.0	39.79	74.18
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	20.7	22.2	31.81	40.56	74.78
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	21.09	22.2	31.73	40.71	75.88
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	22.86	24.12	34.57	44.16	82.68
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	20.31	25.02	36.79	42.24	85.24
$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	15.56	16.73	24.04	30.19	56.61
$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	15.8	16.93	24.45	30.67	57.56
$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	15.43	16.65	23.94	30.12	56.57
$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	15.73	16.92	24.37	30.64	57.46
$\mathcal{A}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	15.9	17.03	24.57	30.89	58.22
$\mathcal{A}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	15.91	17.06	24.56	31.0	58.32
$\mathcal{A}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	16.17	17.31	24.94	31.59	59.25
$\mathcal{A}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	16.23	17.39	25.01	31.6	59.14

Tabelle 10.21: Experimentelle Laufzeit Simultane Exponentiation (ALG 6.2.1) ohne Verwendung vorberechneter Punkte, alle Bitlängen, in ms

Simultane 2^w-ary Exponentiation ovP, 140 - 197 Bit										
Koordinaten					Art	w	exp. ms			
A	B	C	D	E			140	160	192	197
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	1	2	20.42	21.99	31.66	40.1
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	1	2	20.44	21.46	30.85	39.67
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	2	2	19.14	20.23	28.76	36.61
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2	2	21.69	22.81	32.1	40.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	1	2	20.22	25.1	36.76	42.18
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	2	15.66	16.66	23.43	29.75
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	2	15.58	16.63	23.4	29.55
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	2	15.71	16.56	23.34	29.64
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	2	15.59	16.69	23.43	29.7
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	2	15.89	17.09	23.97	29.87
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	2	15.82	17.0	23.97	29.79
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	2	15.77	16.94	23.86	29.63
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	2	15.73	16.97	23.81	29.65

Tabelle 10.22: Experimentelle Laufzeit Simultane 2^w-ary Exponentiation (ALG 6.2.2) ohne Verwendung vorberechneter Punkte, 140 - 197 Bits, in ms

Simultane 2^w-ary Exponentiation ovP, 272 Bit							
Koordinaten					Art	w	exp. ms
A	B	C	D	E			
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	1	3	74.8
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	1	3	73.87
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	2	2	67.7
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2	2	76.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	1	2	85.3
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	2	54.93
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	2	54.61
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	2	54.66
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	2	54.94
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	2	55.19
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	2	54.96
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	2	54.89
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	2	54.97

Tabelle 10.23: Experimentelle Laufzeit Simultane 2^w-ary Exponentiation (ALG 6.2.2) ohne Verwendung vorberechneter Punkte, 272 Bits, in ms

Simultane sliding window Exponentiation ovP, 140 - 197 Bits									
Koordinaten					w	exp. ms			
A	B	C	D	E		140	160	192	197
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	2	17.56	18.75	26.51	33.54
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	2	17.9	19.03	26.95	34.44
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	2	18.22	19.37	27.48	35.02
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	2	19.89	21.22	30.03	38.06
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	2	19.18	23.89	35.38	39.65
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	14.31	15.42	21.84	27.76
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	14.28	15.37	21.74	27.68
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	14.27	15.28	21.73	27.78
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	14.4	15.43	21.86	27.8
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	14.44	15.46	21.8	27.57
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	14.38	15.35	21.74	27.39
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	14.41	15.46	21.82	27.52
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	14.49	15.35	21.89	27.48

Tabelle 10.24: Experimentelle Laufzeit Simultane sliding window Exponentiation (ALG 6.2.3) ohne Verwendung vorberechneter Punkte, 140 - 197 Bits, in ms

Simultane sliding window Exponentiation ovP, 272 Bit						
Koordinaten					w	exp.
A	B	C	D	E		ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	3	62.02
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	3	64.2
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	3	66.26
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	3	71.33
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	2	79.72
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	51.24
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	50.91
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	51.0
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	51.01
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	2	50.77
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}$	2	50.44
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}$	$\mathcal{J}^{\mathcal{M}}$	2	50.45
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}$	2	50.66

Tabelle 10.25: Experimentelle Laufzeit Simultane sliding window Exponentiation (ALG 6.2.3) ohne Verwendung vorberechneter Punkte, 272 Bits, in ms

Basic interleaving Exponentiation ovP, 140 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	19.45
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	18.89
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	18.52
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	24.74
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	20.26
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	17.11
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	17.54
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	17.6
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	17.34
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	17.43
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	17.95
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	17.88
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}$	4	4	16.96

Tabelle 10.26: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 140 Bits, in ms

Basic interleaving Exponentiation ovP, 160 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	20.17
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	19.72
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	19.32
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	27.65
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	22.74
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	18.26
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	18.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	18.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	18.63
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	18.52
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	19.19
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}$	4	4	18.11
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	19.06

Tabelle 10.27: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 160 Bits, in ms

Basic interleaving Exponentiation ovP, 192 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	28.21
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	27.78
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	27.12
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	39.74
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	32.65
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	25.56
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	26.08
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	26.12
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	26.47
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	26.05
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	25.93
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	5	26.84
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	4	26.75

Tabelle 10.28: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 192 Bits, in ms

Basic interleaving Exponentiation ovP, 197 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	36.17
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	35.5
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	34.85
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	48.3
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	39.34
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	32.28
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	32.51
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	32.47
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	32.71
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	32.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	32.73
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	5	33.23
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	33.25

Tabelle 10.29: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 197 Bits, in ms

Basic interleaving Exponentiation ovP, 272 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	5	5	67.0
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	5	5	65.52
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	5	5	62.82
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	5	5	88.39
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	5	5	72.24
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	59.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	59.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	59.15
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	58.91
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	5	5	61.77
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	6	5	61.48
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	6	61.71
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	60.83

Tabelle 10.30: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) ohne Verwendung vorberechneter Punkte, 272 Bits, in ms

wNAF-based interleaving Exponentiation ovP, 140 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	3	3	17.52
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	3	3	17.97
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	18.35
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	21.93
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	21.56
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	15.68
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	15.85
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	15.9
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	16.0
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	16.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	16.17
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	16.4
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	5	16.27

Tabelle 10.31: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 140 Bits, in ms

wNAF-based interleaving Exponentiation ovP, 160 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	18.82
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	19.23
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	19.53
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	23.19
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	26.22
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	17.13
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	17.09
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	17.08
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	17.22
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	17.65
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	17.57
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	17.73
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	17.44

Tabelle 10.32: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 160 Bits, in ms

wNAF-based interleaving Exponentiation ovP, 192 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	25.89
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	26.46
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	26.9
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	32.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	37.67
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	23.6
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	23.87
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	23.82
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	24.02
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	24.33
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	24.1
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	24.48
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	24.54

Tabelle 10.33: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 192 Bits, in ms

wNAF-based interleaving Exponentiation ovP, 197 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	32.76
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	33.67
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	34.4
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	40.79
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	42.94
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	29.39
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	30.07
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	29.99
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	29.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	29.8
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	30.62
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	3	30.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	30.38

Tabelle 10.34: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 197 Bits, in ms

wNAF-based interleaving Exponentiation ovP, 272 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	4	4	58.98
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	4	4	61.76
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	4	4	63.13
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	4	4	75.75
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	4	4	85.43
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	4	53.56
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	4	53.7
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	5	53.85
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	5	54.53
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	4	54.16
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	4	3	54.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	5	3	54.75
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	3	5	54.58

Tabelle 10.35: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) ohne Verwendung vorberechneter Punkte, 272 Bits, in ms

Basic interleaving Exponentiation uvP, 140 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	15.87
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	15.67
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	15.57
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	20.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	17.23
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	14.85
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	14.98
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	15.13
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	15.19
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	15.03
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	15.44
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	15.2
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	15.38

Tabelle 10.36: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 140 Bits, in ms

Basic interleaving Exponentiation uvP, 160 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	16.92
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	16.65
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	16.56
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	22.63
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	19.17
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	15.95
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	16.02
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	16.28
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	16.3
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	16.18
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	5	16.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	16.12
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	16.36

Tabelle 10.37: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 160 Bits, in ms

Basic interleaving Exponentiation uvP, 192 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	23.67
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	23.36
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	23.19
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	32.46
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	27.33
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	22.38
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	22.43
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	22.81
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	5	22.88
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	22.81
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	22.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	22.63
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	22.95

Tabelle 10.38: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 192 Bits, in ms

Basic interleaving Exponentiation uvP, 197 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	30.03
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	29.75
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	29.63
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	39.46
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	33.41
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	28.17
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	28.61
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	28.36
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	28.72
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	28.49
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	28.99
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	5	29.18
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	28.79

Tabelle 10.39: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 197 Bits, in ms

Basic interleaving Exponentiation uvP, 272 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	4	55.9
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	4	55.22
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	4	54.98
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	10	4	76.4
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	10	4	64.16
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	52.55
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	52.36
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	5	52.97
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	52.7
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	5	53.6
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	53.41
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	53.38
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	53.6

Tabelle 10.40: Experimentelle Laufzeit Basic interleaving Exponentiation (ALG 6.2.4) unter Verwendung vorberechneter Punkte, 272 Bits, in ms

wNAF-based interleaving Exponentiation uvP, 140 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.48
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	3	15.3
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	3	15.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	19.11
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	3	18.39
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	13.49
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	13.72
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	13.55
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	3	13.74
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	13.83
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	13.86
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	13.87
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	13.83

Tabelle 10.41: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 140 Bits, in ms

wNAF-based interleaving Exponentiation uvP, 160 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	15.46
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	3	16.3
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	16.91
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	19.57
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	23.31
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	14.76
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.82
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	14.84
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	14.99
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.98
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	3	14.87
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.88
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	14.92

Tabelle 10.42: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 160 Bits, in ms

wNAF-based interleaving Exponentiation uvP, 192 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	21.51
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	22.79
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	23.69
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	11	4	27.82
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	33.89
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	20.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	20.75
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	20.58
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	20.89
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	20.91
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	21.03
$\mathcal{J}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	21.0
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	21.02

Tabelle 10.43: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 192 Bits, in ms

wNAF-based interleaving Exponentiation uvP, 197 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	27.06
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	11	4	28.96
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	11	4	30.09
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	8	3	35.71
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	11	4	39.28
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	25.58
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	25.81
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	25.97
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	5	26.02
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	26.05
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	11	4	26.32
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	3	26.39
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	11	4	26.24

Tabelle 10.44: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 197 Bits, in ms

wNAF-based interleaving Exponentiation uvP, 272 Bit					
Koordinaten			w_1	w_2	exp.
A	B	C			ms
$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	$\mathcal{J}^{\mathcal{M}}$	10	4	49.72
$\mathcal{J}$	$\mathcal{J}$	$\mathcal{J}$	10	4	53.44
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{C}}$	10	4	55.9
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	10	4	65.32
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	10	4	77.12
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	46.75
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	5	47.25
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	4	47.29
$\mathcal{J}^{\mathcal{C}}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	48.2
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	9	5	47.7
$\mathcal{A}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	3	47.87
$\mathcal{P}$	$\mathcal{A}$	$\mathcal{J}^{\mathcal{M}}$	10	4	48.19
$\mathcal{A}$	$\mathcal{J}^{\mathcal{C}}$	$\mathcal{J}^{\mathcal{M}}$	10	5	48.32

Tabelle 10.45: Experimentelle Laufzeit wNAF-based interleaving Exponentiation (ALG 6.2.5) unter Verwendung vorberechneter Punkte, 272 Bits, in ms

10.2 Laufzeitmessungen mit dem Kryptoprozessor

Um die Zeiten einordnen zu können, folgt zunächst eine Beschreibung der Implementierung:

Der Kryptoprozessor wurde von der Arbeitsgruppe *Integrierte Schaltungen und Systeme* und der Arbeitsgruppe *Theoretische Informatik - Kryptographie und Computeralgebra* im Rahmen eines Auftrags des BSI entworfen und implementiert. Ziel dieses Projektes war, einen ersten FPGA-basierten Prototyp zur Punktmultiplikation über Primkörpern zu implementieren. Die Skalierbarkeit des Körpers und der Gruppenordnung zwischen 160 und 512 Bit Länge war eine der Anforderungen, die gestellt wurden. Die Art des Einsatzes des Kryptoprozessors war uns nicht mitgeteilt worden. Insbesondere konnte dieser Prozessor nicht für eine spezielle Anwendung, wie Signaturerzeugung, -verifikation, Schlüsselaustausch oder Verschlüsselung, angelegt werden.

Der Schwerpunkt des Auftrags, der in diese Arbeit einfließt, lag dabei in der Auswahl der Algorithmen, der etwaigen Fenstergröße und des oder der Koordinatensysteme. Im Folgenden wird die Hardware, auf der das System implementiert wurde, beschrieben, dann die Körperarithmetik, die Wahl des Koordinatensystems und deren Begründung, sowie die Punktmultiplikationsmethode. Die Eingliederung der Hardware in den Provider und die experimentellen Ergebnisse schließen diesen Abschnitt ab. An dieser Stelle sei deutlich darauf hingewiesen, dass der Verfasser dieser Arbeit sich nicht mit Hardware-Implementierung befasst oder auskennt. Das Know-How, das von ihm in diesen Teil einfließt, umfasst lediglich die Körper- und EC-Arithmetik sowie die zur Integration in den Provider benötigten Java-Kenntnisse. Alle Informationen, die darüber hinaus gehen, basieren auf der Arbeit von und mit M. Ernst. Dieses Jahr wird auch er seine Dissertation fertigstellen, auf die dann für weitere Details verwiesen sei [Ern03].

10.2.1 Der Kryptoprozessor

Bei der Karte handelt es sich um eine Alpha Data ADM-XPL PCM-Karte mit *Virtex II Pro* FPGA-Technologie. In dieser Arbeit ist sie mit einem XC2VP20 FPGA Baustein bestückt (bis zu 2 Mio. System Gatter). Die physikalische Integration erfolgt über eine ADC-PMC-64 PSI-Adapterkarte [BB03].

10.2.2 Die Körperarithmetik

Die benötigte Körperarithmetik besteht auch hier aus der Addition, der Subtraktion, der Multiplikation und der Inversion modulo p. Die Addition bzw. Subtraktion modulo p stellt keine Schwierigkeiten dar und lässt sich speziell in Hardware sehr effizient implementieren. Anders die Multiplikation und insbesondere die Inversion.

Die Invertierung modulo p kann mit Hilfe des "Kleiner Satz von Fermat" ($a^{p-1} \equiv 1 \mod p$) oder dem erweiterten Euklid'schen Algorithmus berechnet werden [MvV97, S. 67 + S. 69]. Beide Methoden sind sehr aufwendig in ihrer Berechnung, wobei im Fall des "Kleiner Satz von Fermat" die Invertierung ausschließlich auf Additionen und Multiplikationen abgebildet wird und demzufolge auf Divisionen komplett verzichtet werden kann. Aus diesem Grund viel die Wahl auf letztere Technik.

Die Multiplikation modulo p lässt sich auf verschiedene Arten berechnen: Die klassische modulare Multiplikation [MvV97, S. 595], die Barrett Reduktion [MvV97, S. 603ff]) sowie die Montgomery Reduktion [MvV97, S. 600ff].

Das Montgomery Multiplikationsverfahren ist insbesondere für Hardware Implementierungen äußerst attraktiv, was auch durch die große Zahl an aktuellen Publikationen in diesem Bereich belegt wird [Wal99, PO01, TK99].

Die in [PO01, TK99, STK00] u.a. vorgeschlagenen Implementierungen der Montgomery Multiplikation haben ein charakteristisches gemeinsames Merkmal: Ähnlich wie bei der Architektur in [Gro01] wird einer der Operanden bitweise abgetastet, was allgemein als *Radix-2* Design bezeichnet wird. Neben einigen schaltungs- und implementierungstechnischen Vorteilen und den bekannten Nachteilen in Sachen Durchsatz, ergibt sich hierbei eine signifikante algorithmische Vereinfachung: Die im Laufe des Algorithmus benötigte Eingangs-Variable $m' = -m^{-1} \mod b$ reduziert sich zu der Konstanten $m' = 1$ und kann somit in der Implementierung "hart" codiert werden. Für Entwürfe mit höherem *Radix* (siehe [STK01] und [BM02]) ist eine derartige Vereinfachung nicht möglich. Dennoch versprechen Architekturen mit höherem *Radix* erhebliche Performanzsteigungen, da die Anzahl der Iterationen in der Montgomery Multiplikation deutlich verringert werden kann. Die in [TK99, STK00, STK01] beschriebenen Architekturen bauen erkennbar aufeinander auf und dokumentieren klar den Trend hin zu *High-Radix* Architekturen.

So wurde die Montgomery Multiplikation und Reduktion als die für diese Aufgabe passendste erkannt und gewählt.

10.2.3 Die Punktmultiplikation

Bei der Punktmultiplikation musste in Betracht gezogen werden, dass sie für jede mögliche kryptographische Anwendung möglichst effizient sein sollte. Insbesondere musste davon ausgegangen werden, dass keine vorberechneten Punkte zur Verfügung stehen würden.

Diese Gründe sprächen für die Exponentiation mit non-adjacent-forms. Während diese Methode in Software sehr schön zu implementieren ist, stellt sie in Hardware ein Problem dar: Jede Komponente in einer w-NAF ist eine ganze Zahl i mit $-2^w - 1 \leq i \leq 2^w - 1$, i ungerade. Auf Hardware, wo nur Bits zur Verfügung stehen, bräuchte jede Komponente w-Bit Platz: Das erste ist das Vorzeichenbit, die restlichen $w - 1$ Bit sind für die Zahlen 1 bis $2^w - 1$ nötig. Dies ist in Hardware sehr umständlich und platzraubend. Daher fiel die Entscheidung auf die nächst bessere Wahl, die Sliding Window Exponentiation. Die Fenstergröße w wurde im ersten Prototypen fest auf 4 gesetzt, was für sehr viele Bitlängen ohnehin die erste Wahl ist. Weiterentwicklungen sollten unterschiedliche Fenstergrößen jedoch ermöglichen.

10.2.4 Das Koordinatensystem

Ebenso wie eine feste Fenstergröße konnte in diesem ersten Prototyp nur eines der fünf Koordinatensysteme implementiert werden. Das liegt daran, dass man auf Hardware nicht trivial den Exponenten scannen kann. Während in Software eine Entscheidung, ob eine Addition oder eine Verdopplung folgt, leicht ist, ist sie in Hardware nicht ohne weiteres zu treffen. Daher wurde in der ersten Entwicklungs-Iteration das rein Jacobische Koordinatensystem gewählt.

10.2.5 Ergebnisse

Wir präsentieren die besten, mit dem Kryptoprozessor erreichten Laufzeiten. Dabei unterscheiden wir zwischen den Laufzeiten, die alleine für die Punktmultiplikation gemessen wurden, und denen, die die Zeit für den Aufruf aus dem Provider heraus beinhalten. Letztere werden im nächsten Kapitel präsentiert.

Die Tests wurden auf einem Intel Xeon 1.8 GHz mit 1 GB RAM unter Microsoft Windows 2000 Professional durchgeführt. Tabelle 10.2.5 fasst die Ergebnisse der EC Operationen zusammen. Dabei steht *Mult* für die einfache Punktmultiplikation, *Mult Konv.* für die einfache Punktmultiplikation mit anschließender Normie-

Bitlänge n	ADD	DBL	Mult	Mult + Konv.	Konv.
140	0.24	0.17	1.37	1.45	0.29
160	0.24	0.17	1.62	1.72	0.31
192	0.25	0.17	2.09	2.23	0.35
197	0.26	0.19	2.49	2.66	0.38
272	0.28	0.20	4.23	4.53	0.52
300	0.29	0.21	4.91	5.25	0.57
400	0.32	0.23	8.59	9.12	0.84
500	0.35	0.26	13.12	13.96	1.24

Tabelle 10.46: Laufzeit der EC Operationen auf dem Kryptoprozessor in ms

rung der Punkte und *Konv.* für die Normierung alleine. *ADD* und *DBL* bezeichnen wie gehabt die Punktaddition und -verdopplung.

Die Zeiten der fünften Spalte sind mit denen der Software-Implementierung mit den Algorithmen zu vergleichen, die keine vorberechneten Punkte verwenden. Die besten Zeiten, die unter dieser Voraussetzung gemessen wurden, betragen 12.22 ms, 13.21 ms, 18.51 ms, 23.23 ms und 41.97 ms für die Bitlängen 140, 160, 192, 197 und 272 (siehe Tabellen 10.12 bis 10.10). Es ist deutlich zu sehen, dass der Kryptoprozessor viel bessere Zeiten erreicht, als die Java-Implementierung. Abbildung 10.1 zeigt das Verhältnis Laufzeit FlexiECProvider : Kryptoprozessor. Mit steigender Bitlänger scheint sich der Einsatz des Kryptoprozessors auch immer mehr zu lohnen.

Erlaubt man in der Softwareimplementierung die Verwendung vorberechneter Punkte, schneidet LimLee besser ab (Siehe Tabellen 10.19 und 10.20). Dies ist in Abbildung 10.2 dargstellt. Man erkennt, dass auch bei steigender Bitlänge die LimLee-Implementierung dem Kryptoprozessor vorzuziehen ist.

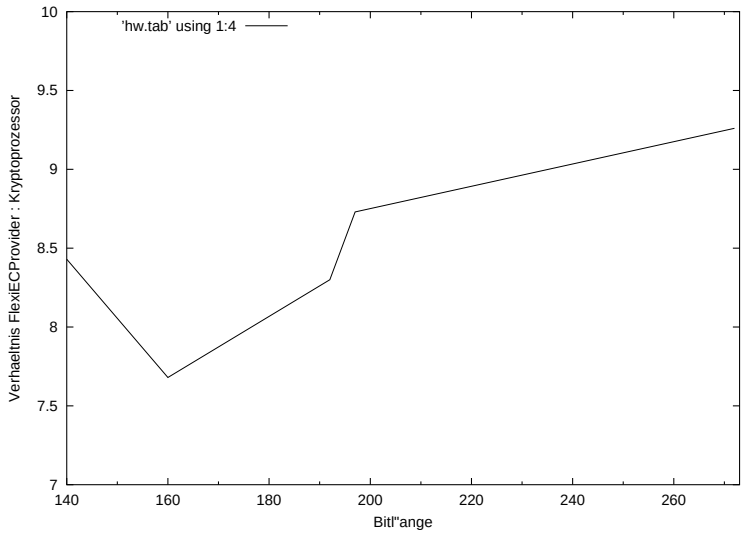


Abbildung 10.1: Verhältnis der Laufzeiten FlexiProvider mit Exponentiation with non-adjacent-forms, $w = 4$ und ovP, zu Kryptoprozessor mit Sliding Window, $w = 4$ und ovP

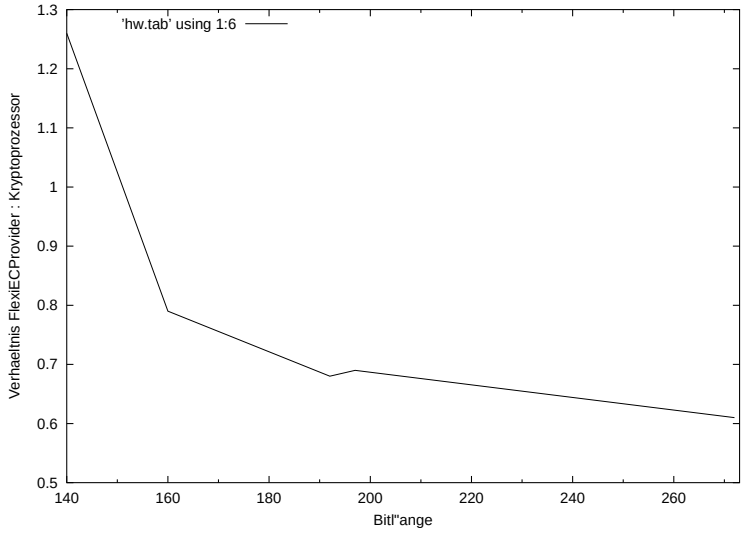


Abbildung 10.2: Verhältnis der Laufzeiten FlexiProvider mit Exponentiation with LimLee, uvP, zu Kryptoprozessor mit Sliding Window, $w = 4$ und ovP

Kapitel 11

Anwendungen

In diesem Kapitel soll aus den Ergebnissen des vorherigen Kapitels und aus experimentellen Ergebnissen für vier Referenzszenarien abgeleitet werden, welche Methoden und welche Koordinatensysteme die beste Wahl sind. Die vier Szenarien wurden bereits in den Anforderungen beschrieben: Der Arbeitsplatz zu Hause oder im Büro zum Versenden von Emails oder Online-Banking, ein iPAQ PocketPC für unterwegs zum online Ticketkauf, das Prüfen von digitalen Postwertzeichen sowie die Anfragen an einen Bankserver für Privatkunden.

11.1 Der FlexiECProvider für zu Hause und für das Büro

Bei dieser Anwendung handelt es sich um den inzwischen alltäglichen Gebrauch von Emails. Im Mittelpunkt stehen dabei die digitalen Signaturen. Iteressant wäre freilich auch die Verschlüsselung, doch kann sie nicht Teil dieser Arbeit sein, da es bisher kein standardisiertes Verfahren für elliptische Kurven gibt, das zur Klasse der Public-Key Verschlüsselungsverfahren gehört.

Elliptische Kurven sind durchaus auch in Protokollen wie TLS oder SSH einzusetzen, es werden darin jedoch nur die EC Signatur und der EC Schlüsselaustausch eingesetzt. Diese Verfahren werden in dieser Arbeit untersucht.

Betrachtet wird in diesem Abschnitt die Email-Anwendung mit der Möglichkeit Nachrichten zu signieren und signierte Nachrichten zu verifizieren. Dabei sollen die Anforderungen aus Kapitel 4 erfüllt werden:

Anforderungen

Die Integrierbarkeit wurde bereits durch den Einsatz des Providers mit dem SMIME Outlook Plugin [Tak] bewiesen. Durch das Framework JCA, mit den nach oben klaren Schnittstellen, kann jede Java-fähige Applikation auf jede Funktionalität des Providers zugreifen. In diesem Fall ist das Plugin diese Applikation, die zwischen Outlook von Microsoft und den Provider gestellt wird.

Ebenso ist die Anwendbarkeit bereits durch das Design der Parameterklassen in den Abschnitten 4.1 und 5.4.1 begründet. Und durch die Standardkonformität ist sichergestellt, dass gesendete Signaturen von anderen Providern und empfangene Signaturen vom FlexiProvider verifiziert werden können.

Da der Provider mit beliebigen Parametersätzen arbeitet, kann sich jeder Anwender seinen Sicherheitsgrad selber wählen. Wir wollen keine Vorschläge dazu machen, Ämter wie das Bundesamt für Informationstechnik oder die Regulierungsbehörde sind die richtigen Stellen, bei denen man sich über den Stand der Technik informieren kann.

Die Erweiterbarkeit ist für den Anwender zu Hause nur insofern relevant, als sie ihm die Möglichkeit bietet, sich immer mit den neuesten Verfahren auszustatten. Er wird den Provider kaum selber erweitern. Die Möglichkeit besteht aber und ist auch erlaubt, da der Code open source ist. Die Erweiterbarkeit ist bereits im Kapitel 5 beschrieben.

Es bleibt zu zeigen, dass der Provider der Anforderung Effizienz genügt. Die Forderung an die Geschwindigkeit ist, dass das Signieren und Verifizieren nicht 100 ms überschreiten darf.

Experimentelle Ergebnisse

Alle folgenden Tests wurden auf der selben Plattform ausgeführt wie die bisherigen Tests. Während diese aber jeweils die Zeit von 1000 Punktmultiplikationen gemessen haben, macht das hier keinen Sinn: Im Allgemeinen wird man zwar bis zu fünf Emails signieren, aber nur jeweils eine Signatur davon verifizieren. Der große Unterschied liegt in der Anzahl der Durchläufe, wie durch die nachfolgenden Ergebnisse gezeigt wird. Die Tests wurden mit den Schlüsselgrößen 160 und 192 Bit durchgeführt, die Bitlängen sind so gewählt, wie sie in [fSidI02] empfohlen werden. Eine Zusammenfassung der Laufzeiten aller Bitlängen is in Tabelle 12.3 zu finden.

Anwendung		LimLee, uvP		NAF, ovP	
		160 Bit	192 Bit	160 Bit	192 Bit
1×	Schlüsselerzeugung	8	10	21	34
1000×	Schlüsselerzeugung	2.231	3.463	13.715	19.033
1×	Signaturerzeugung	27	35	50	75
1000×	Signaturerzeugung	2.99	3.6	14.04	21.689

Tabelle 11.1: Szenario PC: Laufzeit Schlüssel- und Signaturerzeugung in ms pro Durchlauf

Bei diesem Szenario ist zu beachten, dass die vorberechneten Punkte mindestens einmal eingelesen und die dazu gehörigen Parameter erzeugt werden müssen. Dies kostet etwa 900 ms. Dabei kann man auf dem Standpunkt stehen, dass diese Zeit nicht mitberechnet werden sollte, da das Laden der Parameter schon beim Booten des Rechners oder beim Starten der Email-Anwendung ausgeführt werden kann: Da dies im Allgemeinen jeweils mehrere Sekunden dauert, würden die zusätzlichen 900 ms nicht ins Gewicht fallen.

Es würde jedoch einen tieferen Eingriff ins Betriebssystem oder in die Anwendung bedeuten, weshalb wir hier zwei unterschiedliche Lösungen anbieten: Einmal werden die Schlüssel und die Signatur mit LimLee und unter Verwendung vorberechneter Punkte erzeugt und einmal mit der Exponentiation mit Non-adjacent-forms - ohne Verwendung vorberechneter Punkte. Sie sind die jeweils schnellsten Methoden, die für diese Anwendung geeignet ist. Dabei werden im ersten Fall rein Jacobische Koordinaten und die Parameter $h = 7$, $v = 8$ und $h = 6$, $v = 16$ verwendet, um die bestmögliche Laufzeit zu erzielen. Auch im zweiten Fall wird die bestmögliche Ausprägung verwendet, mit Fenstergröße $w = 4$ und Modifiziert Jacobischen Koordinaten zur Punktverdopplung und Jacobischen Koordinaten zur Addition. Die Punkte aus der Vorberechnung werden affin zwischengespeichert (siehe Tabellen 9.21, 9.22 und 9.9). Tabelle 11.1 listet die Zeiten auf, die für eine Schlüsselpaar- und Signaturerzeugung benötigt werden.

Wie man sehen kann, liegt jede der Zeiten unterhalb von 100 ms. Wird nur eine Signatur mit LimLee erzeugt, ist die Laufzeit fast um ein Zehnfaches höher als wenn man die Zeit vieler Operationen misst und den Schnitt für eine einzelne berechnet. Bei der Erzeugung durch die Exponentiation mit Non-adjacent-forms ist das Verhältnis kleiner, der Effekt bleibt aber erhalten.

Die Schlüsselerzeugung ist schneller als die Signaturerzeugung. Das lässt sich darauf zurückführen, dass letztere um eine Invertierung modulo einer Primzahl und um das Hashen der Nachricht aufwendiger ist.

Anwendung		w-NAF-based, uvP		w-NAF-based, ovP	
		160 Bit	192 Bit	160 Bit	192 Bit
$1\times$	Verifikation	71	75	78	88
$1000\times$	Verifikation	15.502	21.689	17.886	24.214

Tabelle 11.2: Szenario PC: Laufzeit Verifikation in ms pro Durchlauf

Anwendung		NAF, ovP	
		160 Bit	192 Bit
$1\times$	Schlüsselaustausch	23.0	25.0
$1000\times$	Schlüsselaustausch	14.073	20.205

Tabelle 11.3: Szenario PC: Laufzeit eines Schlüsselaustauschs in ms

Tabelle 11.1 listet die Laufzeit einer Verifikation auf. Dabei werden wieder die Schlüssellängen 160 und 192 Bit betrachtet und die für diese Anwendung besten Methoden verwendet, w-NAF-based Interleaving: Die linke Hälfte enthält die Laufzeiten, die unter Verwendung vorberechneter Punkte entstanden, die rechte die Laufzeiten der Tests, die keine vorberechneten Punkte verwendeten.

Auch die Verifikation mit bis zu 192 Bit ist schnell genug, um der menschlichen Wahrnehmung zu entschlüpfen. Interessant ist, dass die Verwendung vorberechneter Punkte keinen sehr großen Vorteil bringt.

Zusätzlich präsentieren wir hier noch die Laufzeiten für einen Schlüsselaustausch. Dabei wird angenommen, dass keine vorberechneten Punkte zur Verfügung stehen. Es wird daher auch hier die Exponentiation mit Non-adjacent-forms verwendet mit der Ausprägung, die im Kapitel 9 als die beste berechnet wurde (siehe Tabelle 9.9). Tabelle 11.1 fasst die Ergebnisse für die interessantesten Bitlängen, 160 und 192 Bit, zusammen.

Auch der Schlüsselaustausch bleibt mit seiner Laufzeit weit unterhalb der Toleranzgrenze.

Es wurde somit gezeigt, dass der Provider alle Anforderungen erfüllt, die ein Email- oder SSH-Anwender stellen würde. Er bleibt in allen Fällen weit unterhalb der geforderten Laufzeit.

11.2 Der FlexiECProvider für unterwegs

Bei diesem Szenario handelt es sich um das mit den geringsten Ansprüchen und den höchsten Ansprüchen zugleich: Es wird zwar nicht die gleiche Laufzeit wie beim vorangegangen Szenario verlangt, sondern nur eine bis zu 1000 ms, doch auch diese wird in folgenden Tests nicht erreicht: Es wird sich zeigen, dass eine Signaturerzeugung innerhalb einer Sekunde zu schaffen ist, eine Verifikation diesen Zeitrahmen jedoch um ein Vielfaches sprengt.

11.2.1 Beschreibung der Implementierung

Im Rahmen des DFG Schwerpunktprojekts Sicherheit in der Informations- und Kommunikationstechnik bestand die Möglichkeit, Elliptische Kurven Kryptographie in ein gemeinsames Projekt einzugliedern und sie auf der CeBIT 2003 vorzuführen.

Es wird nur der Teil des Szenarios beschrieben, der für diese Arbeit relevant ist: Mit Hilfe eines mobilen Gerätes sollte es ermöglicht werden, drahtlos eine Verbindung zu einem Bahnserver aufzubauen und sich mit einem digitalen Münzsystem ein - ebenfalls digitales - Ticket für eine Bahnreise zu kaufen. Dabei sollte die Übertragung durch eine symmetrische Cipher geschützt und der dazu benötigte geheime Schlüssel vorher ausgetauscht werden. Digitale Signaturen sollten die Integrität und Authentizität schützen.

Zur Verfügung stand der iPAQ-PocketPC [Com02] von Compaq, mit dem Linux-Betriebssystem LuCAOS [BE], das von der Arbeitsgruppe Baumgarten und Eckert TU München entwickelt wurde.

Die Aufgabe der Arbeitsgruppe von Herrn Buchmann war es, Zertifikate, für die Signaturen und kryptographische Algorithmen zur Verfügung zu stellen. Die Wahl fiel auf ECDSA für die Signaturen und ECDH für den Schlüsselaustausch. Das gemeinsame Gesamtprojekt musste Ende Februar 2003 fertig gestellt sein, da es auf der CEBIT präsentiert wurde. Deswegen sind die Ergebnisse der Untersuchungen dieser Arbeit noch nicht in das Gemeinschaftswerk eingeflossen. Die Laufzeiten der CeBIT-Implementierung lassen sich daher nicht in die oben genannten eingliedern.

Die Anforderungen dieser Anwendung wurden bereits im Kapitel 4 beschrieben. Darin wurde jedoch noch nichts zur Sicherheit gesagt. Alle Mitglieder des DFG-Projekts diskutierten das Thema Laufzeit versus Sicherheit und legten die

Priorität fest, mit Fokus - naturgemäß - auf der CeBIT-Präsentation. Das Hauptproblem der gesamten Arbeit lag in der Laufzeit: Jedes Booten des iPAQ beansprucht mehrere Minuten, genauso wie der Aufbau des graphischen Interfaces. Daher wurde von der Gruppe einstimmig beschlossen, dass für den ersten Prototyp eine Schlüssellänge von 113 Bit ausreichen müsste. Es handelte sich in diesem Stadium schließlich noch nicht um eine Sicherheitsanwendung.

Zur Signaturerzeugung sowie zum Schlüsselaustausch wurde das LimLee-Exponentiations-Verfahren verwendet. Dabei wurden die Parameter auf $h = 6$ und $v = 5$ gesetzt. Es wurde nicht getestet, ob das tatsächlich die beste Wahl ist. Zu diesem Zeitpunkt waren weder die Strategie noch die Punktklasse `PointMixed` implementiert. Daher wurden alle Berechnungen durchgehend mit rein Jacobischen Koordinaten durchgeführt. Die Verifikation ist mit dem w-Naf-based Interleaving Verfahren realisiert.

11.2.2 Experimentelle Ergebnisse

Im Folgenden werden die Zeiten präsentiert, die eine Schlüsselerzeugung, eine Signaturerzeugung und eine Signaturverifikation im beschriebenen Szenario jeweils benötigen. Dabei muss beachtet werden, dass die Zeit nur mit `System.currentTimeMillis()` gemessen wurde. Diese statische Java-Methode misst die Realzeit, nicht die Prozessorzeit. Das heißt, dass in diesen Werten die Zeiten nebenläufiger Prozesse enthalten sind. Auf die Verwendung des Unix-Tools `time` musste verzichtet werden, da es nicht zum Umfang des zur Verfügung stehenden Linux-Pakets gehört.

Mehrere Tests hintereinander ausgeführt zeigen, dass die Zeiten erheblich schwanken. Das lässt sich mit der Auslastung des Gerätes begründen. So benötigt eine EC-Schlüsselerzeugung zwischen 0.42 und 0.43 Sekunden, eine ECDSA-Signaturerzeugung zwischen 0.8 und 0.9 Sekunden und eine Verifikation zwischen 3.9 und 4 Sekunden. Während sich die Laufzeiten der Erzeugung von Schlüsseln und Signaturen noch innerhalb der Toleranzgrenze halten - wenn auch nicht die Schlüssellänge - fällt die Verifikation mit bis zu 4 Sekunden gänzlich aus dem Rahmen. Doch während der CeBIT wurde die Erfahrung gemacht, dass die Präsentation trotz dieser Mängel Eindruck bei den Besuchern machte. Das ist ein Anreiz, weiter an Verbesserungen auch in diesem Bereich zu arbeiten und die in dieser Arbeit vorgeschlagenen Techniken auf dieses Szenario zu übertragen. Aus Zeitmangel muss dies jedoch in zukünftige Arbeiten einfließen.

Der Grund ist wahrscheinlich die Implementierung der BigInteger-Klasse: Von

der Version JDK1.2 zur JDK 1.3 ist die Laufzeit dieser Klasse erheblich verbessert worden. Es liegt der Verdacht nahe, dass die in diesem Szenario verwendete Version diese neue, schnellere Implementierung noch nicht beinhaltet. Das konnte jedoch nicht verifiziert werden, da keine Informationen darüber zu erhalten

11.3 Digitale Signaturen für Postwertzeichen

Wie bereits im Kapitel 4 beschrieben, handelt es ich in diesem Szenario darum, digitale Signaturen, die mittels Barcode auf Umschläge zu drucken, und Signaturen dieser Art zu verifizieren. Dabei bedruckt die Post die Umschläge, das heißt, es werden die Schlüssel der Post verwendet. Daraus folgt wiederum, dass sowohl zur Signaturerzeugung als auch zur Verifikation vorberechnete Punkte verwendet werden können.

Es wird davon ausgegangen, dass etwa 4,000.000 Umschläge pro Tag die Maschinerie durchlaufen. Angenommen, es stünde für jede dieser Aufgaben eine Maschine zur Verfügung, dann dürfte keine der beiden Operationen länger als 22 Millisekunden dauern.

Im letzten Kapitel, Abschnitt 10.2, wurden die Zeiten angegeben, die der Kryptoprozessor zum Zeitpunkt der Fertigstellung dieser Arbeit schon garantiert. Noch bei 272 Bit kann er eine einfache Punktmultiplikation in 7.5 ms auszuführen. Nun sind 272 Bit-Schlüssel eine Größe, die diese Anwendung sicher nicht benötigt: Würden die Umschläge mitsamt den Signaturen 50 Jahre nach dem Kauf vor ihrem Einsatz aufgehoben werden, wären sie vermutlich schon längst auf Grund von Tarifänderungen verfallen.

Die Verwendung von 197 Bit-Schlüsseln erscheint realistischer. Eine Punktmultiplikation benötigt mit dem Prozessor dann bis zu 4.4 ms, für eine Multiplikation der Form $k_1 \cdot P_1 + k_2 \cdot P_2$ daher weniger als 9 ms. Wie lange die Gesamtausführung von Erzeugung und Verifikation braucht, wird erst dann bekannt sein, wenn die Integrierung des Prozessors in den Provider abgeschlossen ist.

In der Zwischenzeit kann eine Software-Lösung betrachtet werden: Man kann bei diesem Szenario davon ausgehen, dass der Basispunkt im Voraus bekannt ist und so zumindest die Signaturerzeugung von der Methode LimLee und den vorberechneten Punkten profitieren kann. Und da die Post die Umschläge ausgibt und auch signiert, können auch bei der Verifikation vorberechnete Punkte verwendet werden.

Anwendung	Zeit in ms	
	197 Bit	192
10000× Schlüsselerzeugung	3.6132	3.1071
10000× Signaturerzeugung	3.8814	3.3655
10000× Signaturverifikation	26.8858	21.8555

Tabelle 11.4: Szenario Postwertzeichen: Laufzeit der Schlüssel- und Signaturgenerierung und Verifikation mit LimLee, bzw. w-Naf-based Interleaving in ms pro Durchlauf

Die folgenden Ergebnisse wurden wie folgt erreicht: Es wurden je 10.000 Schlüsselpaar- und Signaturerzeugungen und Verifikationen durchgeführt. In ersteren wurde die Methode LimLee verwendet, für letztere die w-Naf-based Interleaving Exponentiation. Für die drei Anwendungen wurde die jeweils best-berechnete Ausprägung verwendet, die in den Tabellen 9.23 und 9.46 nachgeschlagen werden können. Tabelle 11.3 fasst die Ergebnisse zusammen.

An ihr kann man erkennen, dass die Signaturen mit 197 Bit-Schlüssel auch mit dem FlexiProvider alleine gut durchzuführen sind. Die Verifikationen sind jedoch eindeutig zu langsam. Setzt man 192 Bit Schlüssel voraus, sind die hohen Anforderungen gerade noch einzuhalten.

Wir haben in diesem Fall daher zwei Lösungen zu bieten: Die erste ist, den KryptoProzessor zumindest für die Verifikationen einzusetzen. Er bietet hohen Komfort durch die Integrierung in den Provider, ist skalierbar und für viele Zwecke einsetzbar und erfüllt die Effizienzanforderungen. Auch kann der FlexiECProvider alleine verwendet werden, aber nur, wenn die Schlüssel die Bitlänge 192 nicht überschreiten. Andernfalls bietet die Kombination aus Provider und Kryptoprozessor die Flexibilität und Effizienz, die diese Anwendung benötigt.

11.4 Der FlexiECProvider für die Bank

Dieses letzte Szenario stellt die höchsten Anforderungen: Wie in Abschnitt 4.4.4 bereits beschrieben, werden bis zu 200 Anfragen pro Sekunde an Server für Privatkunden der Dresdner Bank gestellt. Das bedeutet, dass pro Anfrage nur 5 ms Zeit veranschlagt werden kann. In der Praxis verrichten mehrere Maschinen diese Arbeit, die Berechnungen werden mit spezieller Hardware ausgeführt. Dies ist somit ein Szenario, für das der KryptoProzessor gebaut wurde. Tabelle 11.4 fasst die

Bitlänge n	Schlüsselg.	Signaturg.	SignaturVer.
140	1.61	2.17	4.08
160	1.67	1.97	4.20
192	2.19	2.52	5.22
197	2.64	2.97	6.27
272	4.41	4.95	10.24
300	5.13	5.72	11.82
400	8.89	9.83	19.91

Tabelle 11.5: Laufzeiten der ECDSA Operationen unter Verwendung des Kryptoprozessors in ms

Laufzeiten der ECC Operationen zusammen, wobei aus dem Provider heraus der Kryptoprozessor für die Punktoperationen verwendet wurde.

Die Signaturerzeugung ist mit Hilfe des Kryptoprozessors somit auch bei 272 Bit schnell genug. Doch auch hier ist die Signaturverifikation noch der Schwachpunkt, an dem noch zu arbeiten ist.

Geht man jedoch davon aus, dass für die zwei Anwendungen je drei oder vier Maschinen zur Verfügung stehen, werden alle Anforderungen sogar bis zu 400 Bit erfüllt.

Kapitel 12

Zusammenfassung

Wir fassen die Ergebnisse dieser Arbeit zusammen: Schwerpunkt der Arbeit ist die Untersuchung, wie Punktmultiplikationen und damit Elliptische Kurven Kryptographie am effizientesten implementiert werden können. Der Weg dorthin ist in mehrere Stufen unterteilt:

Es wurde ein flexibler Provider für Elliptische Kurven Kryptographie über endlichen Primkörpern in Java implementiert. Dabei wurden die existierenden Standards berücksichtigt und deren Verfahren implementiert: ECDSA und ECNR für digitale Signaturen und ECDH für den Schlüsselaustausch. Durch die Architektur wird das Erweitern sowohl durch weitere kryptographische Verfahren als auch der Arithmetik über weiteren Körpern ermöglicht, ohne dass Änderungen an der Gesamtstruktur notwendig werden[1].

Der Provider enthält Punkteklassen: Zum einen die Klasse `Point`, die alle Methoden zu den Punktoperationen enthält, die vom zu Grunde liegenden Körper unabhängig sind. Insbesondere ist dies die Punktmultiplikation. Es wurden, um umfassende Tests durchführen zu können, die bekanntesten Algorithmen implementiert, die sich zur Punktmultiplikation eignen. Sie wurden den Arbeiten [MOC97, LL94, MvV97, Moe01, YLL94] entnommen und in einer späteren Stufe erweitert.

Die zweite Klasse ist `PointGFP` und enthält die Methoden, die für die restlichen Punktoperationen benötigt werden, insbesondere die Addition und die Ver-

[1] Der Provider unterstützt ebenfalls bereits die Körperarithmetik von endlichen Körpern der Charakteristik 2 und damit auch die kryptographischen Algorithmen, die auf elliptischen Kurven über diesen Körpern basieren. Diese Implementierung wird in dieser Arbeit jedoch nicht näher betrachtet.

dopplung von Punkten. Dabei ist den Punkten das Jacobische Koordinatensystem zu Grunde gelegt.

Neben dem Jacobischen Koordinatensystem gibt es das Affine, das Projektive, das Chudnowsky Jacobische [DVC87] und das Modifiziert Jacobische [CMO98] System. In der Arbeit [CMO98] wurde vorgeschlagen, diese verschiedenen Systeme so in eine Exponentiationsmethode einzusetzen, dass ihre jeweiligen Vorteile, nicht aber deren Nachteile, die Laufzeit beeinflussen und somit verbessern. Man spricht von gemischten Koordinaten. In diesem Paper wurde diese Technik am Beispiel der Exponentiation mit Non-adjacent-forms untersucht.

Dieser Ansatz wurde systematisch in dieser Arbeit fortgeführt: Anschließend an die Beschreibung der einzelnen Algorithmen zur Punktmultiplikation wurden für die Punktaddition und die Punktverdopplung die Anzahl der einzelnen benötigten Körperoperationen für jede der möglichen Koordinatenkombinationen berechnet. Diese Kombinationen werden in dieser Arbeit Ausprägungen genannt. Dabei wurde berücksichtigt, dass eine Punktaddition oder Verdopplung oft auf mehr als eine Art durchgeführt werden kann. In diesem Fall wurde die effizientere Lösung genommen.

Da es alleine für eine Punktaddition 5^3 und für eine Verdopplung 5^2 mögliche Kombinationen gibt, ist eine optimale Koordinatenauswahl pro Multiplikationsalgorithmus nicht trivial. Aus diesem Grund wurde in einer weiteren Stufe eine Strategie entwickelt, mit deren Hilfe - in Abhängigkeit von der Plattform, des Körpers und des Punktmultiplikationsalgorithmus - die optimale Ausprägung berechnet werden kann. Auch die Ausführung dieser Strategie ist noch sehr aufwendig. Daher wurde ein Java-Programm (Suite) entwickelt, das die einzelnen Schritte ausführt. Der erste Schritt beinhaltet das Messen der Laufzeit der einzelnen Körperoperationen. Dieser Schritt ist daher der von der Plattform abhängige. Ist er jedoch einmal getan, ermöglicht die Suite das Finden der optimalen Ausprägung eines Algorithmus auf derselben Plattform innerhalb von Sekunden.

Die Strategie ist geeignet für alle Plattformen und alle möglichen Punktmultiplikationsalgorithmen und Koordinatensysteme. Das Programm, das die Durchführung der Strategie effizient erlaubt, ist auf Java-Plattformen beschränkt, kann jedoch leicht auf andere übertragen werden.

In der nächsten Stufe der Arbeit wurde der Bogen zur Praxis gespannt: Für eine Plattform und 5 kryptographisch relevante Körpergrößen wurde die Strategie auf alle vorgestellten Algorithmen angewandt. Die ersten acht besten Ausprägungen wurden tabellarisch mitsamt ihren Laufzeiten für jeden der Punktmultiplikationsalgorithmen präsentiert und zusätzlich die Laufzeiten, die entstehen, wenn

ausschließlich eines der fünf Koordinatensysteme verwendet wird. Die Laufzeiten werden dabei als hochgerechnete Laufzeit in Millisekunden pro Punktmultiplikation präsentiert und sind auch als Komplexitäten zu verstehen. Der Grund, warum die hochgerechnete Zeiten statt der jeweiligen Anzahl der Körperoperationen angegeben werden ist, dass diese Anzahlen nicht immer auf den ersten Blick auf eine eindeutige Lösung schließen lassen.

Die Ergebnisse machen deutlich, dass das alleinige Verwenden einer Koordinatenart fast immer eine schlechte Wahl darstellt. Vor allem, wenn es sich um das Affine oder das Projektive System handelt. In vielen Fällen betragen die Differenzen der Laufzeiten zwischen dieser und denen der besten acht Ausprägungen mehrere Millisekunden.

Bei den vier verschiedenen Arten der Punktmultiplikation - a) der einfachen Punktmultiplikation ohne, b) der einfachen mit Verwendung vorberechneter Punkte, c) der mehrfachen ohne und d) der mehrfachen mit Verwendung vorberechneter Punkte - gibt es jeweils einen Punktmultiplikationsalgorithmus, der zum besten Ergebnis führt (siehe auch Tabelle 12.1).

Im ersten Fall a) nehme man die Exponentiation mit Non-adjacent-forms und bei Bitlängen zwischen 140 und 272 Bit eine Fenstergröße $w = 4$. Dabei sollen die Punkte in der Vorberechnung affin berechnet werden. Hintereinander auszuführende Punktverdopplungen sollen in diesem Algorithmus eine Modifiziert Jacobischen Punkt bekommen und das Ergebnis mit den gleichen Koordinaten zurückgeben. Eine Punktverdopplung vor einer Addition soll das Ergebnis in Jacobischen oder in Chudnowsky Jacobischen Koordinaten zurückgeben, eine Verdopplung nach einer Addition soll den Punkt in Jacobischen oder Modifiziert Jacobischen Koordinaten bekommen. Das sind vier Ausprägungen, die alle die gleiche Komplexität besitzen.

Ist die Verwendung vorberechneter Punkte möglich (Fall b)), ist die Methode von Lim und Lee die beste Wahl. Je Bitlänge sind hierbei unterschiedliche Parameter h und v zu wählen: Für 140 Bit $h = 10$ und $v = 2$, für 160 Bit $h = 7$ und $v = 8$, für 192 Bit $h = 6$ und $v = 16$, für 197 Bit $h = 8$ und $v = 4$ und für 272 Bit $h = 8$ und $v = 3$. Dabei übersteigt der benötigte Speicherplatz bei keiner dieser Größen 50 KB, die Punkte sind affin abgelegt. Acht Ausprägungen führen bei der Methode jeweils zum besten Ergebnis. Darunter fällt auch die, die ausschließlich Jacobische Koordinaten verwendet.

Eine doppelte Punktmultiplikation der Form $k_1 \cdot P_1 + k_2 \cdot P_2$ wird laut Hochrechnung sowohl mit als auch ohne Verwendung vorberechneter Punkte mit der w-Naf-based Interleaving Exponentiation ausgeführt (Fälle c) und d)). In beiden Fällen wird die Vorberechnung mit affinen Punkten ausgeführt, die Hauptrechnung

	140	160	192	197	272
$k \cdot P$, Naf ovP	12.22	13.25	18.57	23.23	41.97
$k \cdot P$, LimLee, uvP	1.86	2.2	3.2	3.79	7.24
$k_1 \cdot P_1 + k_2 \cdot P_2$, Si. Sli. Win., ovP	14.27	15.28	21.73	27.39	50.44
$k_1 \cdot P_1 + k_2 \cdot P_2$, w-Naf Interl., uvP	13.49	14.76	20.58	25.58	46.75

Tabelle 12.1: Zusammenfassung der besten experimentellen Ergebnisse in ms

mit Modifiziert Jacobischen. Stehen keine vorberechneten Punkte zur Verfügeng wähle man für jede der fünf Bitlängen $w_1 = w_2 = 4$. Ansonsten ist für die vier kleineren Bitlängen $w_1 = 11$ und $w_2 = 4$ zu wählen, für 512 Bit $w_1 = 10$ und $w_2 = 4$.

Die Differenzen der ersten acht besten berechneten Laufzeiten betragen oft nur Hundertstel Millisekunden. Statt von einer besten Ausprägung, sollte man daher von der besten Ausprägungsgruppe sprechen.

Da die Anzahl der Punktadditionen pro Multiplikation und damit auch die Laufzeit stark von der Anzahl der Einsen und deren Verteilung in der Binärdarstellung des Exponenten abhängt, können die berechneten Laufzeiten nicht als absolut angesehen werden werden. Es handelt sich um Erwartungswerte.

Doch die anschließenden experimentellen Tests bestätigen, dass dieser Weg trotzdem erfolgreich ist: Es wurden pro Bitlänge Tausend zufällige Exponenten gewählt, mit denen jeder Algorithmus mit ihren jeweils präsentierten Ausprägungen ausgeführt wurde. Die Ergebnisse wurden in der Arbeit tabellarisch aufgeführt, und analysiert. Meistens decken sich die experimentellen Ergebnisse mit den hochgerechneten. Schwankungen im Hundertsel- bis Zehntel-Millisekundenbereich können durch unterschiedliche Exponenten auftreten. Auch kann das Unix-Tool *time*, mit dem die Messungen vorgenommen wurden, nur mit einer Genauigkeit von Hundertstel-Millisekunden messen. Eine Ausnahme gibt es: Während bei den hochgerechneten Zeiten die w-Naf-based Interleaving Exponentiation besser abschneidet als die Simultane Sliding Window Exponentiation, ist dies bei den experimentellen Tests umgekehrt. Eine Untersuchung ergab, dass der Grund dafür in der tatsächlich benötigten Anzahl von Punktadditionen liegt: Diese weicht bei dieser Testreihe um mehrere Punktadditionen vom Erwartungswert ab. Dasselbe gilt für die Basic Interleaving Methode. Es müsste untersucht werden, ob diese Werte korrekt berechnet wurden. Dies muss jedoch auf spätere Arbeiten geschoben werden, da dies den zeitlichen Rahmen dieser Arbeit sprengen würde.

Tabelle 12.1 fasst die besten Zeiten der experimentellen Ergebnisse zusammen.

	140	160	192	197	272
$k \cdot P$, ovP	2.6	3.0	3.8	4.4	7.5

Tabelle 12.2: Zusammenfassung der bisherigen experimentellen Ergebnisse des KryptoProzessors in ms

Die Voraussetzung dieser experimentellen Tests war die Erweiterung der Algorithmen zur Punktmultiplikation sowie die Implementierung der Klasse `PointMixed`, die alle fünf Koordinatensysteme in sich vereint. Die Methoden zu Punktaddition und -verdopplung in dieser Klasse wurden so geschrieben, dass man in Form eines Parameters der Methode mitteilen kann, welcher Koordinatenart der Ergebnispunkt sein soll. Innerhalb dieser Methoden wird selbstständig festgestellt, welche Form die Summanden besitzen.

Um von außen auch die Verwendung verschiedener Koordinaten in den Punktmultiplikationsroutinen steuern zu können, wurden diese ebenfalls *parametrisiert*. Für jedes mögliche Ereignis, *hintereinander ausgeführte Verdopplung*, *hintereinander ausgeführte Addition*, *Übergang Addition Verdopplung* und *Übergang Verdopplung Addition*, gibt es je einen Parameter, der die Ausprägung der Punktmultiplikation steuert.

Das Gesamtpaket FlexiECProvider, Strategie und Strategie-Suite ermöglicht es so jedem Implementierer für seine eigene Plattform und Anwendung die optimalen Algorithmen und Ausprägungen zu finden und einzusetzen.

In Zusammenarbeit mit Markus Ernst aus dem Fachgebiet *Integrierte Schaltungen und Systeme* entstand zusätzlich ein KryptoProzessor, der FPGA-basiert eine einfache Punktmultiplikation ausführt. Diese Implementierung ist von 140 Bit bis 512 Bit skalierbar und wird zur Zeit in den Provider integriert. Die Laufzeiten, die bis zur Fertigstellung dieser Arbeit schon mindestens garantiert sind, werden in Tabelle 12.2 zusammengefasst.

In drei unterschiedlichen Szenarien wird nachgewiesen, dass der Provider entweder alleine oder doch zumindest mit Hilfe des KryptoProzessors den in Kapitel 4 definierten Anforderungen genügt.

So ist jeder Schlüsselautausch, Schlüsselgenerierung, Signaturerzeugung und -verifikation bei den relevanten Bitlängen 160 und 192 Bit in weit weniger als 100 ms zu berechnen, welche vom Menschen gerade noch nicht wahrgenommen werden. Der FlexiProvider ist somit für TLS, Email-Anwendungen oder SSH geeignet. Tabelle 12.3 fasst die Laufzeiten dieses Szenarios zusammen. Es werden darin die Laufzeiten pro Operation, angegeben. Dabei wurde einmal nur eine einzelne Ope-

Anwendung		140 Bit	160 Bit	192 Bit	197 Bit	272 Bit
1×	Schlüsselerzeugung	4	8	10	6	8
1000×	Schlüsselerzeugung	1.847	2.231	3.463	3.85	7.591
1×	Signaturerzeugung	26	27	35	37	47
1000×	Signaturerzeugung	2.079	2.99	3.6	4.308	8.121
1×	Verifikation, uvP	72.0	71	75	84	117
1000×	Verifikation, uvP	14.605	15.502	21.689	27.29	49.813
1×	Verifikation, ovP	78	78	88	98	122
1000×	Verifikation, ovP	16.385	17.886	24.214	31.209	55.742
1×	Schlüsselaustausch, ovP	21.0	23.0	25.0	43.0	61.0
1000×	Schlüsselaustausch, ovP	12.475	14.073	20.205	25.56	43.511

Tabelle 12.3: Szenario PC: Laufzeit Schlüssel- und Signaturgenerierung, Schlüsselautausch und Verifikation in ms pro Durchlauf

ration ausgeführt und deren Laufzeit gemessen und einmal wurde jede Operation je tausend mal hintereinander ausgeführt, die Gesamtzeit gemessen und der Schnitt pro Durchlauf berechnet. Es fällt auf, dass viele Durchläufe hintereinander zu viel besseren Zeiten führen als ein einzelner. Dies hilft in diesem Szenario nicht weiter, in den folgenden jedoch schon.

In dem Szenario, in dem Postwertzeichen mit digitalen Signaturen auf Umschläge gedruckt oder diese verifiziert werden sollen, darf eine Signaturerzeugung und -verifikation jeweils höchstens 22 Millisekunden dauern, wobei sehr viele Signaturen hintereinander erzeugt beziehungsweise verifiziert werden müssen. An Tabelle 12.3 erkennt man, dass der Provider diesen Zeiten gerade noch entsprechen kann, bei 197 Bit und mehr muss jedoch der KryptoProzessor verwendet werden.

Das letzte der drei Szenarien betrifft den Privatkundenserver einer Bank: In einer Sekunde werden bis zu 200 Anfragen gestellt, was bedeutet, das für eine Operation höchstens 5 ms veranschlagt werden dürfen. Unter der Annahme, dass mindestens zwei, wenn nicht drei Maschinen bereitgestellt werden, sind diese Anforderungen ebenfalls mit dem Kryptoprozessor zu bewältigen.

Bei einem Szenario wird deutlich, dass weitere Arbeit erforderlich ist: Der Provider wurde für ein CeBIT-Projekt auf einen iPAQ PocketPC übertragen, um drahtlose Verbindungen mittels Signaturen zu sichern. Obwohl die extrem kleine Schlüssellänge von nur 113 Bit verwendet wurde, sind die Laufzeiten vor allem der Verifikation jenseits des tolerablen Bereiches:

Anwendung	113 Bit
Schlüsselerzeugung	0.42 sec
Signaturerzeugung	0.9 sec
Verifikation, uvP	4 sec

In diese Implementierung sind zwar noch nicht die Ergebnisse dieser Arbeit eingeflossen, doch es ist trotzdem deutlich, dass an dieser Stelle noch weitere Arbeit von Nöten ist. Im Vergleich mit den Ergebnissen auf einem PC lässt sich jedoch stark vermuten, dass die ersten Schritte für eine Optimierung in der Verbesserung der Plattform liegen sollten und erst dann in der der Implementierung.

EC Parameter

Für die experimentellen Tests aus Kapitel 10 wurden mit folgenden EC Parameter und deren Basispunkte G ausgeführt. Die Kurvenparameter a und b, der Punkt G sowie dessen Ordnung r werden in Hexadezimal Darstellung, die Körpergröße p und und der Kofaktor k werden mit Dezimaler Darstellung präsentiert. Die Parameter sind mit der Zahlentheorie-Bibliothek LiDIA [LiDIA98] erzeugt worden.

```
140 Bit -> 1, 3, 6, 1, 4, 1, 8301, 3, 1, 2, 9, 0, 2
--------------------------------------------------

a: "2d28b928abc5a55e702f3cced29d8ef4a2",
b: "32686ef0af043db94362703e361162a2f02",
p: "1065893997345355660477881994661617905822031",
G: "04 000003f0 a4457928 137690b7 e0ebeec2 175e2d0a
       0000069a aec3ff65 22eb1eb8 41839158 b968c6e0",
r: "c3c6184b3f800159ac6b4713c9ac6a7f989",
k: "01"


160 Bit -> 1, 3, 6, 1, 4, 1, 8301, 3, 1, 2, 9, 0, 3
--------------------------------------------------

a: "1DCB49C5 8770F58C 69C79F97 F60DD78F 7118E821",
b: "1D78807E 19BD084D D030EACC A927D930 0C6D58D6",
p: "8282940187131411355335387472427078848647593387 59",
G: "04 2B34420F 73F08BD5 5A8B0E73 3DAB7880 A0CA2673
       21B657AB 425FCB27 C55249A1 D2717AC3 2427510F",
r: "9115FD05 8B000000 EA7838B4 A173F0DD 06F4B979",
k: "01"
```

```
192 Bit -> 1, 3, 6, 1, 4, 1, 8301, 3, 1, 2, 9, 0, 8
------------------------------------------------

a: "35469090367cbf9384c77359885915e8b74ff1c4de237e16",
b: "a8be3c57d41418c81f7e60be7ab92cb7f7297ef0789d952",
p: "39716160333463059538576194887410685981561170599\
    2613219749",
G: "04 8f9befff 7ce568f1 5c705e40 c8f1508b 54851e64
       4fd27c88 2de72cb2 218c4d36 ccc1d687 27423a52
       92f0b2cd 747665f9",
r: "a1f99b2d0c0000000000c7f3f3833784ba35c816ee657845",
k: "01"


197 Bit -> 1, 3, 6, 1, 4, 1, 8301, 3, 1, 2, 9, 0, 9
--------------------------------------------------

a: "63092cbdb7c7278d5411533217e7fef7d20771ac50d3f676",
b: "a81b066b4652f6fb38d6310f2309dd53065a2df4faff976cf",
p: "19299624867242523703141313655089295675202910194061
    51048859521",
G: "04 00000016 10a5bd92 30b86161 2c62eef7 428b4b62
       2624ace2 f69562d7 00000018 65a2ea40 a7c58610
       fe651b21 c2d19474 f4c252b7 9fac0c3e",
r: "1ebefedaa1c000000000068c78e8075a1889bd33fe1b6fdb7f",
k: "01"
```

```
// 272 Bit -> 1, 3, 6, 1, 4, 1, 8301, 3, 1, 2, 9, 0, 23
-------------------------------------------------

a: "53e92426c189f078448be81bf3000a4a0c0e974e03928ef428\
    35551e6d9a26c9260d",
b: "4c7c65aa9f9fdef3a4328c34c8e204ef6cf493292071915f0a\
    b6436f750d1a0dd598",
p: "554343970868281909694798971083474841214395456832 55\
    849265527646282170029632716983 41",
G: "04 3a7f 5ca06197 15ae5540 b1929b8c e758706f a46fb861
       2a05b81a 3ae311aa d9728893
       869c 2d7a759f 863d4ebf 73198fba bf03a26a 5a8723e2
       1bbacc81 5c554569 ff167bf0",
r: "bb0214672a8000000000000000001415994b98cece4f8d4ec\
    ab8603d85b754ca15f",
k: "01"
```

Literaturverzeichnis

[Bai] Harald Baier. Web-tool gec - Generate Elliptic Curves. http://pkidemo.cdc.informatik.tu-darmstadt.de/gec/overview.html.

[Bai02] H. Baier. *Efficient Algorithms for Generating Elliptic Curves over Finite Fields Suitable for Use in Cryptography*. PhD thesis, Darmstadt University of Technology, 2002.

[BB03] Nima Barraci and Sven Becker. Integration des Kryptoprozessors ECP in den Java-basierten FlexiECProvider. http://www.vlsi.informatik.tu-darmstadt.de/theses/theses_content.html#2%003, 2003. Studienarbeit.

[BE] U. Baumgarten and C. Eckert. LucaOS - Konzeption eines sicheren Betriebssystems für mobile Endgeräte der nächsten Generation. http://www13.in.tum.de/SPP/index.shtml.

[BM02] L. Batina and G. Muurling. Montgomery in practice: How to do it more efficient in hardware. In B. Preneel, editor, *Topics in Cryptology - CT-RSA 2002*, pages 40–52. LNCS, Springer-Verlag, 2002.

[Buc01] Johannes A. Buchmann. *Introduction to Cryptography*. Undergraduate Texts in Mathematics. Springer-Verlag, 2001.

[CMO98] Cohen, Miyaji, and Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 1998.

[Com02] Compaq. Technische daten - ipaq h 3800 familie. https://spps.iig.uni-freiburg.de/bscw/bscw.cgi/

d301-5/*/*/*/*/*/Compaq\%%20iPAQ\%20H3800\%3a\%20Technische\%20Daten, 2002.

[DVC87] G. V. Chudnovsky D. V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorisation tests. *Advances in Applied Mathematics*, 7:385 – 434, 1987.

[Ern03] Markus Ernst. *Configurable Hardware Architectures for Elliptic Curve Cryptosystems*. PhD thesis, TU Darmstadt, 2003. to be finished 2003.

[FG03] FlexiProvider-Group. FlexiProvider, a powerful toolkit for cryptography. http://www.flexiprovider.de, 2003.

[fSidI02] Bundesamt für Sicherheit in der Informationstechnik. Geeignete Kryptoalgorithmen. http:www.bsi.de, November 2002.

[Gor98] Rob Gordon. *essential JNI*. Prentice Hall, 1998.

[Gro01] J. Grossschaedel. A bit-serial unified multilier architecture for finte fields $GF(p)$ and $GF(2^m)$. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)*, pages 202–219. LNCS, Springer-Verlag, 2001.

[IEE] IEEE P1363 Draft standard specifications for public key cryptography. http://grouper.ieee.org/groups/1363/P1363/.

[Küh03] Ulrich Kühne. Serverauslastung. Persönliche Kommunikation, 2003. Dresdner Bank.

[Knu98] Jonathan Knudsen. *Java Cryptography*. The Java Series. O'Reilly, first edition, May 1998.

[Kob87] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Kra93] D. W. Kravitz. Digital signature algorithm. U.S. Patent 5, 231, 668, July 1993.

[LiDIA98] LiDIA. LiDIA 1.3.1 – *a library for computational number theory*. Technische Universität Darmstadt, 1998. Available via anonymous FTP from ftp.informatik.tu-darmstadt.de:/pub/TI/systems/LiDIA or via WWW from http://www.informatik.tu-darmstadt.de/TI/LiDIA.

[LL94] C. H. Lim and P. J Lee. More flexible exponentiation with precomputation. In Yvo G. Desmedt, editor, *Advances in Cryptology – CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 108–113. Springer-Verlag, August 1994.

[LV99] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 1999.

[Mil86] V. S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology - Crypto '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417 – 426. Springer Verlag, 1986.

[MOC97] A. Miyaji, T. Ono, and H. Cohen. Efficient elliptic curve exponentiation. In S. Quing Y. Han, T. Okamoto, editor, *International Conference on Information and Communications Security - ICIS '97*, volume 1334 of *Lecture Notes in Computer Science*, pages 282 – 290, 1997.

[Moe01] Bodo Moeller. Algorithms for multi-exponentiation. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2001*, pages 165–180. Springer-Verlag, 2001.

[MvV97] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, 1997.

[Nie94] Jakob Nielsen. *Usability Engineering*. AP Professional, Boston, 1994.

[NR93] K. Nyberg and R. Rueppel. A new signature scheme based on the DSA giving message recovery. In *First ACM Conference on Computer and Communications Security*, pages 58 – 61. ACM Press, 1993.

[PO01] C. Paar and G. Orlando. A scaleable $gf(p)$ elliptic curve processor architecture for programmable hardware. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)*, pages 348–363. LNCS, Springer-Verlag, 2001.

[SeM] SeMoA-Group. The secure mobile agents project. `http://www.semoa.org`.

[SH02] M. Schneider-Hufschmidt. Gestaltungsprinzipien und -standards, styleguides, guidelines (sommersemester 2002).

http://www.mmk.ei.tum.de/~alt/ebof/vlss02/ebof_ss02_vl03.pdf, 2002.

[Sil86] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, 1986.

[SM98a] Inc. Sun Microsystems. Java native interface specification. http://java.sun.com/products/jdk/1.1/docs/guide/jni/index.html, 1996-1998.

[SM98b] Inc. Sun Microsystems. Java cryptography architecture api specification & reference. http://java.sun.com/products/jdk/1.2/docs/guide/security/CryptoSpec.html, October 1998.

[Sol00] Jerome A. Solinas. Efficient arithmetic on Koblitz curves. *j-DESIGNS-CODES-CRYPTOGR*, 19(2–3):195–249, mar 2000.

[Sta00a] Standards for Efficient Cryptography Group. SEC 1, Elliptic Curve Cryptography. http://www.secg.org, September 2000.

[Sta00b] Standards for Efficient Cryptography Group. SEC 2, Recommended Elliptic Curve Cryptography Domain Parameters. http://www.secg.org, September 2000.

[STK00] E. Savas, A. F. Tenca, and C. K. Koc. A scaleble and unified multilier architecture for finte fields $gf(p)$ and $gf(2^m)$. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, pages 277–292. LNCS, Springer-Verlag, 2000.

[STK01] E. Savas, G. Todorov, and C. K. Koc. High-radix design of a scaleable modular multiplier. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)*, pages 185–201. LNCS, Springer-Verlag, 2001.

[Str64] E. Straus. Problems and solutions: Addition chains of vectors. *American Mathematical Monthly*, 71(806–808), 1964.

[SUN] SUN. *Java™ 2 Platform, Standard Edition, v 1.4.1 API Specification*. http://java.sun.com/j2se/1.4.1/docs/api/.

[Tak] Markus Tak. FlexiS/MIME Outlook Plugin. http://www.informatik.tu-darmstadt.de/TI/FlexiPKI/FlexiSMIME/FlexiSMIME%.html.

[TK99] A. F. Tenca and C. K. Koc. A scaleable architecture for montgomery multiplication. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 99)*, pages 94–108. LNCS, Springer-Verlag, 1999.

[tMWM03] Daniela Gerd tom Markotten, Sven Wohlgemuth, and Günter Müller. Mit sicherheit zukunftsfähig. *PIK Sonderheft „Sicherheit 2003“*, 26(1):5–14, 2003.

[Wal99] C. D. Walter. Montgomery's multiplication technique: How to make it smaller and faster. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 99)*, pages 80–93. LNCS, Springer-Verlag, 1999.

[Wis01] Wissenschaftlicher Rat der Dudenredaktion, editor. *Das Fremdwörterbuch*, volume 5 of *Duden*. Duden Verlag, 7., neu bearbeitete und erweiterte Auflage edition, 2001.

[WtMJM03] Sven Wohlgemuth, Daniela Gerd tom Markotten, Uwe Jendricke, and Günter Müller. DFG-Schwerpunktprogramm „ Sicherheit in der Informations- und Kommunikationstechnik“. *it - Information Technology, Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 45(1), 2003.

[X9.99] ANSI X9.62, public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA). Available from the ANSI X9 catalog, 1999.

[YLL94] S.-M. Yen, C.-S. Laih, and A. Lenstra. Multi-exponentiation. In *IEE Proceedings - Computers and Digital Techniques*, volume 141, pages 325–326, 1994.