Software and Hardware Implementation of Hyperelliptic Curve Cryptosystems

Dissertation

Submitted to the Fakultät für
Elektrotechnik und Informationstechnik
at the
Ruhr-Universität Bochum

for the
Degree of Doktor-Ingenieur

by

Thomas Wollinger

Bochum, Germany, May 2004

Thesis Advisor: Prof. Dr.-Ing. Christof Paar
Thesis Reader: Prof. Dr. Dr. h.c. Gerhard Frey

To my family, my son Marian, daughter Tabea, and wife Anja, for their endless love.

To my mother and father (especially on the occasion of his 60th birthday), to my sisters and brothers who always have supported me.

Abstract

The hyperelliptic curve cryptosystem is one of the emerging cryptographic primitives of the last years. This system offers the same security as established public-key cryptosystems, such as those based on RSA or elliptic curves, with much shorter operand length. Consequently, this system allows highly efficient computation of the underlying field arithmetic. However, until recently the common belief in industry and in the research community was that hyperelliptic curves are out of scope for any practical application. The reason being the complex group operation leading to a worse overall performance compared to established public-key primitives.

The thesis at hand is a step towards the practical use of hyperelliptic curve cryptosystems (HECC) by narrowing the performance gap between elliptic curve (EC) and hyperelliptic curve cryptosystems. We were able to reduce the complexity of the group operation for small genus hyperelliptic curves and we provide efficient algorithms for the computation of the hyperelliptic curve cryptosystem. Our theoretical comparison between elliptic curve and hyperelliptic curve cryptosystems, as well as our software and hardware implementations show that the performance of both cryptographic primitives are in the same range. Surprisingly, the hyperelliptic curve cryptosystems even outperforms elliptic curves using certain curve parameters. The implementations we investigated range from a general purpose processor to a variety of different embedded processors, and also includes the prototype implementation of a hyperelliptic curve coprocessor on FPGAs.

We were able to lower the complexity of the hyperelliptic curve group operations compared to the best known formulae. The highest performance on an embedded system was achieved on the ARM7TDMI running at 80MHz. The scalar multiplication (for a group order of approximately 2^{160}) for ECC, genus-2 HECC, and genus-3 HECC could be computed in about 100 milliseconds. More detailed analysis show that on embedded

processors one can improve the performance by almost a factor of 8 when combining instruction and data cache. Additional speed up of almost 50% can be gained when using a fixed underlying field and curve.

In addition, we developed a new metric to compare different cryptographic primitives based on the atomic operations of a processor, contrary to traditional methods using the bit complexity or specific timings. Hence, we provide a fair(er) and practical comparison metric not depending on special optimization tricks or theory.

We also analyzed the parallelism of the scalar multiplication on three different levels to find the optimal architecture for a hyperelliptic curve cryptosystem. We wrote a software tool for the simulation of the different architecture options. Our main finding is that architectures based on the inversion-free formulae should be preferred compared to those using group operations containing inversions. This kind of architecture performs the scalar multiplication (for a group order of approximately 2^{160}) in only $19,769$ clock cycles using three field multipliers (of type $D = 32$), one field adder and one field squarer.

Furthermore, we implemented various designs of HECC coprocessors on FPGAs. Our HECC coprocessor optimized for speed is considerably faster than the best previous implementations and the implementation designed for reduced silicon usage utilizes less area than the smallest design published.

Kurzdarstellung der Dissertation

Verschlüsselung basierend auf hyperelliptischen Kurven (HEC) ist ein asymmetrisches kryptographisches Verfahren das in den letzten Jahren zunehmend auf Interesse gestoßen ist. Diese Verschlüsselungstechnik gewährleistet dieselbe Sicherheit wie etablierte Verfahren (beispielsweise RSA oder elliptische Kurven), allerdings benötigen HEC kürzere Operandenlängen. Diese Eigenschaft führt zu einer effizienten Berechung der zugrunde liegenden Körperoperationen. Allerdings wurde bis vor kurzem vielfach angenommen, dass die Verschlüsselung basierend auf hyperellipischen Kurven wegen der komplexen Gruppenoperationen nicht für praktische Anwendungen geeignet ist.

In dieser Dissertation werden verschiedene Techniken vorgestellt, die HEC wesentlich praktikabler machen. Dies wurde durch die Reduzierung der Komplexität der Gruppenoperation für Kurven mit kleinem Geschlecht erreicht. Es werden effektive Algorithmen für ihre Berechnung vorgestellt. Die vorgestellten theoretischen Betrachtungen sowie die Software- und Hardware-Implementierungen zeigen, dass der Durchsatz von Verschlüsselungsverfahren basierend auf elliptischen und hyperelliptischen Kurven im selben Bereich liegt. Erstaunlicherweise übertreffen hyperelliptische Kurven unter bestimmten Voraussetzungen sogar die Verschlüsselungsgeschwindigkeit von elliptischen Kurven. Die Implementierungen, die in dieser Arbeit untersucht werden, reichen von Pentium Prozessoren über verschiedene eingebettete Prozessoren bis zu einer prototypischen Implementierung auf FPGAs.

Die in dieser Arbeit erzielten Resultate verringern die Komplexität der Gruppenoperation im Vergleich zu den besten Formeln, die bis dato publiziert wurden. Der beste Durchsatz auf den eingebetteten Prozessoren wurde auf dem ARM7TDMI mit einer Taktrate von 80MHz erzielt. Die Skalarmultiplikation (für eine Gruppenordnung von ungefähr 2^{160}) basierend auf hyperelliptischen Kurven konnte in unter 100ms für Kurven mit Geschlecht zwei und drei berechnet werden. Die genauere Analyse der Im-

plementierung auf eingebetteten Prozessoren zeigt, dass man den Durchsatz durch den Einsatz von Instruction Cache and Data Cache um einen Faktor von acht erhöhen kann. Ein zusätzlicher Geschwindigkeitsgewinn von bis zu 50% kann erreicht werden, wenn die Körper- und Kurvenparameter festgelegt sind.

Außerdem wurde eine neue Metrik entwickelt, welche verschiedene kryptographische Primitive basierend auf atomaren Operationen eines Prozessors vergleicht. Diese ist im Unterschied zu traditionellen Methoden, welche die Bit-Komplexität oder spezifische Zeitmessungen verwenden. Infolgedessen wird eine faire(re) und praktische Vergleichsmetrik zur Verfügung gestellt, die unabhängig von speziellen Optimierungen oder Theorien die Effizienz kryptographischer Algorithmen bewertet.

Des Weiteren werden in dieser Dissertation die Parallelität der Skalarmultiplikation analysiert, um die optimale Architektur für einen Coprozessor basierend auf hyperelliptischen Kurven zu finden. Das hierfür entwickelte Softwarepaket simuliert verschiedenste Architekturoptionen. Eines der Hauptergebnisse dieser Analyse ist, dass Architekturen basierend auf projektiven Koordinaten Vorteile gegenüber solchen basierend auf affinen Koordinaten aufweisen. Diese Art von Architekturen, die aus drei Körpermultiplizierern ($D = 32$), einem Körperaddierer und einem Körperquadrierer besteht, führt eine Skalarmultiplikation in 19.769 Taktzyklen aus.

Die Vorstellung verschiedener Designs von Coprozessor-Architekturen basierend auf hyperelliptischen Kurven auf FPGAs schließt die Arbeit ab. Der auf Geschwindigkeit optimierter Coprozessor ist um ein Vielfaches schneller als die beste veröffentlichte Implementierung. Die Implementierung die minimale Fläche zum Ziel hat, ist kleiner als die beste publizierte Architektur.

Preface

The results presented in this dissertation were accomplished in the three years I worked as researcher at the University of Bochum. I hope that the results presented are a step towards practical acceptance of hyperelliptic curve cryptosystems. I hope this work is of use in both university and industry settings in which applications of hyperelliptic curve cryptosystems are being studied.

The work I present in this dissertation would not have been possible without the help of many people that supported me throughout the last three years. First, and foremost many thanks go out to my advisor *Prof. Christof Paar*. His advice and expertise resolved many hurdles that I encountered throughout the research. I would also like to thank Prof. Christof Paar for the friendship and camaraderie that we developed while working together. I am grateful to my thesis committee, especially to Prof. Gerhard Frey for the valuable suggestions, comments and advise that they gave me.

I would like to thank my colleagues who by now have become good friends. Jan Pelzl with whom I developed the fast group operation and who wrote most of the software used in this thesis. Thanks also for all the nights we spend together writing yet another paper together (I will miss them!). The work related to find the optimal architecture for HECC would not have been possible without the help of Guido Bertoni (STMicroelectronics Milano) and Prof. Luca Breveglieri (Politecnico di Milano). Very special thanks to HoWon Kim, who spent almost a year in our group and did all the FPGA implementation

of the hyperelliptic curve cryptosystem.

Thanks to Jorge Guajardo who was (and is) the living cryptographic "reference book" for the group and who had an answer to all questions. I thank André Weimerskirch, Sandeep Kumar, and Kai Schramm for all the fruitful discussions and exchanging ideas. Thanks to our computer administrators, Marcel Selhorst and Christian Röpke, for being so patient with me. Thanks to all the students I supervised. Thanks for your contribution to this thesis and "helping" me to enlarge my leadership skills. And all the other guys in the COSY group, who from the fist day on were always in a good mood, provided a COSY atmosphere to work and were always willing to help. Not to forget our team assistant Irmgard Kühn who contributed quite a bit to the group atmosphere and was always a help when struggling with the university administration.

There are also several other people I would like to thank for contribution to my work. Thanks to Tanja Lange and Roberto Avanzi who patiently answered my endless number of questions about the theory of hyperelliptic curves and who proof red part of the thesis. Volker Wittelsberger who helped getting the tools for the embedded processors running. The ECC implementation was done jointly with Gökay Saldamli (Oregon State University) and Prof. Çetin K. Koç (Oregon State University). I wish to thank Jonathan Hammell and Daniel Klodt for correcting my English, a task which certainly belonged to the less enjoyable ones needed for the completion of this thesis.

Last but not least, I would like to thank my family, my parents, my sisters and brothers, and my friends for being patient with me and for their support. Special thanks go out to my wife Anja for her unconditional help and understanding during the past three years and especially during these last months writing the thesis. Thanks also to my children, Marian and Tabea, who were a great joy and motivation for me.

To all of you thank you very much!

Thomas

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

Historians divide the history of humanity in the year before and after Christ, B.C. versus A.D. If one would try to provide a similar division for the history of cryptography, one would probably choose the year 1976 to be the cryptographical year zero. Before 1976, (at least outside of the intelligence community) one did not have an elegant way for distributing keys over an insecure communication channel. The turning point in the cryptographic sense was the contribution of Whitfield Diffie and Martin Hellman [DH76] who introduced the concept of public-key cryptography.

Public-key cryptography can be used to provide the following services: a) key distribution and key establishment; b) digital signatures; and c) encryption. In modern applications, public-key primitives are used to provide all three services, in particular the services a) and b). For the encryption and authentication of large data streams one uses symmetric key algorithms because public-key algorithms are relatively inefficient. Hence, all protocols, such as IPSec [KA98], SSL [FKK96], and TLS [DA99], use both public and private cryptography and are therefore called hybrid protocols. Digital signatures have been a driving force behind the usage of public-key algorithms in the last decade. They provide integrity, sender authentication as well as non-repudiation. Thus, the sender of a message can not deny the creation of a message which can be crucial,

e.g., for online shopping.

Since 1976, three different variants of public-key cryptosystems of *practical relevance* have emerged, namely cryptosystems based on the difficulty of integer factorization (e.g., RSA [RSA78]), solving the discrete logarithm (DL) problem in finite fields (e.g., Diffie-Hellman key exchange [DH76] or Digital Signature Algorithm), and the DL problem in the group of algebraic curves over a finite field. Elliptic curve [Mil86, Kob87] and hyperelliptic curve cryptosystems (HECC) [Kob88] are the most well-known types of the last kind. HECC are a generalization of elliptic curve cryptosystems (ECC) and were suggested for cryptographic applications in 1988 by Koblitz.

Since their introduction, elliptic curve cryptosystems have been extensively studied not only by the research community but also in industry. In particular, there are several standards involving EC, such as the IEEE P1363 [P1399] standardization effort and the bank industry standards [ANS99]. The situation in the case of HECC is quite different. Until recently it was often believed that HECC are out of scope for practical application. Koblitz's idea to use HEC for cryptographic applications has only lately been analyzed and implemented both in software and in more hardware-oriented platforms such as field programmable gate arrays (FPGAs). However, there are still many open engineering problems regarding HECC and we will focus on some of them in this work.

The above mentioned asymmetric primitives can be used in a variety of different applications. It is obvious that the conditions for a practical implementation is very much application driven. Imagine a scenario where a number of PDAs communicate with a server. On the server side high data throughput is a necessity, because the server has to encrypt the data traffic of all personal devices. Area and power consumption is not critical on the server. However, moderate speed is acceptable on the PDAs, but the battery life and size of the device demand small area and low power. In short, both systems have very different requirements for the implementation of the cryptographic

primitives.

Additionally, one notices that more and more IT applications are embedded systems. In fact, 98% of all microprocessors sold today are embedded in household appliances, vehicles, and machines on factory floors [EGH00, BW00], whereas only 2% are used in PCs and workstations. We also should keep in mind that the number of microprocessors in most of our modern kitchens is higher than in our offices. Shortcomings of the embedded processors are that they possess CPUs with very low clock rates and a relatively small amount of memory. Hence, embedded processors often have 100 – 1000 times lower computational power than conventional PCs. In addition to many other challenges, the integration of security and privacy in the existing and new embedded applications will be a major one.

In this work we are going to study the usability of HECC on general purpose processors and embedded software and hardware platforms. Throughout our work we will present different implementation techniques targeting different platforms. We show that HECC is one of the cryptosystems that should be considered for use in embedded environments because of their short operand length. Considering the implementation aspects of the public-key algorithms, one notices that a major difference is the bit-length of the operands. It is widely accepted that for commercial applications one needs at least 1024-bit operands for RSA or the DL problem in finite fields. In the case of ECC or HECC applications, a group order of size approximately 2^{160} is believed to be sufficient for moderate security. Hence, one needs to work with at least 160 bit long numbers in the case of ECC. For HECC of genus two we will need a field $\mathbb{F}_q$ with 80-bit long operands, for the same level of security genus-3 HEC can even be realized securely with approximately 55-bit operands.

Most general purpose processors and also most embedded processors are designed to execute instructions in essence sequentially. As a consequence, one can only in limits use

the parallelism inherent in some cryptographic algorithms. However, hardware implementations allow for parallelism which can result in significant performance increases. Towards the end of this thesis we analyze the parallelism in HECC and present our prototype implementation of a HECC coprocessor on FPGA.

Some implementations introduced in this thesis are based on HEC over fields $\mathrm{GF}(2^m)$, where m is a composite integer. HEC over such fields have potential cryptographic weaknesses based on the Weil descent attack [Fre98, GHS02]. However, all introduced implementation techniques *do not use* the composite field structure to improve the performance, unlike [WBV+96, GP97]. This implies in particular that all software and hardware techniques introduced here are also applicable to HEC over fields with a prime extension.

1.2 Summary of Research Contribution

In this thesis, we mainly consider the engineering aspects of the cryptographic primitive HEC. We introduce four main contributions which we hope will improve the overall acceptance of HECC as one of the public-key primitives for future applications: 1) improvement of the group operation; 2) implementation of HECC on general and embedded processors; 3) finding optimal parallel architecture for HECC; 4) implementation of a HECC coprocessor on hardware-oriented platform, namely a FPGA. We also developed a metric for comparing the software performance of cryptographic primitives. Furthermore, we generalized Karatsuba's method for polynomial reduction to lower the computational complexity.

1.2.1 Improvement of the Group Operation

Part of this work was published in [PWGP03, PWP03, PWP04b, PWP04a].

The first algorithm to compute the group operation of HEC was presented by Cantor [Can87]. Cantor's approach works on elements given in Mumford's representation [Mum84]. We present an explicit description of the original Cantor algorithm. We were able to speed up the group addition and the group doubling using Cantor's idea for genus-2, 3, and 4 HEC compared to the best known work presented in [Nag00]. We could lower the complexity of the group addition and doubling formulae by up to 26% and 78%, respectively, compared to [Nag00]. We reached similar performance for the doubling operation for certain curves compared to Harley's algorithm.

In [GH00], the authors presented a modified version of the original Cantor algorithm. The idea of [GH00] was the basis for further optimization of HECC. The work at hand presents an improvement for the doubling operation in the case of genus-2 HEC. We show also for the first time generalized explicit formulae for genus-3 curves including fields of characteristic 2 and, in addition, we were able to decrease the complexity of these formulae. Furthermore, we present (for the first time) explicit formulae for genus-4 curves based on Harley's algorithm. Genus-4 HECC did not draw a lot of attention in the past because they seem to be far less efficient than genus-2 HECC, genus-3 HECC, and ECC. Our contribution saves 134 and 175 multiplications/squarings for the group addition and doubling, respectively, compared to previous work considering genus-4 curves by Nagao [Nag00]. However, genus-4 curves might have cryptographic weakness which may render them unsuitable for practical systems.

One of the techniques used to speed up the group operation was introduced by Karatsuba [KO63]. His algorithm efficiently computes the multiplication of two polynomials. Karatsuba multiplication can also be used to increase the performance in the reduction of two polynomials. We generalized Karatsuba's method in order to decrease the complexity of the polynomial reduction. We were able to obtain a similar cost reduction compared to Karatsuba multiplication, namely to save one multiplication for the

additional cost of three additions.

At this point we want to stress that the presented HECC group operations are most probably not the optimal formulae. Hence, our work should be considered an upper bound for the computational effort needed for HECC encryption rather than the best possible one. However, our results appear to be the best published ones to date. We encourage the research community to further reduce the complexity of the group operation of HECC.

1.2.2 New Complexity Metric for HECC and ECC

Part of this work was presented in [PWGP03].

A fair comparison between ECC and HECC was difficult to achieve due the different field sizes, types of operations, and the non-deterministic nature of the HEC operations, in particular, the computation of polynomial GCDs (Greatest Common Divisors) when using Cantor's algorithm. In addition, a lot of the published ECC results contain many platform specific optimizations which vary greatly between different implementations. We introduce a new metric for ECC and HECC over characteristic two fields which is based on an atomic operation count rather than on the (theoretical) bit complexity or specific timings.

The most interesting results are: (a) ECC, genus-2, and genus-3 HECC are in the same performance range (b) under certain conditions genus-2 and/or genus-3 hyperelliptic curves are faster than ECC at the same level of security. This result, however, implies to use very specific curves for HECC. Our new metric is validated by comparing our theoretical and practical results.

1.2.3 Software Implementation of HECC

Part of this work was published in [PWGP03, PWP03, WPW+04, PWP04b, PWP04a].

We implemented all group operations introduced in this contribution on an Intel Pentium Processor and on embedded microprocessors. One of the reasons to do so was to prove the correctness of our newly derived group formulae. Furthermore, we could show that HECC can reach the performance of ECC and, in some cases, even outperforms ECC.

We provide a thorough comparison of ECC and HECC, taking the latest advances in HECC group operations into account. We were able to achieve a competitive throughput for the cryptosystems implemented on a wide range of embedded platforms, namely ARM, ColdFire, and PowerPC. The best timings for the scalar multiplication with a group order of approximately 2^{163} for certain HEC cryptosystems could be achieved on ARM7TDMI running at 80MHz, resulting in 87 and 94 milliseconds for genus 2 and 3, respectively. The scalar multiplication for ECC could be performed on the same platform in 108 ms. Hence, HECC reaches the same throughput as ECC, and furthermore, in some cases HECC outperformed ECC. This fact holds for our implementations on the embedded as well as on the general processors. However, we want to point out, that the HECC implementations were based on curves using special parameters.

Furthermore, we show that for the two algorithm types implemented, the instruction cache on the PowerPC had a fundamental influence regarding the speed of one scalar multiplication. The time needed to perform one scalar multiplication can be decreased by more than a factor of three when using the instruction cache, and by almost a factor of eight when using instruction as well as data cache. In addition, we could speed up the throughput of the HEC scalar multiplication by up to 50% by focusing on a fixed underlying field and curve, which is an attractive option for implementations on

embedded systems. Combining all of these results, we showed that both families of algorithms are well suited for embedded applications.

In many low cost and embedded applications lower security margins are adequate. In practice, if a group order of 2^{128} is sufficient, we can perform the field operations of HECC with a smaller number of word operations. The underlying field operations can be implemented very efficiently for genus-4 HEC using 32-bit microprocessors (e.g., ARM). We provide a study of the efficiency of HECC considering this kind of low security application. However, one should keep in mind the security limitations of genus-4 HECC.

1.2.4 Optimized Parallel Architectures for HECC

Part of this work was published in [BBWP04b, BBWP04a].

We present optimized parallel architectures for a HECC considering the most recent explicit formulae to compute group operations. This was achieved by theoretically evaluating a variety of different architectures.

We investigate various parallel architectures for a genus-2 HECC using affine and projective coordinate systems considering the group order of approximately 2^{160}. We studied the parallelism of the HECC at three levels, namely the field operation level, the group operation level, and the scalar multiplication level. We analyzed the trade-offs between parallelization options, the latency, and area-time optimized configurations.

For the architecture using the affine coordinate system, we found that using a single multiplier with digit-size $D = 8$ is best suited. This architecture is able to compute the scalar multiplication in $76,797$ clock cycles. When providing more resources it is possible to lower the latency to $56,139$ clock cycles. Using projective coordinates the scalar multiplication can be performed in $19,769$ clock cycles using three field multipliers of type $D = 32$, one field adder and one field squarer, if area constraints are not considered.

However, the optimal solution in terms of latency and area uses two multipliers of type $D = 8$, one adder and one squarer. The main finding is that architectures based on the inversion-free formulae should be preferred compared to those using group operations containing inversions.

1.2.5 High Speed HECC Coprocessor on FPGA

There have been some efforts to implement HECC on hardware devices, like FPGAs [Wol01, WP02, BCLW02, Cla02, Cla03, EMY04]. Most of the implementations, however, consider the algorithm introduced by Cantor and not the explicit formulae.

We propose prototype implementations of genus-2 hyperelliptic curve cryptographic coprocessors. We present for the first time an FPGA implementation considering explicit formulae based on affine coordinates. We provide three different designs of the HECC coprocessor aiming for high performance. Additionally, we tried to reduce the area for two design options.

Our fastest version of the HECC coprocessor can perform one scalar multiplication in $415\mu s$ and is therefore 81% faster than the best previous implementation. The reduced area implementation utilizes 81% less area than the smallest design published.

The HECC coprocessor designs were evaluated by considering both the hardware requirements and the time constraints of the cryptographic application. Using the area-time product, our coprocessors is an order of magnitudes better than previous publications. Through this improvement, HECC is now approaching the performance range of ECC FPGA implementations.

1.3 Outline

In Chapter 2 we introduce the mathematical background of hyperelliptic curve cryptosystems, restricting ourselves to the material necessary to understand the dissertation. We show the basic definitions and properties, introduce divisors and the divisor class group. With these definitions we are able to define the Jacobian of the HEC. Next we define a polynomial representation of the equivalence classes. Furthermore, we introduce the group operations based on Cantor's and Harley's algorithm. We conclude Chapter 2 with some considerations about hyperelliptic curve cryptosystems and their security.

Chapter 3 summarizes the previous work on HECC and is divided into four sections. The first two sections cover all publications dealing with the improvement of the group operations based on Cantor's and Harley's algorithm. Parts three and four introduce the previous publications implementing HECC in software and hardware.

In Chapter 4 we state our improvements on the group operations for the hyperelliptic curve cryptosystem. We first introduce the methods used for speeding up the group operation, especially the generalization of the reduction using Karatsuba's algorithm. Then we introduce our optimized group operations based on Cantor's algorithm, Harley's algorithm, and the inversion-free formulae introduced by Lange. Finally we end this chapter with a summary.

In Chapter 5 we introduce our new metric for a theoretical comparison of various cryptographic primitives implemented on a certain processor. We focus on the underlying assumptions of our metric and apply it to a comparison of ECC and HECC on a 32-bit processor.

Chapter 6 deals with the implementation of HECC in software. In the first part we present the methodology used for the software implementation. Secondly, we show our ECC and HECC software implementations on general purpose processors. In the third

part, implementation results on embedded processors (ARM, ColdFire, and PowerPC) are presented. Furthermore, we validate our theoretical comparison metric introduced in Chapter 5 using the ARM implementations. Furthermore, we study the performance of HECC for low cost security applications. We end this chapter with a short summary.

Chapter 7 describes the analysis of the parallelism of the HECC group addition, group doubling, and scalar multiplication. We provide a theoretical comparison of different architecture options. Parallelism of HECC was exploited on three different levels, namely on the field arithmetic level, the group operation level, and the scalar multiplication level. We target affine and inversion-free formulae for genus-2 curves. The analysis results in the introduction of the optimal architecture for HECC. Finally we present a summary and an outlook.

The fastest known HECC implementation results achieved on FPGA are presented in Chapter 8. We propose three different prototype implementations targeting a genus-2 hyperelliptic curve cryptosystem using affine coordinates. We describe the HECC coprocessor, outline our methodology, the different design options, and finally put our results in perspective to previous ECC and HECC FPGA implementations.

We end this dissertation with the conclusions of our work and some suggestions for further research.

2 Mathematical Background

In this section, we present an elementary introduction to the theory of hyperelliptic curves over finite fields of arbitrary characteristic, restricting attention to material that is relevant for this work. For more details, the reader is referred to [Kob89,Kob98,MWZ98].

The idea that groups formed from hyperelliptic curves (HEC) are suitable for discrete logarithm cryptosystems was first introduced 1988 by Neal Koblitz [Kob88]. Hyperelliptic curves are a special class of algebraic curves and can be viewed as generalization of elliptic curves. There are hyperelliptic curves of every genus $g \geq 1$. A hyperelliptic curve of genus $g = 1$ is an elliptic curve.

Most of the material presented in this chapter was taken from [MWZ98] and some of the examples are from [Gau00a]. For an introduction to algebraic geometry the reader is referred to [Ful69].

2.1 Basic Definitions and Properties

This section provides the main definitions and properties of hyperelliptic curves.

Definition 2.1.1 *If a field* $\mathbb{F}$ *has the property that every polynomial with coefficients in* $\mathbb{F}$ *factors completely into linear factors, then we say that* $\mathbb{F}$ *is* algebraically closed.

Definition 2.1.2 *[MWZ98] Let $\mathbb{F}$ be a finite field, and let $\overline{\mathbb{F}}$ be the algebraic closure of $\mathbb{F}$. An equation of the form*

$$\mathcal{C} : v^2 + h(u)v = f(u) \qquad \text{in } \mathbb{F}[u, v], \tag{2.1}$$

where $h(u) \in \mathbb{F}[u]$ is a polynomial of degree at most g, $f(u) \in \mathbb{F}[u]$ is a monic polynomial of degree $2g + 1$, and there are no solutions $(u, v) \in \overline{\mathbb{F}} \times \overline{\mathbb{F}}$ which simultaneously satisfy the equation $v^2 + h(u)v = f(u)$ and the partial derivative equations $2v + h(u) = 0$ and $h'(u)v - f'(u) = 0$ defines a hyperelliptic curve $\mathcal{C}$ of genus g over $\mathbb{F}$ $(g \geq 1)$.

A *singular point* on a curve $\mathcal{C}$ is a solution $(u, v) \in \overline{\mathbb{F}} \times \overline{\mathbb{F}}$ which simultaneously satisfies the equation $v^2 + h(u)v = f(u)$ and the partial derivative equations $2y + h(u) = 0$ and $h'(u)y - f'(u) = 0$. From Definition 2.1.2 we see that hyperelliptic curves do not have any *singular point.*

Lemma 2.1.3 *[MWZ98] Let $\mathcal{C}$ be a hyperelliptic curve over $\mathbb{F}$ defined by Equation (2.1).*

1. *If $h(u) = 0$ then $\text{char}(\mathbb{F}) \neq 2$.*
2. *If $\text{char}(\mathbb{F}) \neq 2$, then the change of variables $u \to u, v \to (v - h(u)/2)$ transforms $\mathcal{C}$ to the form $v^2 = f(u)$ where $f \in \mathbb{F}[u]$*
3. *Let $\mathcal{C}$ be an equation of the form (2.1) with $h(u) = 0$ and $\text{char}(\mathbb{F}) \neq 2$. Then $\mathcal{C}$ is a hyperelliptic curve if and only if $f(u)$ has no repeated roots in $\overline{\mathbb{F}}$.*

For the proof of Lemma 2.1.3 consult [MWZ98].

Definition 2.1.4 *[MWZ98] Let $\mathbb{K}$ be an extension field of $\mathbb{F}$. The set of $\mathbb{K}$-rational points on $\mathcal{C}$, denoted $\mathcal{C}(\mathbb{K})$, is the set of all points $P = (x, y) \in \mathbb{K} \times \mathbb{K}$ that satisfy (2.1),*

together with a special point at infinity denoted ∞*. The set of points* $\mathcal{C}(\overline{\mathbb{K}})$ *will simply be denoted by* $\mathcal{C}$*. The points in* $\mathcal{C}$ *other than* ∞ *are called finite points.*

Note, the point at infinity lies in the projective plane $P^2(\mathbb{F})$. It is the only projective point lying on the line at infinity that satisfies the homogenized hyperelliptic curve equation. If $g \geq 2$ then ∞ is a singular (projective) point (which is allowed since ∞ is not in $\overline{\mathbb{F}} \times \overline{\mathbb{F}}$).

Definition 2.1.5 *[MWZ98] Let* $P = (x, y)$ *be a finite point on a hyperelliptic curve* $\mathcal{C}$*. The* opposite *point of* P *is the point* $\tilde{P} = (x, -y - h(x))$*. We also define the opposite of* ∞ *to be* $\tilde{\infty} = \infty$ *itself. If a finite point* P *satisfies* $P = \tilde{P}$*, then the point is said to be* special*; otherwise, the point is said to be* ordinary*.*

Figure 2.1 shows an example of a hyperelliptic curve over the field of real numbers.

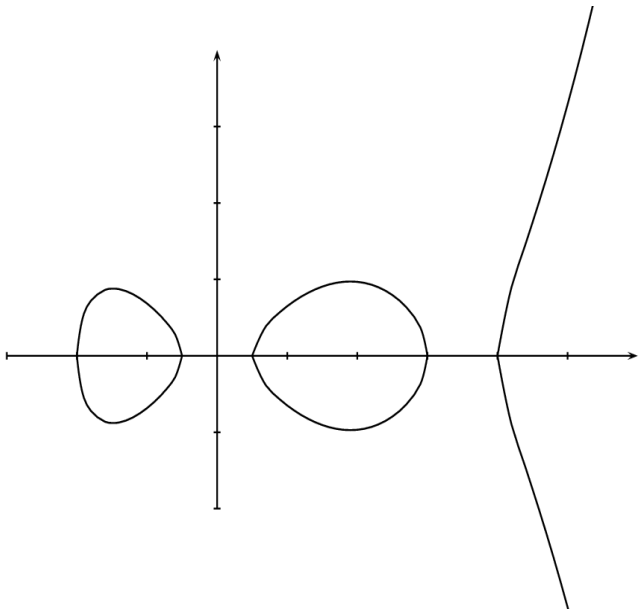


Figure 2.1: Hyperelliptic curve $\mathcal{C}$.

2.2 Polynomials and Rational Functions

In this section we only introduce the properties of polynomials and rational functions when they are viewed as functions on a hyperelliptic curve.

Definition 2.2.1 *[MWZ98] The coordinate ring of $\mathcal{C}$ over $\mathbb{F}$, denoted $\mathbb{F}[\mathcal{C}]$, is the quotient ring*

$$\mathbb{F}[\mathcal{C}] = \mathbb{F}[u, v]/(v^2 + h(u)v - f(u)),$$

where $(v^2 + h(u)v - f(u))$ denotes the ideal in $\mathbb{F}[u, v]$ generated by the polynomial $v^2 + h(u)v - f(u)$. Similarly, the coordinate ring of $\mathcal{C}$ over $\overline{\mathbb{F}}$ is defined as

$$\overline{\mathbb{F}}[\mathcal{C}] = \overline{\mathbb{F}}[u, v]/(v^2 + h(u)v - f(u)).$$

An element of $\overline{\mathbb{F}}[\mathcal{C}]$ is called a polynomial function on $\mathcal{C}$.

Lemma 2.2.2 *[MWZ98] The polynomial $r(u, v) = v^2 + h(u)v - f(u)$ is irreducible over $\overline{\mathbb{F}}$, and hence $\overline{\mathbb{F}}[\mathcal{C}]$ is an integral domain.*

The proof for Lemma 2.2.2 is shown in [MWZ98]. Note that each polynomial function $G(u, v) \in \overline{\mathbb{F}}[\mathcal{C}]$ is represented as $G(u, v) = a(u) - b(u)v$, where $a(u), b(u) \in \overline{\mathbb{F}}$, are unique.

Definition 2.2.3 *[MWZ98] Let $G(u, v) = a(u) - b(u)v$ be a polynomial function in $\overline{\mathbb{F}}[\mathcal{C}]$. The conjugate of $G(u, v)$ is defined to be the polynomial function $\overline{G}(u, v) = a(u) + b(u)(h(u) + v)$.*

Definition 2.2.4 *[MWZ98] Let $G(u, v) = a(u) - b(u)v$ be a polynomial function in $\overline{\mathbb{F}}[\mathcal{C}]$. The norm of G is the polynomial function $N(G) = G\overline{G}$.*

The norm is used to transform questions about polynomial functions in two variables into easier questions about polynomials in a single variable. In the next lemma the properties of the norm are given and the proof can be found in [MWZ98].

Lemma 2.2.5 *[MWZ98] Let $G, H \in \overline{\mathbb{F}}[\mathcal{C}]$ be polynomial functions.*

1. *$N(G)$ is a polynomial in $\overline{\mathbb{F}}[u]$.*
2. *$N(\overline{G}) = N(G)$*
3. *$N(GH) = N(G)N(H)$.*

Definition 2.2.6 *[MWZ98] The function field $\mathbb{F}(\mathcal{C})$ of $\mathcal{C}$ over $\mathbb{F}$ is the field of fractions of $\mathbb{F}[\mathcal{C}]$. Similarly, the function field $\overline{\mathbb{F}}(\mathcal{C})$ of $\mathcal{C}$ over $\overline{\mathbb{F}}$ is the field of fractions of $\overline{\mathbb{F}}[\mathcal{C}]$. The elements of $\overline{\mathbb{F}}(\mathcal{C})$ are called rational functions on $\mathcal{C}$.*

It is important to point out that $\overline{\mathbb{F}}[\mathcal{C}]$ is a subring of $\overline{\mathbb{F}}(\mathcal{C})$ and that every polynomial function is also a rational function. Next, we are going to define the value of a rational function at a finite point.

Definition 2.2.7 *[MWZ98] Let $R \in \overline{\mathbb{F}}(\mathcal{C})$, and let $P \in \mathcal{C}$, $P \neq \infty$. Then R is said to be defined at P if there exist polynomial functions $G, H \in \overline{\mathbb{F}}[\mathcal{C}]$ such that $R = G/H$ and $H(P) \neq 0$; if no such $G, H \in \overline{\mathbb{F}}[\mathcal{C}]$ exist, then R is not defined at P. If R is defined at P, the value of R at P is defined to be $R(P) = G(P)/H(P)$.*

One can see from the definitions above, that the well-defined value $R(P)$ is independent of the choice of G and H. The following definition will introduce the degree of the polynomial function and in Lemma 2.2.9 we state the properties of the degree.

Definition 2.2.8 *[MWZ98] Let $G(u,v) = a(u) - b(u)v$ be a non-zero polynomial function in $\overline{\mathbb{F}}[\mathcal{C}]$. The degree of G is defined to be*

$$\deg(G) = \max[2\deg_u(a), 2g + 1 + 2\deg_u(b)].$$

Lemma 2.2.9 *[MWZ98] Let $G, H \in \overline{\mathbb{F}}[\mathcal{C}]$.*

1. $\deg(G) = \deg_u(N(G))$.
2. $\deg(GH) = \deg(G) + \deg(H)$.
3. $\deg(G) = \deg(\overline{G})$.

Now we are going to define the value of the rational function at ∞.

Definition 2.2.10 *[MWZ98] Let $R = G/H \in \overline{\mathbb{F}}(\mathcal{C})$ be a rational function.*

1. *If $\deg(G) < \deg(H)$ then the value of R at ∞ is defined to be $R(\infty) = 0$.*
2. *If $\deg(G) > \deg(H)$ then R is not defined at ∞.*
3. *If $\deg(G) = \deg(H)$ then $R(\infty)$ is defined to be the ratio of the leading coefficients (with respect to the* deg *function) of G and H.*

2.3 Zeros and Poles

The purpose of this section is to introduce the orders of zeros and poles of rational functions as well as the notion of a uniformizing parameter. The proofs of all the lemmas and theorems can be found in [MWZ98].

Definition 2.3.1 *[MWZ98] Let $R \in \overline{\mathbb{F}}(\mathcal{C})^*$ and let $P \in \mathcal{C}$. If $R(P) = 0$ then R is said to have a zero at P. If R is not defined at P then R is said to have a pole at P, in which case we write $R(P) = \infty$.*

Lemma 2.3.2 *[MWZ98] Let $G \in \overline{\mathbb{F}}[\mathcal{C}]^*$ and $P \in \mathcal{C}$. If $G(P) = 0$ then $\overline{G}(\tilde{P}) = 0$.*

The following three lemmas are used in Theorem 2.3.6 that introduces the existence of uniformizing parameters.

Lemma 2.3.3 *[MWZ98] Let $P = (x, y)$ be a point of $\mathcal{C}$. Suppose that $G = a(u) - b(u)v \in \overline{\mathbb{F}}[\mathcal{C}]^*$ has a zero at P and that x is not a root of both $a(u)$ and $b(u)$. Then $\overline{G}(P) = 0$ if and only if P is a special point.*

Lemma 2.3.4 *[MWZ98] Let $P = (x, y)$ be an ordinary point on $\mathcal{C}$ and let $G = a(u) - b(u)v \in \overline{\mathbb{F}}[\mathcal{C}]^*$. Suppose that $G(P) = 0$ and x is not a root of both $a(u)$ and $b(u)$. Then G can be written in the form $(u - x)^s S$, where s is the highest power of $(u - x)$ which divides $N(G)$, and $S \in \overline{\mathbb{F}}(\mathcal{C})$ has neither a zero nor a pole at P.*

Lemma 2.3.5 *[MWZ98] Let $P = (x, y)$ be a special point on $\mathcal{C}$. Then $(u - x)$ can be written in the form $(v - y)^2 S(u, v)$, where $S(u, v) \in \overline{\mathbb{F}}(\mathcal{C})$ has neither a zero nor a pole at P.*

Theorem 2.3.6 *[MWZ98] Let $P \in \mathcal{C}$. Then there exists a function $U \in \overline{\mathbb{F}}(\mathcal{C})$ with $U(P) = 0$ such that the following property holds: for each polynomial function $G \in \overline{\mathbb{F}}[\mathcal{C}]^*$, there exists an integer d and function $S \in \overline{\mathbb{F}}(\mathcal{C})$ such that $S(P) \neq 0, \infty$ and $G = U^d S$. Furthermore, the number d does not depend on the choice of U. The function U is called a uniformizing parameter for P.*

Using Theorem 2.3.6, one can define the order of a polynomial function at a point.

Definition 2.3.7 *[MWZ98] Let $G \in \overline{\mathbb{F}}[\mathcal{C}]^*$ and $P \in \mathcal{C}$. Let $U \in \overline{\mathbb{F}}(\mathcal{C})$ be uniformizing parameter for P, and write $G = U^d S$ where $S \in \overline{\mathbb{F}}(\mathcal{C})$, $S(P) \neq 0, \infty$. The order of G at P is defined to be* $\mathrm{ord}_P(G) = d$.

Lemma 2.3.8 *[MWZ98] Let $G_1, G_2 \in \overline{\mathbb{F}}[\mathcal{C}]^*$ and $P \in \mathcal{C}$, and let* $\mathrm{ord}_P(G_1) = r_1$, $\mathrm{ord}_P(G_2) = r_2$.

1. $\mathrm{ord}_P(G_1 G_2) = \mathrm{ord}_P(G_1) + \mathrm{ord}_P(G_2)$.
2. *Suppose that $G_1 \neq -G_2$. If $r_1 \neq r_2$ than* $\mathrm{ord}_P(G_1 + G_2) = \min(r_1, r_2)$. *If $r_1 = r_2$ then* $\mathrm{ord}_P(G_1 + G_2) \geq \min(r_1, r_2)$.

Next, we generalize Lemma 2.3.2 and get Lemma 2.3.9.

Lemma 2.3.9 *[MWZ98] Let $G \in \overline{\mathbb{F}}[\mathcal{C}]^*$ and $P \in \mathcal{C}$. Then* $\mathrm{ord}_P(G) = \mathrm{ord}_{\bar{P}}(G)$.

Theorem 2.3.10 *[MWZ98] Let $G \in \overline{\mathbb{F}}[\mathcal{C}]^*$. Then G has a finite number of zeros and poles. Moreover,* $\sum_{P \in \mathcal{C}} \mathrm{ord}_P(G) = 0$.

In the following we are going to define the order of a rational function at a point.

Definition 2.3.11 *[MWZ98] Let $R = G/H \in \overline{\mathbb{F}}(\mathcal{C})^*$ and $P \in \mathcal{C}$. The order of R at P is defined to be* $\mathrm{ord}_P(R) = \mathrm{ord}_P(G) - \mathrm{ord}_P(H)$.

2.4 Divisors

This section introduces the concept of divisors and theirs basic properties.

Definition 2.4.1 *[MWZ98] A divisor is a finite formal sum of $\mathcal{C}(\overline{\mathbb{F}})$-points,*

$$D = \sum_{P_i \in \mathcal{C}} m_i P_i, m_i \in \mathbb{Z}$$

where only a finite number of the m_i are non-zero. The degree of D, denoted $\deg D$*, is the integer $\sum_{P \in \mathcal{C}} m_P$. The order of D at P is the integer m_P: we write* $\mathrm{ord}_P(D) = m_P$.

Example: [Gau00a] Intuitively, the order of a function f at a point P is the measure of the multiplicity of zero of f, see Figure 2.2, i.e. multiplicity of the intersection of the curve $\mathcal{C}$ with $f = 0$.

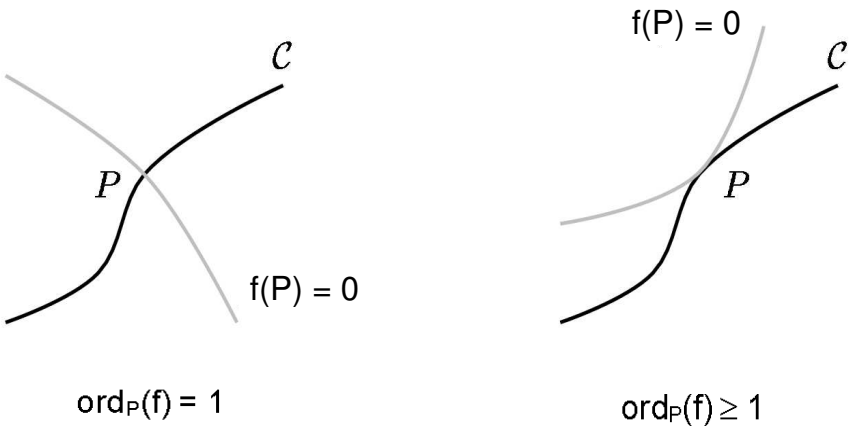


Figure 2.2: Multiplicity of zero [Gau00a].

Example: Assume we have a hyperelliptic curve $C : v^2 + uv = u^5 + 5u^4 + 6u^2 + u + 3$ over $\mathbb{F}_7$, an example of a divisor is

$$D = 2(2,2) + 3(5,3) + (1,1) + (6,4)$$

and the degree of D is

$$\sum m_i = 2 + 3 + 1 + 1 = 7.$$

Definition 2.4.2 *The number of points of a divisor is called the weight of the divisor.*

The set of all divisors, denoted by $\mathbb{D}$, forms an additive group under the addition rule:

$$\sum_{P_i \in \mathcal{C}} m_i P_i + \sum_{P_i \in \mathcal{C}} n_i P_i = \sum_{P_i \in \mathcal{C}} (m_i + n_i) P_i$$

Let $\mathbb{D}^0$ denote the subgroup consisting of divisors of degree 0.

In the next two definitions we introduce the GCD of divisors and the divisor of a rational function.

Definition 2.4.3 *[MWZ98] Let $D_1 = \sum_{P \in \mathcal{C}} m_P P$ and $D_2 = \sum_{P \in \mathcal{C}} n_P P$ be two divisors. The greatest common divisor of D_1 and D_2 is defined to be*

$$\gcd(D_1, D_2) = \sum_{P \in \mathcal{C}} \min(m_P, n_P) P - (\sum_{P \in \mathcal{C}} \min(m_P, n_P)) \infty.$$

Definition 2.4.4 *[MWZ98] Let $R \in \overline{\mathbb{F}}(\mathcal{C})^*$. The divisor of R is*

$$div(R) = \sum_{P \in \mathcal{C}} (\mathrm{ord}_P R) P.$$

Consulting Theorem 2.3.10, one notices that the divisor of a rational function is a finite formal sum and has degree 0.

Lemma 2.4.5 *Let $G \in \overline{\mathbb{F}}[\mathcal{C}]^*$, and let $div(G) = \sum_{P \in \mathcal{C}} m_P P$. Then $div(\overline{G}) = \sum_{P \in \mathcal{C}} m_P \tilde{P}$.*

2.5 Principal Divisors

For the definition of the Jacobian, principal divisors are needed.

Definition 2.5.1 *[MWZ98] A divisor $D \in \mathbb{D}^0$ is called principal divisor if $D = div(R)$ for some rational function $R \in \overline{\mathbb{F}}(\mathcal{C})^*$. The set of all principal divisors is denoted as $\mathbb{P}$.*

Example: [Gau00a] Let f be a function. The divisor $\mathrm{div}(f)$ can be decomposed into a difference of two divisors:

$$\mathrm{div}(f) = \mathrm{div}_0(f) - \mathrm{div}_\infty(f),$$

where $\mathrm{div}_0(f)$ corresponds to the intersection of $\mathcal{C}$ with the curve $f = 0$ and $\mathrm{div}_\infty(f)$ to the intersection of $\mathcal{C}$ with $\frac{1}{f} = 0$.

In Figure 2.3, we show an example, where $\mathrm{div}(f) = P_1 + P_2 + P_3 + P_4 - (2Q_1 + 2Q_2)$.

2.6 The Jacobian $\mathbb{J}$

Definition 2.6.1 *[MWZ98] The quotient group*

$$\mathbb{J} = \mathbb{D}^0/\mathbb{P}$$

is called the Jacobian of the curve $\mathcal{C}$. If $D_1, D_2 \in \mathbb{D}^0$ and $D_1 - D_2 \in \mathbb{P}$ then we write $D_1 \sim D_2$. In that case, D_1 and D_2 are said to be equivalent divisors.

Hence, the Jacobian is a finite quotient group of one infinite group by another infinite group. Every element on the Jacobian is an equivalence class of divisors. Figure 2.4,

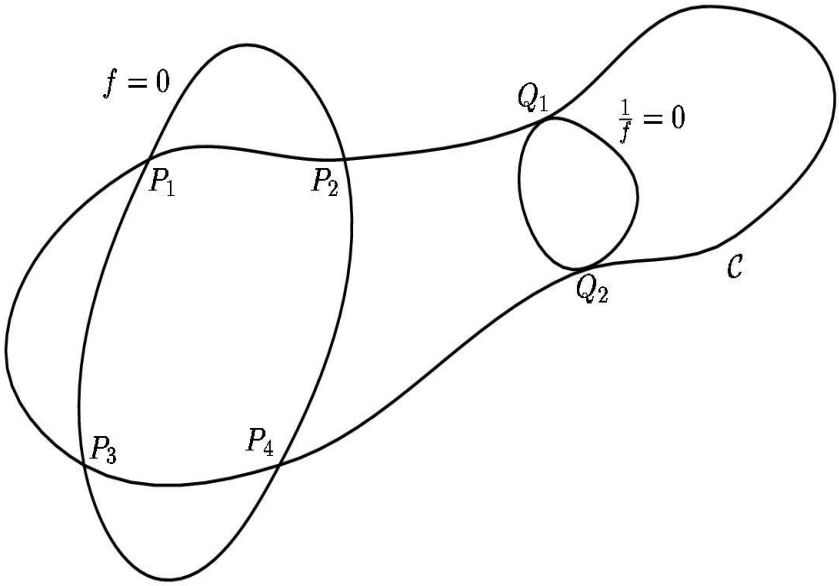


Figure 2.3: Example of a principal divisor [Gau00a].

shows two rational functions defined over a curve (plotted over $\mathbb{R}$ for visualization purposes).

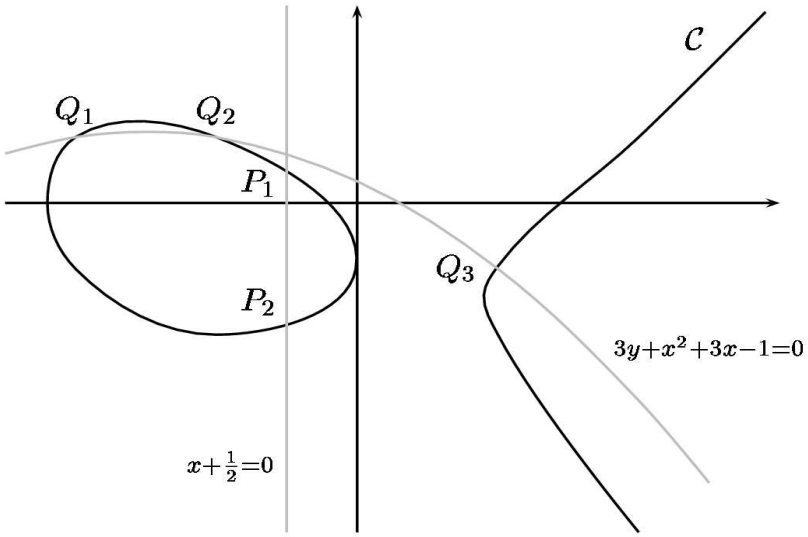


Figure 2.4: Example of rational functions defined over a curve $\mathcal{C}$ [Gau00a].

A convenient set of coset representatives and algorithms to perform operations are needed for the computations on $\mathbb{J}$. In the Section 2.8 the unique representation of elements in the Jacobian group is being introduced. The addition rules are stated in Section 2.9 and Section 2.10. Before doing so we will introduce reduced divisors in the next section.

2.7 Reduced Divisors

This section provides the definition of reduced divisors. We first have to define the support and the semi-reduced divisor to be able to get the reduced divisor.

Definition 2.7.1 *[MWZ98] Let $D = \sum_{P_i \in \mathcal{C}} m_i P_i$ be a divisor. The* support *of D is the set*

$$supp(D) = \{P_i \in \mathcal{C} \mid m_i \neq 0\}.$$

Definition 2.7.2 *[MWZ98] A semi-reduced divisor is a divisor of the form*

$$D = \sum_{P_i \in \mathcal{C}} m_i P_i - \left(\sum_{P_i \in \mathcal{C}} m_i \right) \infty$$

where each $m_i \geq 0$ and the $P_i \in \mathcal{C} \backslash \{\infty\}$ are finite points such that when $P_i \in supp(D)$ then $\tilde{P}_i \notin supp(D)$, unless $P_i = \tilde{P}_i$, in which case $m_i = 1$.

The following lemma shows that for each element $D \in \mathbb{D}^0$ there exists a semi-reduced divisor D_1 equivalent to D:

Lemma 2.7.3 *[MWZ98] For each divisor $D \in \mathbb{D}^0$ there exists a semi-reduced divisor $D_1 \in \mathbb{D}^0$ such that $D \sim D_1$.*

The proof can be found in [MWZ98].

Note that semi-reduced divisors are not unique in their equivalence class. In the case of hyperelliptic curves, one can show a stronger result (using the Riemann-Roch theorem, see [Ful69]) that every element of $\mathbb{J}$ can be uniquely represented by a so-called reduced divisor. The reduced divisors are defined as follows:

Definition 2.7.4 *[MWZ98] Let* $D = \sum_{P \in \mathcal{C}} m_i P_i - (\sum_{P \in \mathcal{C}} m_i)\infty$ *be a semi-reduced divisor. If* $\sum_{P \in \mathcal{C}} m_i \leq g$ *(*g *is the genus of* $\mathcal{C}$*) then* D *is called a* reduced divisor*.*

Hence, a divisor $D = \sum m_i P_i - (\sum m_i)\infty \in \mathbb{D}^0$ is said to be reduced if:

1. All of the m_i are non-negative, and if P_i is equal to its opposite then $m_i \leq 1$.
2. If $P_i \neq \tilde{P}_i$, then P_i and $\tilde{P}_i$ do not both occur in the sum.
3. $\sum m_i \leq g$.

One can show that each coset of the quotient group $\mathbb{J} = \mathbb{D}^0/\mathbb{P}$ has exactly one reduced divisor.

Theorem 2.7.5 *[MWZ98] For each divisor* $D \in \mathbb{D}^0$ *there exists a unique reduced divisor* D_1 *such that* $D \sim D_1$*.*

A proof for the theorem presented above can be found in [MWZ98].

2.8 Representation of Divisors

In Section 2.7, a unique representation of all elements in the Jacobian group was introduced with the concept of divisors. Mumford presented an alternative representation of the divisors, that is more efficient for practical applications [Mum84].

Semi-reduced divisors can be described as a pair of polynomials as stated in the following theorem.

Lemma 2.8.1 *[MWZ98] Let $P = (x, y)$ be an ordinary point of $\mathcal{C}$. Then for each $k \geq 1$, there exists a unique polynomial $b_k(u) \in \overline{\mathbb{F}}[u]$ such that*

1. *$deg_u b_k < k$;*
2. *$b_k(x) = y$; and*
3. *$b_k^2(u) + b_k(u)h(u) \equiv f(u) (\mathrm{mod}(u - x)^k)$.*

Theorem 2.8.2 *[MWZ98] Let $D = \sum m_i P_i - (\sum m_i)\infty$ be a semi-reduced divisor, where $P_i = (x_i, y_i)$. Let $a(u) = \prod (u - x_i)^{m_i}$. There exists a unique polynomial $b(u)$ satisfying:*

1) $\deg_u b < \deg_u a$;

2) $b(x_i) = y_i$ for all i for which $m_i \neq 0$;

3) $a(u)$ divides $b(u)^2 + b(u)h(u) - f(u)$.

Notation: Then $D = \gcd(div(a(u)), div(b(u) - v))$. Usually $\gcd(div(a(u)), div(b(u) - v))$ *will be abbreviated to $div(a(u), b(u) - v)$ or, more simply, to $div(a, b)$.*

The proofs for the lemmas and the theorem can be found in [MWZ98] and the zero divisor is represented as $div(1, 0)$.

Lemma 2.8.3 *Let $a(u), b(u) \in \overline{\mathbb{F}}[u]$ such that $deg_u b < deg_u a$. Then $a|(b^2 + bh - f)$ if and only if $div(a, b)$ is semi-reduced.*

Now we have an alternative representation for semi-reduced divisors, but as discussed above, each element of $\mathbb{J}$ can be represented uniquely by a reduced divisor. A reduced divisor is a semi-reduced divisor but of degree less than or equal to g. Hence the polynomial a is of degree less than or equal to g.

Example: Consider the hyperelliptic curve

$$\mathcal{C} : v^2 + (u^2 + u)v = u^5 + u^3 + 1$$

of genus $g = 2$ over the finite field $\mathbb{F}_{2^5}$ defined with the primitive polynomial $P(x) = x^5 + x^2 + 1$, and let $P(\alpha) = 0$. Let $P_1 = (\alpha^{30}, 0)$ and $P_2 = (0, 1)$ be two points on the curve. Now compute the polynomial representation of

$$D = P_1 + P_2 - 2\infty = \mathrm{div}(a, b).$$

As shown in Theorem 2.8.2, $a(u)$ is calculated as

$$\begin{aligned} a(u) &= \prod (u - x_i)^{m_i}. \\ \text{It follows that} \quad a(u) &= (u + \alpha^{30})(u + 0) = (u + \alpha^{30})u. \end{aligned}$$

To calculate the polynomial $b(u) = cu + a$ of degree ≤ 1 one has to determine its coefficients such that $b(x_i) = y_i$ for all i for which $m_i \neq 0$. Hence one obtains two equations with two variables:

$$\begin{aligned} b(x_1) &= y_1 = 0 = cx_1 + d = c\alpha^{30} + d \\ b(x_2) &= y_2 = 1 = cx_2 + d = c \cdot 0 + d. \end{aligned}$$

The second equation yields $d = 1$. In the next step compute the inverse of

α^{30} modulo the primitive polynomial $P(x) = x^5 + x^2 + 1$:

$$[\alpha^{30}]^{-1} = \alpha \bmod x^5 + x^2 + 1.$$

With the knowledge of the inverse, it is easy to find $c = \alpha$. Hence $b(u) = \alpha u + 1$. With Theorem 2.8.2 the polynomial representation of the given semi-reduced divisor is:

$$\mathrm{div}(a, b) = (u^2 + \alpha^{30}u, \alpha u + 1).$$

It can be seen that the degree of $a(u)$ is equal to g and therefore the polynomial representation found is also the representation for the reduced divisor.

For the setup of a hyperelliptic curve cryptosystem we need to generate random divisors with the properties given above. A method to generate a random divisor is given in [Kob89] and summarized in [Pel02]. In the next sections we introduce the group operation for HECC.

2.9 Cantor's Group Operations on the Jacobian

This section gives a brief description of the algorithms used for adding and doubling divisors on the Jacobian introduced by Cantor [Can87] and given for even characteristic in [Kob89]. Let $\mathcal{C}$ be a hyperelliptic curve of genus g defined over a finite field $\mathbb{F}$, and let $\mathbb{J}$ be the Jacobian of $\mathcal{C}$. Let $P = (x, y) \in \mathcal{C}$, and let σ be an automorphism of $\overline{\mathbb{F}}$ over $\mathbb{F}$. Then $P^\sigma \stackrel{def}{=} (\sigma(x), \sigma(y))$ is also a point on $\mathcal{C}$.

Definition 2.9.1 *A divisor $D = \sum m_P P$ is said to be defined over $\mathbb{F}$ if $D^\sigma \stackrel{def}{=} \sum m_P P^\sigma$ is equal to D for all automorphisms σ of $\overline{\mathbb{F}}$ over $\mathbb{F}$.*

Algorithm 1 describes the group law. The group operation is performed in two steps. First we have to find a semi-reduced divisor $D' = \mathrm{div}(u', v')$, such that $D' \sim D_1 + D_2 = \mathrm{div}(u_1, v_1) + \mathrm{div}(u_2, v_2)$ in the group $\mathbb{J}$ (see Algorithm 1, Step 1 to Step 3). In the second step we have to reduce the semi-reduced divisor $D' = \mathrm{div}\ (u', v')$ to an equivalent divisor $D = (u, v)$ (see Algorithm 1, Step 4 to Step 9). Figure 2.5 The addition is visualized for a genus-2 curves "geometrically".

Algorithm 1 Group addition

Require: $D_1 = \mathrm{div}(u_1, v_1)$, $D_2 = \mathrm{div}(u_2, v_2)$
Ensure: $D = \mathrm{div}(u_3, v_3) = D_1 + D_2$
1: $d = \gcd(u_1, u_2, v_1 + v_2 + h) = s_1 u_1 + s_2 u_2 + s_3(v_1 + v_2 + h)$
2: $u'_0 = u_1 u_2 / d^2$
3: $v'_0 \equiv [s_1 u_1 v_2 + s_2 u_2 v_1 + s_3(v_1 v_2 + f)] d^{-1} (\mathrm{mod} u')$
4: $k = 1$
5: **while** $\deg u'_k > g$ **do**
6: $\quad k = k + 1$
7: $\quad u'_k = \frac{f - v'_{k-1} h - (v'_{k-1})^2}{u'_{k-1}}$
8: $\quad v'_k \equiv (-h - v'_{k-1}) \bmod u'_k$
9: **end while**
10: Output $(u_3 = u'_k, v_3 = v'_k)$

The following two theorems show that Algorithm 1 works and the proofs can be found in [MWZ98]. Theorem 2.9.2 provides the correctness of the composition part and Theorem 2.9.3 of the reduction part.

Theorem 2.9.2 *[MWZ98] Let $D_1 = div(u_1, v_1)$ and $D_2 = div(u_2, v_2)$ be semi-reduced divisors. Let u and v be defined as in Step 2 and 3 of Algorithm 1. Then $D = div(u, v)$ is a semi-reduced divisor and $D \sim D_1 + D_2$.*

Theorem 2.9.3 *[MWZ98] Let $D = div(u, v)$ be a semi-reduced divisor. Then the divisor $D_3 = div(u_3, v_3)$ returned by Algorithm 1 is reduced and $D_3 \sim D$.*

Doubling a divisor is easier than general addition and therefore, Steps 1, 2, and 3 of Algorithm 1 can be simplified as follows:

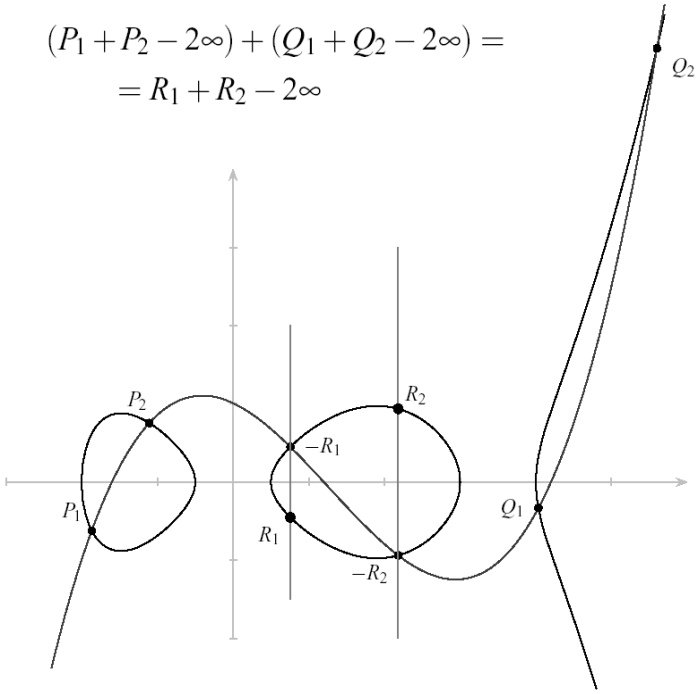


Figure 2.5: Geometrical visualization of the addition on genus 2 HEC.

1: $d = \gcd(u, 2v + h) = s_1 u + s_3(2v + h)$

2: $u' = u^2/d^2$

3: $v' = [s_1 uv + s_3(v^2 + f)]d^{-1} (\mathrm{mod}\, u')$

2.10 Harley's Group Operations on a Jacobian

In [GH00], the authors noticed that one can reduce the number of operations by distinguishing between possible cases according to the properties of the input divisors. They described an efficient algorithm (using Karatsuba multiplication, CRT, and Newton Iteration) to reduce the overall complexity of the group operations.

To determine explicit formulae, it is essential to know the weight of the input divisor. The weight of a divisor is defined as the number of finite points in the support of D.

For example, in the case of a hyperelliptic curve of genus 2, a divisor can have weight 0, 1, or 2. For each case, implementations of different explicit formulae are required. In Figure 2.6, the main flow of the doubling algorithm of a genus-2 HEC is illustrated.

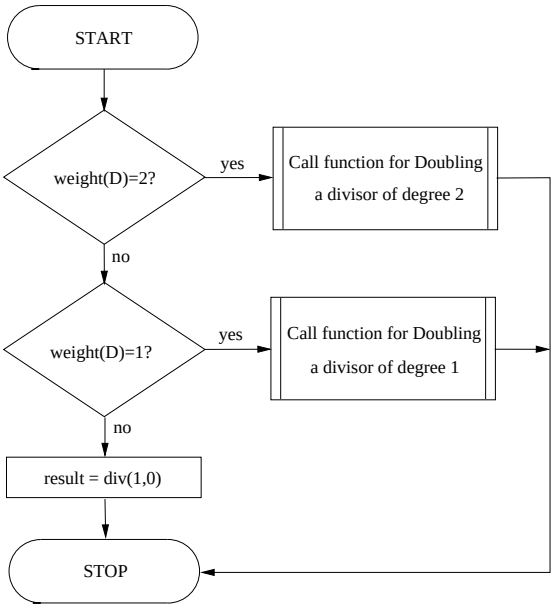


Figure 2.6: Main flow of Harley's doubling algorithm.

Two random polynomials of small degree have a linear factor in common with probability of $\approx 1/q$. Hence, in the case of a hyperelliptic curve of genus 2, we have no factor in common with probability $P \approx 1 - 2^{-80}$. Therefore, in most cases the $\gcd(u_1, u_2) = 1$. For the remainder of this work, we call this the *frequent case*. An implementation based on explicit formulae can be realized by avoiding the non frequent cases. This can be done on basis of a protocol which restarts if a non frequent case occurs. We now restrict to genus two curves over fields of odd characteristic for the moment. Therefore we may assume $h = 0$.

2.10.1 The Frequent Case of Doubling

The frequent case of doubling a divisor occurs if and only if $D = (u_1, v_1)$ *is of weight 2 and* $gcd(u_1, v_1) = 1$*, i.e., both* u_1 *and* v_1 *do* not *have a factor in common. This means*

that D consists of two distinct ordinary points.

According to Cantor's algorithm, $u' = u_1^2$ (see Algorithm 1). The v' polynomial can be calculated using the property $u_1(x)|(v_1(x)^2 - f(x))$ and therefore $u'(x)|(v'(x)^2 - f(x))$. Since $v' \equiv v_1 \bmod u_1$, $v'(x)$ can be seen as a square root of $f(x)$ modulo $u_1^2(x)$ and $v_1(x)$ as a square root of $f(x)$ modulo $u_1(x)$. Hence, we can obtain $v'(x)$ by performing one step of the Newton Iteration [GH00]:

$$v' \equiv v_1 - \frac{f - v_1^2}{2v_1} \bmod u'. \tag{2.2}$$

In order to obtain a unique representation of D', we need to reduce the polynomials $u'(x)$ and $v'(x)$. The polynomials of the reduced divisor $D_2 = (u_2, v_2)$ are

$$\begin{aligned} u_2 &= \frac{f - (v')^2}{u'} \\ u_2 &= \text{monic}(u_2) \\ v_2 &\equiv -v' \bmod u_2. \end{aligned} \tag{2.3}$$

Equations (2.2) and (2.3) can be computed efficiently using following substitutions proposed by Harley:

$$\begin{aligned} k &= \frac{f - v_1^2}{u_1} \\ s &= \frac{k}{2v_1} \bmod u. \end{aligned}$$

Then Equation (2.2) simplifies to $v' = v_1 - u_1 s$ and Equation (2.3) can be rewritten:

$$\begin{aligned} u_2 &= \frac{f - (v')^2}{u'} \\ &= \frac{(v_1^2 - f) + 2su_1v_1 + s^2u^2}{u_1^2} \\ &= s^2 - \frac{k - 2sv_1}{u_1}. \end{aligned}$$

Algorithm 2 combines all steps of the most frequent case for arbitrary characteristic including all subexpressions and uses $h = 0$ [Lan02a].

Algorithm 2 Frequent Case for Group Doubling ($g = 2$).

Require: $D_1 = \mathrm{div}(u_1, v_1)$
Ensure: $D_2 = \mathrm{div}(u_2, v_2) = 2D_1$
1: $k = \frac{v_1^2 - v_1 h - f}{u_1}$ (exact division)
2: $s \equiv \frac{k}{h + 2v_1} \bmod u_1$
3: $u' = s^2 + \frac{k - s(h + 2v_1)}{u_1}$ (exact division)
4: $u_2 = u'$ made monic
5: $v_2 \equiv -(h + su_1 + v_1) \bmod u_2$

In the case of genus-3 and genus-4 curves an extra reduction step is necessary to obtain a reduced divisor. Hence, for genus-3 and genus-4 curves, Steps 6 and 7 have to be added to Algorithm 2 in order to end up with a reduced divisor $D_3 = div(u_3, v_3) = 2D_1$.

$$\begin{aligned} 6: &\quad u_3 = \frac{f - v_2 h - (v_2)^2}{u_2} \quad \text{(exact division)} \\ 7: &\quad v_3 \equiv -(v_2 + h) \bmod u_3 \end{aligned}$$

2.10.2 The Frequent Case of Adding

The frequent case of adding two divisors occurs if and only if $D_1 = (u_1, v_1)$ *and* $D_2 = (u_2, v_2)$ *are both of weight 2 and* $gcd(u_1, u_2) = 1$, *i.e. both* u_1 *and* u_2 *do* not *have a factor*

in common, meaning the support of D_1 does not contain any point or opposite of a point in the support of D_2.

In Algorithm 1, u' is given as $u' = u_1 u_2$ (Step 2). Since in the frequent case $d = d_1 = 1$, $s_1 = e_1$, $s_2 = e_2$ and $s_3 = 0$, v' is obtained by $v' \equiv (e_1 u_1 v_2 + e_2 u_2 v_1) \bmod u'$ (see Algorithm 1, Step 3). Applying Garner's Algorithm for the Chinese Remainder Theorem [MvOV97] results in:

$$v' \equiv \Big[\Big(\frac{v_2 - v_1}{u_1} \bmod u_2\Big) u_1 + v_1\Big] \bmod u'.$$

Since $D' = (u', v')$ is a semi-reduced divisor, a reduction is necessary. Harley performs a reduction which is optimized by reusing some pre-computed quantities. The reduction of u' can be written as

$$\begin{aligned} u' &= \frac{f - (v')^2}{u'} && (2.4)\\ &= \frac{1}{u_1 u_2}\Big\{f - \Big[(\frac{v_2 - v_1}{u_1} \bmod u_2) u_1\Big]^2 - 2 v_1 u_1 (\frac{v_2 - v_1}{u_1} \bmod u_2) - v_1^2\Big\} && (2.5)\\ &= \frac{1}{u_2}\Big\{\frac{f - v_1^2}{u_1} - (\frac{v_2 - v_1}{u_1} \bmod u_2)\Big[(\frac{v_2 - v_1}{u_1} \bmod u_2) u_1 + 2 v_1\Big]\Big\} && (2.6) \end{aligned}$$

Equation 2.6 can be simplified and computed efficiently using the following substitutions proposed by Harley:

$$\begin{aligned} k &= \frac{f - v_1^2}{u_1} && (2.7)\\ s &\equiv \frac{v_2 - v_1}{u_1} \bmod u_2 && (2.8)\\ z &= s u_1 && (2.9) \end{aligned}$$

Equation 2.6 can now be rewritten:

$$u' = \frac{k - s(z + 2v_1)}{u_2}$$

In the next step, u' is made monic and v' is reduced, resulting in a reduced divisor $D_3 = D_1 + D_2 = (u_3, v_3)$. All steps for arbitrary characteristic including those for the precomputed quantities and considering $h = 0$ are given in Algorithm 3 [Lan02a].

Algorithm 3 Frequent Case for Group Addition ($g = 2$).

Require: $D_1 = \text{div}(u_1, v_1)$, $D_2 = \text{div}(u_2, v_2)$
Ensure: $D_3 = \text{div}(u_3, v_3) = D_1 + D_2$
1: $k = \frac{f - v_1 h - v_1^2}{u_1}$ (exact division)
2: $s \equiv \frac{v_2 - v_1}{u_1} \bmod u_2$
3: $z = su_1$
4: $u' = \frac{k - s(z + h + 2v_1)}{u_2}$ (exact division)
5: $u_3 = u'$ made monic
6: $v_3 \equiv -(h + z + v_1) \bmod u_3$

For genus-3 and genus-4 curves an additional reduction step is necessary to calculate the reduced divisor $D_4 = div(u_4, v_4) = D_1 + D_2$:

$$6: \quad u_4 = \frac{f - v_3 h - (v_3)^2}{u_3} \quad \text{(exact division)}$$
$$7: \quad v_4 \equiv -(v_3 + h) \bmod u_4$$

Based on Algorithms 2 and 3, the upcoming explicit formulae for the group operations on hyperelliptic curves of genera 2, 3 and 4 are derived.

2.11 Hyperelliptic Cryptosystems and Security

The Diffie-Hellman problem on $\mathbb{J}$ is closely related to the well-studied discrete logarithm problem (DLP) [MvOV97]. These problems are significant to public-key cryptography

because they form the basis for the security of many cryptographic schemes. The DLP on $\mathbb{J}$ can be stated as follows: given two divisors $D_1, D_2 \in \mathbb{J}$, determine the smallest integer m such that $D_2 = mD_1$, if such an m exists.

A central ingredient in cryptosystems based on the DL problem in an abelian group is an efficient process for computing mD for $D \in \mathbb{J}$ and for large integers m.

$$\underbrace{D + D + \cdots + D}_{m \text{ times}} = mD$$

This operation is called *divisor multiplication* or *scalar multiplication*, and dominates the execution time of hyperelliptic cryptosystems. The binary algorithm and its variants [MvOV97, Gor98] can be used to efficiently compute mD. The main operations in the algorithm are group additions and group doublings.

It is widely accepted that for most cryptographic applications based on EC or HEC one needs a group order of size at least $\approx 2^{160}$. Thus, for HECC over $\mathbb{F}_q$ we will need at least $g \cdot \log_2 q \approx 160$, where g is the genus of the curve. Hence, for a curve of genus two, we will need a field $\mathbb{F}_q$ with $|\mathbb{F}_q| \approx 2^{80}$.

Pollard's rho method and its variants [GLV98, Pol78, Wie86] are the most important examples for algorithms solving the DLP with complexity $O(\sqrt{n})$ in groups of order n. However, some special cases of HEC were discovered in [FR94, Rüc99], which can be attacked with lower complexity than $O(\sqrt{n})$. The first algorithm which computes the DL in subexponential time for sufficiently large genera was published in [ADH94]. The algorithm was improved and implemented, e.g., in [FS97, Eng99a, Gau00a, Gau00b, EG02]. This algorithm has a lower complexity than Pollard's rho method for $g > 4$.

In [FR94], the authors described of the Tate pairing, that is a bilinear map from the divisor class group of a curve $\mathcal{C}$ over a finite field $\mathbb{F}_q$ into the multiplicative group $\mathbb{F}^*_{q^k}$. Hence, for small k, the DLP in the divisor class group can be solved with the

subexponential index-calculus algorithms in $\mathbb{F}_{q^k}^*$. In [Gau00b] it is shown that index-calculus algorithms in the Jacobian of HEC have a lower complexity than the Pollard rho method for curves of genus greater than 4. However, the author further studied special cases by keeping only a fraction of the divisors in the factor base resulting in a reduction of the base by a factor of n. In this case, the author obtained an overall complexity of $O(\frac{q^{2g}}{g+1})$. Thus, the complexity of the attack of genus-4 curves is lower than the complexity of the rho method. However, no practical comparisons between the two approaches have been made.

Recently, Thériault optimized the sub-exponential algorithm to compute the discrete logarithm in the Jacobian of low genus hyperelliptic curves [Thé03]. The underlying field for HEC with genus higher than two might have to be larger than believed in order to achieve a certain security level, see the correction factor in Table 2.1. For the moment, the most efficient attacks presented in [Thé03] are not practical, because of the high storage usage. Note, we took this security consideration into account and enlarge the field sizes accordingly.

Table 2.1: Comparison of the complexity of different attacks on HECC.

g	1	2	3	4
Index calculus	q^2	q^2	q^2	q^2
Rho [Gau00b]	$q^{1/2}$	q	$q^{3/2}$	q^2
reduced factor base [Thé03]	n.a.	n.a.	$q^{3/2}$	$q^{8/5}$
with large primes [Thé03]	n.a.	n.a.	$q^{10/7}$	$q^{14/9}$
Correction factor for $\log_2 \lvert\mathbb{F}_q\rvert$	-	-	1.05	1.286

In order to find secure HECC one has to consider criteria to ensure that a curve is not supersingular as this implies weaknesses under the Frey-Rück attack (see [Gal01]). However, there are no hyperelliptic supersingular curves of genus 2^n-1 and characteristic 2 for any integer $n \geq 2$ [SZ02].

Later in this thesis we present explicit formulae based on special types of curve equa-

tions, resulting in especially fast doubling. For our improvements we use a genus-2 curve of the form $y^2 + xy = x^5 + f_1x + f_0$ where $f_0, f_1 \in \mathbb{F}_{2^n}$. Similar curves where chosen for the genus-3 and genus-4 cases. Obviously, they form a special class of curves but the free choice of f_0 and f_1 still leads to sufficiently many. We are not aware of any security limitation of the curves that we used in this publication.

There is a contribution that dealt with securing practical implementations [Ava03]. In this paper, countermeasures against differential power analysis for HECC were introduced by modeling two techniques used for elliptic curves cryptosystems.

In addition, one should consider the Weil decent attack methodology and especially the GHS Weil decent attack. Consider E to be a non-supersingular elliptic curve defined over a field $K = \mathbb{F}_{2^m}$, and m is composite. The idea of the attack is to reduce the ECDLP in $E(\mathbb{F}_{2^m})$ to the DLP in the jacobian variety of a curve of larger genus defined over a proper subfield $k = \mathbb{F}_l$ of K. The attack was first presented by Frey [Fre98]. In [GHS02], the authors showed how to attack this kind of systems when considering finite fields of characteristic two of composite degree. Concluding from the above mentioned publications, using fields with composite extension degree can have cryptographic weaknesses which can potentially lead to attacks.

By our choices of fields we tried to get a comparison in group size of ECC and HECC as tight as possible, therefore some of the fields we used are based on m composite. However, all implementation techniques *do not use* the composite field structure, unlike the work in [WBV+96, GP97]. Hence, all the software and hardware implementations should be viewed as example cases to show the efficiency of the different systems and are also applicable to HECC based on prime extension fields.

3 Previous Work

In this section, we first summarize previous improvements on group operations of genus-2, genus-3, and genus-4 curves. We start with the work done on speeding up the operations based on Cantor's algorithm, followed by the work done to increase the performance of Harley's algorithm. In the second part we present the previous publications implementing HEC in software and hardware.

In the remainder of the thesis I refers to a field inversion, M to a field multiplication, and S to a field squaring. In some references, the authors did not distinguish between multiplications and squarings. This will be denoted as M/S.

3.1 Previous Improvements of Cantor's Algorithm

Nagao improved the polynomial arithmetic for Cantor's algorithm [Nag00]. He mainly applied the following ideas:

- Division of polynomials without field inversions.
- Computation of the greatest common divisor with one inversion.
- Interleaving superfluous calculations in the reduction part.
- Expressing points on the Jacobian in a different form.

Nagao evaluated the computational cost of the group operations by applying the stated improvements for genus $2 \leq g \leq 10$. The best results for genus two to four HEC are stated in Table 4.2.

Note that in [Nag00] the author estimates the cost of the operations using polynomial arithmetic. We were able to decrease the number of operations needed in all cases by optimizing the explicit notation of these group operations.

3.2 Previous Improvements of Harley's Algorithm

The first work trying to improve the group operation was done from Spallek [Spa94]. She developed explicit formulae for genus-2 curves targeting degree two polynomial and odd characteristic. In [GH00], the authors introduced a slightly different way to reduce the computational effort for the HECC group operations significantly. This was achieved by reducing the number of operations by distinguishing between different cases of the properties of the input divisors. Using the Karatsuba multiplication algorithm [KO63], the Chinese remainder theorem, the Newton iteration, and a reordering of operations, the authors further reduced the overall complexity of the group operations.

In [Lan01], Lange developed for the first time explicit formulae for even characteristic fields and genus-2 curves following the approach from [GH00]. Additionally, in this work the author describing the group operation distinguishing between all the different cases of the properties of the input divisors. For the most frequent case the group addition could be computed using 2 inversion, 24 multiplication, and 3 squarings. The group doubling takes 2 inversion, 26 multiplication, and 6 squarings. However, no implementation was given.

The follow-up implementation based on Harley's algorithm, was done by Matsuo, Chao and Tsuji in [MCT01]. They could save some multiplications compared to Harley's al-

gorithm. A group addition takes 2 inversions and 25 multiplications/squarings and a group doubling takes 2 inversions and 27 multiplications/squarings (see Table 4.3). The authors showed an implementation of both the improved Harley algorithm and the elliptic curve algorithm for comparison. Each one was implemented over some optimal extension fields (OEF): a 93-bit OEF for hyperelliptic curves of genus 2 and a 186-bit OEF for elliptic curves. A Pentium III@886MHz with the GNU C++-2.95.2 compiler was used. A further speed-up for HEC of genus 2 of odd characteristic was achieved in 2002 by Miyamoto, Doi, Matsuo, Chao and Tsuji [MDM+02]. The authors suggested Montgomery's trick of simultaneous inversions to compute two inverses by performing only one field inversion and three field multiplications (for more details see Chapter 4.1). An alternative representation was described in order to avoid inversion. The disadvantage of the alternative representation is the increased amount of multiplications. The new algorithm was implemented on a Pentium III@886MHz. At the same conference, Masashi Takahashi further improved the arithmetic genus-2 curves of odd characteristic [Tak02]. With the help of a small change in the order of a special operation, he could save one multiplication compared to [MDM+02].

The extension of the explicit formulae for arithmetic on genus-2 curves of [MDM+02] and [Tak02] to fields of even characteristic and to arbitrary equations of the curve were done independently from each other and almost at the same time in [SMCT02] and [Lan02a]. In [SMCT02], the authors were able to reduce the number of field operation needed, resulting in 1I + 25M for the group addition. The group doubling was performed in 1I + 27M. The authors used all of the techniques known from the previous publications, like Karatsuba multiplication and Montgomery's multiple inversion technique. In [Lan02a], the author was able to further reduce the complexity of the group operations: I+22M+3S for group addition and I+22M+5S for group doubling. Timings for the implementation of those formulae are also given. Various libraries for

the field arithmetic over prime fields and binary fields on a Pentium IV@1.5GHz were investigated.

In [MDM+02, Lan02b, Lan03], the authors introduced a way to calculate the HEC group operations without using inversions. To reach this aim, at least one coordinate had to be added to represent the elements of the divisor class group. The resulting explicit formulae are advantageous in applications where inversions are expensive compared to multiplication. The authors in [MDM+02] were the first to publish this kind of approach. Their results were improved and generalized to even characteristic in [Lan02b, Lan03]. In [Lan02b], 47 multiplications and 4 squarings are required for a group addition. For the group doubling operation, 40 multiplications and 3 squarings are necessary.

In [Lan02c], the author introduced weighted coordinates on genus-2 curves, resulting in more efficient inversion-free formulae. Furthermore, the paper presents an extensive study about the usage of curves of different characteristic for varying applications. Very recently, [Lan03] gave a thorough comparison of arithmetic on hyperelliptic curves of genus-2 curves containing mainly the material of the previous three papers [Lan02a, Lan02b, Lan02c].

Genus-3 HEC group operations using odd characteristic were improved applying Harley's methodology in [KGM+02]. The authors adopted the methods from [MDM+02,Har00] to increase the performance. The proposed algorithm was implemented on an Alpha Workstation 21264@667MHz using an underlying field $\mathbb{F}_q$, where $q = 2^{61} - 1$. In [GMA+04], the authors introduced very recently an improvement of the genus-3 algorithm. The proposed algorithm takes I + 70M and I + 71M for a group addition and doubling, respectively. The publication introduces two additionally variants algorithms for the group operation in order to adjust the algorithm to various platforms. Furthermore, the authors implemented a 160 bit scalar multiplication on a 64-bit CPU, namely Alpha EV68@1.25GHz.

The work at hand introduces explicit formulae for genus-3 curves of arbitrary characteristic and improves the formulae for the group operation of genus-2 and genus-3 HEC. Furthermore we present explicit formulae for genus-4 curves.

3.3 Previous Software Implementations of HECC

Since HEC cryptosystems were proposed, there have been several software implementations on general purpose machines and, only recently, publications dealing with hardware implementations of HECC.

The results of previous HECC software implementations are summarized in Table 3.1. The table entries are sorted in chronological order. All implementations up to [SS00] use Cantor's algorithm with polynomial arithmetic. Starting with [MCT01], the implementations make use of explicit formulae. The table includes only implementations that are considered to be secure, namely curves of genus smaller than five, and shows only the fastest numbers given in each publication. For example, the implementation presented in [Sma99] is not included in Table 3.1, because it focused only on HECC with genus five or higher.

The authors in [MCT01] could save two multiplications/squarings and three multiplications/squarings in the group addition and doubling operations, respectively, compared to Harley's algorithm. This implementation was followed by [MDM+02, Tak02, Lan02a, Ava04]. In [Ava04], the author did an extensive study of HECC using prime fields. The only genus-3 curve implementations based on the explicit formulae were presented in [KGM+02, Ava04].

In [Ngu02] the author did the first implementation of HECC on an embedded processor namely using the FrameXE smart card coprocessor. To our knowledge the work at hand is the only work providing extensive implementations of HECC on various different

Table 3.1: Execution times of recent HEC implementations in software.

reference	processor	genus	group order	$t_{scalarmult.}$ in ms
[Kri97]	Pentium@100MHz	2	$\approx 2^{128}$	520
		3	$\approx 2^{126}$	1200
		4	$\approx 2^{124}$	1100
[SS98]	Alpha@467MHz	3	$\approx 2^{177}$	83.3
		3	$\approx 2^{267}$	25700
		3	$\approx 2^{339}$	37900
		4	$\approx 2^{164}$	96.6
	Pentium-II@300MHz	3	$\approx 2^{177}$	11700
		4	$\approx 2^{164}$	10900
[SS00]	Alpha21164A@600MHz	3	$\approx 2^{180}$	98
		3	$\approx 2^{177}$	40
		4	$\approx 2^{164}$	43
[MCT01]	PentiumIII@866MHz	2	$\approx 2^{186}$ OEF	1.98
[MDM+02]	PentiumIII@866MHz	2	$\approx 2^{186}$ OEF	1.69
[KGM+02]	Alpha21264@667MHz	3	$\approx 2^{183}$	0.932
[Lan02a]	Pentium-IV@1.5GHz	2	$\approx 2^{160}$	18.875
		2	$\approx 2^{180}$	25.215
		2	$\approx 2^{160}$	5.663
		2	$\approx 2^{180}$	8.162
[GMA+04]	Alpha EV68@1.25GHz	3	$\approx 2^{160}$	0.176
[Ava04]	AMD Athlon@1GHz	2	$\approx 2^{128}$	0.867
		2	$\approx 2^{144}$	1.21
		2	$\approx 2^{160}$	1.41
		2	$\approx 2^{192}$	1.676
		2	$\approx 2^{224}$	3.085
		2	$\approx 2^{256}$	3.528
		2	$\approx 2^{320}$	6.85
		2	$\approx 2^{512}$	23.323
		3	$\approx 2^{128}$	1.604
		3	$\approx 2^{144}$	1.808
		3	$\approx 2^{160}$	2.792
		3	$\approx 2^{192}$	3.538
		3	$\approx 2^{224}$	4.53
		3	$\approx 2^{256}$	5.343
		3	$\approx 2^{320}$	10.28
		3	$\approx 2^{512}$	36.209

embedded systems and analyzing the achieved results.

3.4 Previous Hardware Implementations of HECC

This section gives a short overview of the hardware implementations targeting HECC.

The first work discussing hardware architectures for the implementation of HECC appeared in [Wol01,WP02]. The authors describe efficient architectures to implement the necessary field operations and polynomial arithmetic in hardware. All of the presented architectures are speed and area optimized. In [Wol01], they also estimated that for a hypothetical clock frequency of 20 MHz, the scalar multiplication of HECC would take 21.4 ms using the window NAF method.

In [BCLW02] the authors presented the first complete hardware implementation of a hyperelliptic curve coprocessor on FPGA. This implementation targets a genus-2 HEC over $\mathbb{F}_{2^{113}}$. The target platform is a Xilinx II FPGA. Point addition and point doubling with a clock frequency of 45MHz take 105μs and 90μs, respectively. The scalar multiplication could be computed in 20.2 ms.

In [Cla02, Cla03] the authors presented extended results of [BCLW02]. They implemented a HECC coprocessor using a variety of base fields, ranging from $\mathbb{F}_{2^{83}}$ to $\mathbb{F}_{2^{163}}$, as well as two different multipliers (digit size D=1 and D=4). The scalar multiplication takes between 9ms and 40ms, and uses between 22,000 and 119,000 slices. Analyzing the implementation numbers given in the paper, one notice, that especially the design options using D=4 multipliers have unreasonable hardware requirements.

Note that publications mentioned so far adopt the Cantor algorithm to compute group operations. Today, there exist more efficient algorithms to compute group addition and group doubling, as described in the previous sections.

The first approach to implement hyperelliptic curve cryptosystem in hardware using explicit formulae is presented in [EMY04]. The authors used the inversion-free group operations for HECC introduced in [Lan02b]. The results presented used a field GF(2^{113}), two different methods for the scalar multiplication (Binary Expansion Method and NAF), and two different digit multipliers ($D = 1$ and $D = 4$). They were able to reach a speed of 2.03ms for the best scalar multiplication. More details about the results presented in [EMY04] are given in Table 3.2.

Table 3.2: Previous results of the HEC implementations on FPGA.

		group order	Clock Cycles	Slices	Freq [MHz]	Time [ms]
[Cla03]	affine coord.					
	binary (D=1)	$\approx 2^{166}$	-	22,000	-	10.00
	binary (D=4)	$\approx 2^{166}$	-	60,000	-	9.00
[EMY04]	proj. coord.					
	binary (D=1)	$\approx 2^{226}$	339,057	22,183	45.6	7.53
	NAF (D=1)	$\approx 2^{226}$	332,913	21,550	45.6	7.39
	binary (D=4)	$\approx 2^{226}$	95,286	25,911	46.7	2.12
	NAF (D=1)	$\approx 2^{226}$	91,606	25,271	45.3	2.03

4 Improving the Explicit Formulae

In this chapter we describe our work on improving the group operations for the HECC. Part of the work presented in this section was published in [PWGP03,PWP03,PWP04b, PWP04a]. The ideas of [GH00] were the starting point for our improvements. For more details concerning Harley's algorithm, see Section 2.10.

We will first start to describe the methods that we used to improve the HEC group operations. One of the techniques applies the Karatsuba method to reduce the complexity of polynomial reduction, which we generalized in our work. This will be followed by the presentation of our optimized group operation. We decreased the complexity for Cantor's and Harley's algorithm and the inversion-free group operations. We end this chapter with a short summary of the fastest explicit formulae for HECC to date.

4.1 Methods to Improve the Explicit Formulae

In this section, we list all techniques used to improve the efficiency of the explicit formulae.

4.1.1 Montgomery's Trick of Simultaneous Inversions

The idea of Montgomery is reduce the number of inversions at the cost of some cheaper operations, e.g., multiplications [Coh93, Algorithm 10.3.4], by simultaneous inversion. The idea is to combine two or more inversions into one inversion. Without loss of generality (WLOG) we consider the group addition on genus-2 HECC to illustrate its application.

Harley's Algorithm needs two field inversions, see Algorithm 3, Steps 2 and 5. These steps can be modified. Instead of computing

$$s(x) = s_1 x + s_0 \equiv \frac{v_2 - v_1}{u_1} \bmod u_2,$$

one first computes the resultant r of u_1 and u_2

$$inv \equiv \frac{r}{u_1} \bmod u_2 \quad \text{(no field inversion needed)}$$

and

$$s'(x) = rs \equiv (v_2 - v_1) \cdot inv \bmod u_2.$$

In the latter step only one inversion is necessary to obtain both r^{-1} and s_1^{-1}. The variable s_1^{-1} is required in Algorithm 3 to make u' monic and r^{-1} is required to calculate $s(x) = s'r^{-1}$. First, one has to compute the inverse $w_1 = (rs_1')^{-1}$, then s and r^{-1} can be calculated:

$$r^{-1} = w_1 s_1'$$

$$s(x) = s_1 x + s_0 = \frac{s'}{r} = \frac{s_1'}{r} x + \frac{s_0'}{r}.$$

s_1^{-1} is then obtained by

$$s_1^{-1} = w_1 r^2 = \frac{r}{s_1'}.$$

4.1.2 Reordering of the Normalization Step

Reordering allows one to calculate the required monic polynomial u' (Algorithm 2, Step 4, and Algorithm 3, Step 5) while saving field operations [Tak02]. The highest order term of u' results from the product of s and z divided by u_2. Since u_2 is monic, the leading coefficient of u' is s_1^2. Thus, the step of making u' monic is unnecessary if the polynomial s is already monic (i.e., $s_1 = 1$). In [Tak02], the author shows that this simplification of using S_{monic} instead of s saves one multiplication in total for the case of genus 2. In this contribution, we show that in the case of genus-3 and genus-4 curves one can save even more multiplications. In the following we show, that u' is already monic:

$$\begin{aligned}
\text{let} \quad w_4 &= \frac{1}{s_1} \quad \text{and} \quad w_5 = w_4^2 \\
u' &= \frac{1}{u_2}\Big[s_{monic}(s_{monic}u_1 + w_4(h + 2v_1)) - w_5\frac{f - v_1h - v_1^2}{u_1}\Big] \\
&= \frac{1}{u_2}\Big[sw_4(sw_4u_1 + w_4(h + 2v_1)) - w_5\frac{f - v_1h - v_1^2}{u_1}\Big] \quad \text{with } s_{monic} = sw_4 \\
&= \frac{w_4^2}{u_2}\Big[s(su_1 + h + 2v_1) - \frac{f - v_1h - v_1^2}{u_1}\Big] \\
&= -w_4^2u = -\frac{u}{s_1^2}
\end{aligned}$$

Using the reordering, the calculation of v' changes as follows:

$$\begin{aligned}
\text{with} \quad w_3 = s_1, \quad v' &= -(w_3s_{monic}u_1 + h + v_1) \bmod u' \\
&= -(w_3w_4su_1 + h + v_1) \bmod u' \\
&= -(su_1 + h + v_1) \bmod u'
\end{aligned}$$

4.1.3 Karatsuba Multiplication

In 1962, Karatsuba introduced an algorithm to multiply two polynomials [KO63]. Compared to the schoolbook method, the Karatsuba Algorithm (KA) saves multiplications of the coefficients at the cost of extra additions. Hence, if the ratio of the complexity of a coefficient multiplication and a coefficient addition is higher than a certain value, KA is more efficient. Since its introduction, further work was done to improve the KA and to find bounds of the complexity [Knu81, LSW83, Win77]. In [Ber01], Bernstein presented a survey of different methods to multiply polynomials. In [WP03a], detailed information on the usage of KA in order to multiply with the least cost is provided. Furthermore, a table states the computational cost of the KA for polynomials up to a degree of 127.

In [KO63], the efficient multiplication of two polynomials of degree 1 is introduced. We will briefly develop the KA and refer the interested reader to the given literature. Given two polynomials $A(x) = a_1x + a_0$ and $B(x) = b_1x + b_0$. Let $D_0 = a_0b_0$, $D_1 = a_1b_1$ and $D_{0,1} = (a_0 + a_1)(b_0 + b_1)$. The product of two polynomials can be computed as:

$$C(x) = D_1x^2 + (D_{0,1} - D_0 - D_1)x + D_0$$

The total cost is four additions and three multiplications. Whereas, if using the schoolbook method, four multiplications and one addition are needed. Thus, we save one multiplication at the cost of three extra additions.

4.1.4 Karatsuba Reduction

The technique introduced by Karatsuba can be used to efficiently compute the reduction of two polynomials. We generalized the use of Karatsuba for the reduction of polynomials in Chapter 4.2.

4.1.5 Efficient Division

In [vzGG99], an efficient way to calculate the quotient of two polynomials is presented. The division is based on the observation that the quotient of two polynomials of degree deg_1 and deg_2, with $deg_1 > deg_2$, depends only on the $deg_1 - deg_2 + 1$ highest coefficients of the dividend and the $deg_1 - deg_2 + 1$ highest coefficients of the divisor. Hence, we do not have to consider all coefficients of the polynomials.

This efficient division is, for example, used in Step 8 in Table A.6. One has to compute $u_3 = (f - v'h - v'^2)/u'$ with $deg(f - v'h - v'^2) = 7$ and $deg(u') = 4$. Thus, only the four highest coefficients of the two terms are needed to compute u_3 (Notice, that $f_7 = u'_4 = 1$ and therefore do not appear in the formula).

4.1.6 Calculation of the Resultant Using Bezout's Matrix

The calculation of the resultant using Bezout's matrix in the case of genus-3 HEC can be performed very efficiently. Notice, that there is no benefit when applying the Bezout's matrix to genus-2 and genus-4 HEC.

The resultant of two polynomials $a = \prod_{i=1}^{n}(x - \alpha_i)$ and $b = \prod_{j=1}^{m}(x - \beta_j)$ is defined by $r(a, b) = \prod_{i=1}^{n}\prod_{j=1}^{m}(\beta_j - \alpha_i)$.

Let $a = x^3 + ax^2 + bx + c$ and $b = x^3 + dx^2 + ex + f$, then the resultant calculated with the Bezout's matrix is given by

$$\begin{aligned} r(a,b) &= (f + ea - c - bd)[(-c + f)^2 - (-a + d)(fb - ce)] + (fa - cd) \\ &\quad [(fa - cd)(-a + d) - 2(-b + e)(-c + f)] + (fb - ce)(-b + e)^2. \end{aligned}$$

In general, the total cost of this operation is 12 multiplications and 2 squarings. For two input polynomials in the case of doubling, the resultant is less complex and can be

computed with 6 multiplications and 2 squarings. For more details, see Tables A.6 and A.7.

4.2 Karatsuba Reduction

In our contribution, we used the idea of the algorithm presented by Karatsuba [KO63] to minimize the computational complexity of polynomial modulo reduction, denoted as Karatsuba reduction. In the context of HECC, this idea was first used for genus-2 curves targeting small polynomials in [GH00, KGM+02]. In this work, we generalized the procedure for polynomials of arbitrary degree.

Let the polynomial $A(x) = \sum_{i=0}^{m+d} a_i x^i$ to be reduced by the polynomial $P(x) = x^m + p(x) = x^m + \sum_{i=0}^{m-1} p_i x^i$ of degree m. Recursive application of Equation (4.1) is a simple way to perform modulo reduction whenever encountering powers of x greater than $m-1$.

$$x^m = -p(x) = \sum_{i=0}^{m-1} (-p_i) x^i \tag{4.1}$$

4.2.1 Reduction with Degree Difference One

In this subsection, we will derive an algorithm to reduce a polynomial $A(x)$ with degree $m+1$, where m is the degree of the reduction polynomial. If using the method described, one can save one multiplication at the extra cost of three additions. The following example will sketch the idea of the Karatsuba reduction.

Example: *Let, the polynomial $A(x)$ be of degree 3 and let $P(x)$ be monic of degree 2.*

$$\begin{aligned} B(x) &= A(x) \bmod P(x) \\ \sum_{i=0}^{1} b_i x^i &= \sum_{i=0}^{3} a_i x^i \bmod (x^2 + \sum_{i=0}^{1} p_i x^i) \end{aligned}$$

In the first step, $x^3 = -x(p_1 x + p_0) = -p_1 x^2 - p_0 x$ can be substituted in the polynomial $A(x)$ and we set $a_3 = T_0$. Hence,

$$B(x) = [a_2 - T_0 p_1]x^2 + [a_1 - T_0 p_0]x + a_0 \bmod P(x). \tag{4.2}$$

In the second step, $x^2 = -p_1 x - p_0$ is substituted in Equation (4.2) and we replace $T_1 = a_2 - T_0 p_1$.

$$B(x) = [a_1 - T_0 p_0 - T_1 p_1]x + [a_0 - T_1 p_0] \bmod P(x) \tag{4.3}$$

In Equation (4.3), we can apply Karatsuba in order to save one multiplication, as we have already computed $T_0 p_1$.

$$B(x) = [a_1 - (T_0 + T_1)(p_0 + p_1) + T_0 p_1 + T_1 p_0]x + [a_0 - T_1 p_0] \bmod P(x) \tag{4.4}$$

In Equation (4.3), the total cost of the polynomial reduction without using Karatsuba reduction is 4 multiplications and 4 additions (1M and 1A to calculate T_1; 3M and 3A in Equation (4.3)). In Equation (4.4), after applying Karatsuba reduction, we need 3 multiplications and 7 additions (1M and 1A to calculate T_1; 2M and 6A in Equation (4.4)).

Now, consider an algorithm to reduce polynomials with $m + 1$ coefficients, where m is any positive integer and the degree of the monic reduction polynomial $P(s)$.

$$B(x) = \begin{cases} \sum_{l=1}^{(m-1)/2}[(-T_1p_{2l} - T_0p_{2l-1} + a_{2l})x - T_1p_{2l-1} \\ -T_0p_{2l-2} + a_{2l-1}]x^{2l-1} - T_1p_0 + a_0 & \text{for odd m} \\ (-T_1p_{m-1} - T_0p_{m-2} + a_{m-1})x^{m-1} + \sum_{l=1}^{m/2-1}[(-T_1p_{2l} - T_0p_{2l-1} \\ +a_{2l})x - T_1p_{2l-1} - T_0p_{2l-2} + a_{2l-1}]x^{2l-1} - T_1p_0 + a_0 & \text{for even m} \end{cases} \tag{4.6}$$

$$= \begin{cases} \sum_{l=1}^{(m-1)/2}\Big[(-T_1p_{2l} - T_0p_{2l-1} + a_{2l})x - (T_1 + T_0)(p_{2l-1} + p_{2l-2}) \\ + T_1p_{2l-2} + T_0p_{2l-1} + a_{2l-1}\Big]x^{2l-1} - T_1p_0 + a_0 & \text{for odd m} \\ \Big[-(T_1 + T_0)(p_{m-1} + p_{m-2}) + T_1p_{m-2} + T_0p_{m-1} + a_{m-1}\Big]x^{m-1} \\ + \sum_{l=1}^{m/2-1}[(-T_1p_{2l} - T_0p_{2l-1} + a_{2l})x - (T_1 + T_0)(p_{2l-1} + p_{2l-2}) \\ +T_1p_{2l-2} + T_0p_{2l-1} + a_{2l-1}]x^{2l-1} - T_1p_0 + a_0 & \text{for even m} \end{cases} \tag{4.7}$$

Compute

$$\begin{aligned} B(x) &= A(x) \bmod P(x) \\ \sum_{i=0}^{m-1} b_i x^i &= \sum_{i=0}^{m+1} a_i x^i \bmod (x^m + \sum_{i=0}^{m-1} p_i x^i). \end{aligned}$$

After substituting $x^{m+1} = -\sum_{i=0}^{m-1} p_i x^{i+1}$ and $x^m = -\sum_{i=0}^{m-1} p_i x^i$ we get

$$\begin{aligned} B(x) &= a_{m+1}\Big[p_{m-1}p_0 + \sum_{i=1}^{m-1} x^i(p_{m-1}p_i - p_{i-1})\Big] + a_m\Big[-\sum_{i=0}^{m-1} p_i x^i\Big] + \sum_{i=0}^{m-1} a_i x^i \\ &= \sum_{i=1}^{m-1}\Big[p_i(a_{m+1}p_{m-1} - a_m) - a_{m+1}p_{i-1} + a_i\Big]x^i + p_0(a_{m+1}p_{m-1} - a_m) + a_0. \end{aligned}$$

For the sake of simplicity we set the leading coefficients $T_0 := a_{m+1}$ and $T_1 := a_m - a_{m+1}p_{m-1}$.

$$B(x) = \sum_{i=1}^{m-1}\Big[-p_iT_1 - T_0p_{i-1} + a_i\Big]x^i - p_0T_1 + a_0 \tag{4.5}$$

We have to consider two cases — m odd and even — in order to be able to apply Karatsuba reduction for the reduction of $A(x)$ in Equation (4.6). The result is shown in Equation (4.7).

Table 4.1: Cost of standard polynomial reduction versus Karatsuba reduction.

m	standard	Karatsuba	$t_{mult}/$ t_{add}
2	$4M + 4A$	$3M + 7A$	3
3	$6M + 6A$	$5M + 9A$	3
4	$8M + 8A$	$6M + 14A$	3
5	$10M + 10A$	$8M + 16A$	3
6	$12M + 12A$	$9M + 21A$	3
7	$14M + 14A$	$11M + 23A$	3
8	$16M + 16A$	$12M + 28A$	3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
m	$2mM + 2mA$	$\lceil 3/2m \rceil M + (8m - 3\lceil 3/2m \rceil)A$	3

For m odd one has to perform $(m-1)/2$ times $3M+7A$ and additionally $2M+2A$. Each partial addend comprises the following three multiplications: $(T_1 + T_0)(p_{2l-1} + p_{2l-2})$, $T_0 p_{2l-1}$, and $T_1 p_{2l}$. Note, that $T_0 p_{2l-1}$ occurs twice but has to be calculated only once and that $T_1 p_{2l-2}$ is known from the previous step. In addition, two multiplications $a_{m+1} p_{m-1}$ and $T_1 p_0$ have to be computed.

For m even, one has to provide in total $m/2$ times $3M$ and $7A$. Hence, one more multiplication is necessary: $(T_1 + T_0)(p_{m-1} + p_{m-2})$. $T_1 p_{m-2}$ was already calculated in an earlier step and $T_0 p_{m-1}$ was computed in order to get T_1.

Corollary 4.2.1 *Let $\#M$ and $\#A$ be the number of multiplications and additions, respectively, required to reduce a polynomial with degree $m+1$ modulo a reduction polynomial of degree m. Using Karatsuba reduction as shown in Equation (4.7), we need*

$$\lceil 3/2m \rceil M + (8m - 3\lceil 3/2m \rceil)A. \tag{4.8}$$

In the case of the schoolbook method, as it can be seen in Equation (4.5), one needs to perform $(m-1)(2M + 2A) + 2M + 2A = m(2M + 2A)$ operations. Table 4.1 shows a comparison of the Karatsuba reduction and the schoolbook method. From the table,

it can be seen that the ratio between the execution time of multiplication and addition is always 3. Therefore, Karatsuba reduction is more efficient if the execution time for one coefficient multiplication is higher than the time to perform three additions, which is the case for most software and hardware implementations.

The complete calculation using Karatsuba reduction with degree difference one is merged in Algorithm 4.

Algorithm 4 Karatsuba reduction (degree difference $d = 1$).

Require:
Polynomial $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i$ of degree m,
Polynomial $A(x) = \sum_{i=0}^{m+1} a_i x^i$ of degree $m+1$
Ensure: $B(x) = A(x) \bmod P(x) = \sum_{i=0}^{m-1} b_i x^i$

1: $T_0 = a_{m+1}$
2: $T_1 = a_m - T_0 p_{m-1}$
3: $b_0 = a_0 - T_1 p_0$
4: **for** $l = 1$ to $\lfloor \frac{m-1}{2} \rfloor$ **do**
5: $\quad b_{2l-1} = a_{2l-1} - (T_0 + T_1)(p_{2l-1} + p_{2l-2}) + T_1 p_{2l-2} - T_0 p_{2l-1}$
6: $\quad b_{2l} = a_{2l} - T_1 p_{2l} - T_0 p_{2l-1}$
7: **end for**
8: **if** m even **then**
9: $\quad b_{m-1} = a_{m-1} - (T_0 + T_1)(p_{m-1} + p_{m-2}) + T_1 p_{m-2} + T_0 p_{m-1}$
10: **end if**
11: Output $B(x) = \sum_{i=0}^{m-1} b_i x^i$

4.2.2 Reduction with Arbitrary Degree Difference

In this section we generalize the results from Section 4.2.1 to arbitrary degree difference d of the polynomial to be reduced and the reduction polynomial. Let

$$B(x) = A(x) \bmod P(x) = \sum_{i=0}^{m+d} a_i x^i \bmod x^m + \sum_{i=0}^{m-1} p_i x^i \tag{4.9}$$

with an integer $d \geq 1$.

We reduce the polynomial $\sum_{i=0}^{m+d} a_i x^i$ by recursively inserting $x^{m+j} = -x^j \sum_{i=0}^{m-1} p_i x^i$

$$
\begin{aligned}
B(x) &= \sum_{l=\frac{d+1}{2}}^{\frac{m+d-2}{2}} \big\{[a_{2l} - T_0 p_{2l-d} - T_1 p_{2l-d+1}]x + [a_{2l-1} - T_0 p_{2l-1-d} - T_1 p_{2l-d}]\big\} x^{2l-1} \\
&\quad + (a_{d-1} - T_1 p_0) x^{d-1} + \sum_{i=0}^{d-2} a_i x^i \qquad (4.12) \\
&= \sum_{l=\frac{d+1}{2}}^{\frac{m+d-2}{2}} \big\{[a_{2l} - (T_0 + T_1)(p_{2l-d} + p_{2l-d+1}) + T_0 p_{2l-d+1} + T_1 p_{2l-d}]x \\
&\quad + [a_{2l-1} - T_0 p_{2l-1-d} - T_1 p_{2l-d}]\big\} x^{2l-1} + (a_{d-1} - T_1 p_0) x^{d-1} + \sum_{i=0}^{d-2} a_i x^i \qquad (4.13)
\end{aligned}
$$

---

for $0 \leq j \leq d$. The reduction for arbitrary degree difference consists of two substitution steps, that are repeatedly computed $d/2$ or $(d-1)/2$ times for d even or odd, respectively. In the following we describe these steps in more detail and summarize them in Algorithm 5.

STEP 1: We substitute $x^{m+d} = -x^d \sum_{i=0}^{m-1} p_i x^i$ and $T_0 = a_{m+d}$ into Equation (4.9) :

$$
B(x) = (a_{m+d-1} - T_0 p_{m-1}) x^{m+d-1} + \sum_{i=-d}^{m+d-2} (a_i - T_0 p_{i-d}) x^i + \sum_{i=0}^{d-1} a_i x^i \qquad (4.10)
$$

STEP 2: We substitute $x^{m+d-1} = -x^{d-1} \sum_{i=0}^{m-1} p_i x^i$ and $T_1 = (a_{m+d-1} - a_{m+d} p_{m-1})$ into Equation 4.10:

$$
B(x) = \sum_{i=d}^{m+d-2} (a_i - T_0 p_{i-d} - T_1 p_{i-d+1}) x^i + (a_{d-1} - T_1 p_0) x^{d-1} + \sum_{i=0}^{d-2} a_i x^i \qquad (4.11)
$$

Next, we assume WLOG m and d to be odd. We rewrite Equation (4.11) in order to be able to apply Karatsuba reduction on Equation (4.12). The result is given in Equation (4.13).

After the second step, we have a polynomial $B(x)$ of degree $m+d-2$. Thus, we are able

to reduce the degree of the polynomial by 2. Step 1 and Step 2 have to be repeated $d/2$ or $(d+1)/2$ times in order to reduce $A(x)$ for d odd or even, respectively. After each iteration of Step 1 and 2 we set $A(x)_i = B(x)_{i-1}$. Algorithm 5 summarizes this iteration of Step 1 and 2.

Algorithm 5 Karatsuba reduction (arbitrary degree difference).

Require:
Polynomial $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i$ of degree m,
Polynomial $B^{(d)}(x) = \sum_{i=0}^{m+d} b_i^{(d)} x^i$ of degree $m+d$
Ensure: $B(x) = B^{(d)}(x) \bmod P(x) = \sum_{i=0}^{m-1} b_i x^i$

while $d > 0$ **do**
  $T_0 = b_{m+d}^{(d)}$
  $T_1 = b_{m+d-1}^{(d)} - T_0 p_{m-1}$
  $b_{d-1}^{(d-2)} = b_{d-1}^{(d)} - T_1 p_0$
  for $l = \lceil \frac{d+1}{2} \rceil$ to $\lfloor \frac{m+d-2}{2} \rfloor$ **do**
    $b_{2l}^{(d-2)} = b_{2l}^{(d)} - (T_0 + T_1)(p_{2l-d} + p_{2l-d+1}) + T_0 p_{2l-d+1} - T_1 p_{2l-d}$
    $b_{2l-1}^{(d-2)} = b_{2l-1}^{(d)} - T_0 p_{2l-d-1} - T_1 p_{2l-d}$
  end for
  if m even **then**
    $b_{m+d-2}^{(d-2)} = -(T_0 + T_1)(p_{m-1} + p_{m-2}) + T_1 p_{m-2} + T_0 p_{m-1} + b_{m+d-2}^{(d)}$
  end if
  for $j = 0$ to $d-2$ **do**
    $b_j^{(d-2)} = b_j^{(d)}$
  end for
  $d = d - 2$
end while
if $d = 0$ **then**
  for $j = 0$ to $m-1$ **do**
    $b_j^{(-1)} = b_j^{(d)} - b_m^{(d)} \cdot p_j$
  end for
end if
Output $B(x) = \sum_{j=0}^{m-1} b_j x^i = B^{(-1)}(x)$

Corollary 4.2.2 *Let $\#M$ and $\#A$ be the number of multiplications and additions, respectively, to reduce a polynomial of degree $m+d$, where d is a positive integer and where m is the degree of the reduction polynomial. Thus, we need to perform $d+1$ reduction steps in order to replace all terms with occurrences of x^k where $k \geq m$. In*

Equation (4.13) the cost of the two successive reduction steps is reduced by combining the calculation of the coefficients using Karatsuba reduction. The cost of each "double-step" is equivalent to the cost described in Corollary 4.2.1. Thus, the total cost is as stated in Equation (4.14).

$$Cost \begin{cases} \lfloor \frac{d+1}{2} \rfloor \Big[\lceil 3/2m \rceil M + (8m - 3\lceil 3/2m \rceil)A \Big] & \text{for odd } d \\ \lfloor \frac{d+1}{2} \rfloor \Big[\lceil 3/2m \rceil M + (8m - 3\lceil 3/2m \rceil)A \Big] + mM + mA & \text{for even } d \end{cases} \tag{4.14}$$

In the case of even d, we have to add an additional step after applying Karatsuba. This last step reduces the polynomial with degree m, by substituting $x^m = \sum_{i=0}^{m-1}(-p_i)x^i$. Hence, we need m additional multiplications and m additions. Applying Karatsuba reduction to the additional step does not result in a more efficient way to reduce the polynomial of degree m.

If we use the schoolbook method instead, we need m multiplications and m additions for each reduction step, thus $m(d+1)[M+A]$ in total.

4.2.3 Optimal Reduction Polynomials

In Subsections 4.2.1 and 4.2.2, we considered only reduction polynomials with all coefficients non-zero. In most cases, reduction polynomials will have only some non-zero coefficients. In this case Corollary 4.2.3 should be considered.

Corollary 4.2.3 *Let $\#M$ and $\#A$ be the number of multiplications and additions, respectively, to reduce a polynomial $A(x)$ with degree $m+d$, where d is a positive integer. The reduction polynomial is given by $P(x) = x^m + x^{t_1} + x^{t_2} + x^{t_3} + \cdots + x^{t_i}$, where $i+1$ is the number of non-zero coefficients. Hence, the cost to reduce the polynomial $A(x)$ is*

given in Equation (4.15).

$$Cost \begin{cases} \lfloor \frac{d+1}{2} \rfloor \left[\lceil 3/2i \rceil M + (8i - 3\lceil 3/2i \rceil)A \right] & \text{for odd } d \\ \lfloor \frac{d+1}{2} \rfloor \left[\lceil 3/2i \rceil M + (8i - 3\lceil 3/2i \rceil)A \right] + iM + iA & \text{for even } d \end{cases} \tag{4.15}$$

Corollary 4.2.4 *In case of schoolbook or Karatsuba reduction, a polynomial with the smallest possible number of non-zero coefficients results in a minimum of computational cost.*

4.3 Improving the Group Operation

After Koblitz suggested HEC for the use for a cryptosystem in 1989, it took over 10 years until the first improvements of the group operation were published. In [GH00], the authors proposed explicit formulae for the group operations. For the deviation of the explicit formulae, polynomial calculations are mapped onto field operations. For genus-2 HEC, such a mapping can be calculated quite easily, since the degree of the occurring polynomials is low. Deriving explicit formulae for higher genus HEC is more complicated since with increasing genus the degree of the polynomials rises. For higher genera, the problem becomes intractable by hand and the use of additional tools for the mathematical evaluation of all equations is required.

Choice of HEC with Certain Properties: A detailed analysis of the explicit formulae gives rise to certain types of curves with low complexity, i.e., optimal performance regarding the number of required field operations for the execution of the group operations. As a result of this analysis, curves of the form $y^2 + y = f(x)$ over extension fields of characteristic two turn out to be the best option. Unfortunately, this type of curve is supersingular for genus 2 and genus 4 [Gau00b, Gal01]. To our knowledge, curves of this form for genus 3 have no security limitations [SZ02]. Hence, curves of the form

$y^2 + h(x)y = f(x)$, with $h(x) = x$ give the best performance for genus 2 and genus 4.

According to [Lan03], for genus-2 curves, even characteristic, and $\deg[h(x)] = 1$, we can achieve $f_3 = f_4 = 0$. With the substitution $x \rightarrow x - f_4/5$ we obtain a curve where $f_4 = 0$. $y \rightarrow y + h_1^{-1} f_3 x^2$ provides $f_3 = 0$. In addition, if we can find a b such that $f_3 h_0 + b^2 h_1 + b h_1^2 = f_2 h_1^{-1}$ has a solution, we can achieve $f_2 = 0$ by substituting $y \rightarrow y + h_1 f_3 x^2 + bx$. For our improvements we assume genus-2 curves of the form $y^2 + xy = x^5 + f_1 x + f_0$ where $f_0, f_1 \in \mathbb{F}_{2^n}$. Similarly, one can achieve efficient curve equations for genus-3 $y^2 + xy = x^7 + f_1 x + f_0$ where $f_0, f_1 \in \mathbb{F}_{2^n}$ and genus-4 HEC $y^2 + xy = x^9 + f_1 x + f_0$ where $f_0, f_1 \in \mathbb{F}_{2^n}$.

Note, we integrated the parameters mentioned above to speed-up our implementation of HECC. However, all explicit formulae presented in the appendix are for the most general case.

4.3.1 Optimization of Cantor's Group Operation

In this subsection, we consider the group operations presented in [Can87] and apply the techniques introduced in Chapter 4.1, resulting in efficient group operations for genus-2, genus-3, and genus-4 HECC. Table 4.2 summarizes the efforts made to date and our contribution to speed up Cantor's algorithm. The detailed instruction, how to calculate the group operations are given in the appendix.

This is the first contribution that presents explicit formulae of Cantor's algorithm for genus 2, 3, and 4. Hence, we were able to speed up the group operations for all genera. In the following, we will compare our results with the ones presented in [Nag00]. Our improvements are stated in Table 4.2 and are based on the following assumptions:

- We consider the cardinality of the resulting group to be $\approx 2^{160}$.

- We apply different time complexities for the field inversions over $\mathbb{F}_{2^m}$ (the following numbers are based on the implementation results presented in Table 6.9). Some of the implementations might have potential cryptographical weaknesses according to [Fre98, GHS02], however, we would like to emphasize that the implementation techniques are also applicable to fields with prime extensions, see Page 4.

  - $m = 81$ for genus-2 HEC and $m = 57$ for genus-3 HEC: one inversion has the same time complexity as six multiplications.
  - $m = 53$ for genus-4 HEC: one inversion has the same time complexity as four multiplications.
  - We compare our results with the best known formulae published.

Based on the assumption given above, one draws the following conclusions: Our genus-2 HEC group operation saves up to $6M/S$ in the case of adding divisors and $1I$ as well as up to $20M/S$ in the case of doubling a divisor. This complies with an improvement of 9% and 43% for addition and doubling, respectively. Using our formulae for certain genus-3 curves one can save 24% (corresponding to $40M/S$) and 78% (equals $1I$ and $107M/S$) of the computational cost for adding and doubling, respectively. In addition, we were able to reduce the computational cost of genus-4 curves by $76M/S$ for group addition and $1I$ and $171M/S$ doubling. Hence, the speed up in these cases were 26% and 64%, respectively.

4.3.2 Optimization of Harley's Group Operation

The approach presented in [GH00] was also optimized by us using the techniques in Chapter 4.1. We were able to speed up the group operation of genus-2, genus-3 and genus-4 HECC. Table 4.3 presents a summary of our work, as well as of all the previous

work that has been done on improving Harley's algorithm. All the steps necessary to compute the group operations using the explicit formulae are presented in the appendix.

The analysis of our results in Table 4.3 is based on: the cardinality of the underlying group is $\approx 2^{160}$, in the case genus-2 HEC and genus-3 HEC one inversion has the same time complexity as six multiplications, and for genus-4 HEC one inversion has the same time complexity as four multiplications. The time complexity assumptions are based on our implementation results (see Table 6.9). Hence one can draw the following conclusions from our newly derived group operations based on Harley's considerations:

- We introduced for the first time genus-3 explicit formulae for arbitrary characteristic.
- This work presents (for the first time) explicit formulae for hyperelliptic curve cryptosystems of genus 4 based on Harley's algorithm.
- We could improve the explicit formulae for arbitrary characteristic for genus-3 and genus-4 hyperelliptic curves compared to the previous publications [KGM$^+$02] and [Nag00], respectively.
- For characteristic two fields, we could decrease the costs to calculate the group operations on special genus-2, genus-3, and genus-4 curves.
- Considering certain curve parameters we were able to save 35% for the genus-2 doubling operation compared to the best known formulae [Lan02a, Lan03].
- Our fastest genus-3 group curve uses $42M/S$ less for the group doubling as presented in [GMA$^+$04]. These savings correspond to 54%.
- In the case of genus-4 curves there is no other publication introducing explicit formulae. Hence, we compare our results with the best known formulae based on

Cantor's equations and presented in [Nag00]. Our results need one inversion as well as 140 and 188 less multiplications/squarings for the addition and doubling, respectively. Hence, we were able to improve the performance by 46% and 66% for the genus-4 HEC group operations.

4.3.3 Optimization of Inversion-Free Explicit Formulae

In [MDM+02, Lan02b, Lan03], the authors introduced a way to calculate the HEC group operations without using inversions. The authors in [MDM+02] were the first to publish this kind of approach. Their results were improved and generalized to even characteristic in [Lan02b, Lan03]. For the remainder of the paper we refer to *affine* coordinates, when talking about the usual representation. *Projective* denotes the inversion-free representation (analogous to the elliptic curve cryptosystem).

In order to save the inversion one has to introduce a coordinate space Z. Hence, the two polynomials representing a divisor are given as following: $u(x) = x^2 + U_1/Zx + U_0/Z$ and $v(x) = V_1/Zx + V_0/Z$. We abbreviate a divisor in projective system as $[U_1, U_0, V_1, V_0, Z]$.

In order to obtain a greater speed-up for some scalar multiplication algorithms one can use mixed coordinates as inputs for the addition. Mixed addition was first introduced in [Lan02b] building on [CMO98].

We optimized the group doubling, group addition, and the mixed addition and the newly derived formulae can be found in Table A.15, A.16, and A.17, respectively. In addition, we optimized the explicit formulae for inversion-free doubling using affine input, see Table A.18. This formulae can be useful to get another small speed gain, in applications where area or code size is not an issue. The decrease of field operations for projective group operations could be achieved by applying the methods described in

Section 4.1.

Table 4.4 compares our result with the best ones previously published. The complexity of the squaring operations for characteristic two fields is very low and can therefore be neglected in the comparison. We could achieve up to 18% improvement with our newly derived formulae in the case of the doubling operation. Hence, the scalar multiplication using double-and-add can be improved by 14% for 160bit security.

4.4 Summary: HECC Group Operation

The aim of this section is to give a summary of the chapter. In the first part of the chapter we presented our techniques used for the improvement of the explicit formulae. The section includes the description of well known techniques like Montgomery's trick of simultaneous inversions, Karatsuba multiplication, and efficient division. Furthermore, we explained how to use the Bezout's matrix and the reordering of calculation steps to improve the HECC group operations.

In the second part of the chapter, we generalized the Karatsuba reduction that is used for efficient polynomial reduction. This technique allows to substitute one multiplication by three addition. Therefore, Karatsuba reduction is more efficient if the execution time for one coefficient multiplication is higher than the time to perform three additions.

In the third part of the chapter we introduced our explicit formulae. In this section, we provided the most efficient HECC group operations at the time of writing. The bold numbers in Table 4.2, 4.3 and 4.4 indicate the fastest group operations to date where we include special cases. Table 4.5 summarizes the explicit formulae one should use for an efficient implementation of HECC. In most of the cases, the group operations based on Harley's algorithm perform better. However, it is noticeable that the performance of the doubling operation for certain curves based on Harley or Cantor are almost the

same.

We showed that explicit formulae for the group doubling based on Cantor's algorithm in the case of genus-3 HEC are faster than the explicit formulae based on Harley. This result was achieved through the choice of specific curve parameters that allow further optimization in Cantor's algorithm.We are aware of the fact that Harley's approach should always be faster compared to Cantor's algorithm. However, even after a lot of effort was put into the improvement of Harley's algorithm we were not able to match the performance numbers we achieved with the Cantor approach.

At this point we want to clarify that we applied a sequence of improvements techniques to achieve the presented results. This is somewhat of an ad-hoc approach which does not necessarily lead to optimal results. Rather, the complexity of the explicit formulae presented in this chapter should be viewed as upper bound. One can most probably improve the given group operations further, by finding new improved methods or by restructuring the operations and using the methods given in Chapter 4.1. Hence, this chapter presented the historical development of HECC as well as the best explicit group operations known today. We challenge the research community as well as industry to further improve the group operations leading to a faster computation of HECC.

Table 4.2: Group operations using Cantors Algorithm.

genus		field charac.	curve properties	cost addition	cost doubling
2	Cantor [Nag00]	general		$3I + 70M/S$	$3I + 76M/S$
	Nagao [Nag00]	odd	$h(x) = 0,\ f_i \in \mathbb{F}_2$	$2I + 52M/S$	$2I + 49M/S$
	our work	general	$h_i \in \mathbb{F}_2,\ f_4 = 0$	2I + 44M + 4S	2I + 42M + 8S
		two	$h_i \in \mathbb{F}_2,\ f_4 = 0$	2I + 42M + 4S	2I + 40M + 8S
		two	$h(x) = x,\ f_4 = 0$	2I + 42M + 4S	I + 23M + 6S
				Table A.4	Table A.5
improvements				9%	43%
3	Cantor [Nag00]	general		$4I + 200M/S$	$4I + 207M/S$
	Nagao [Nag00]	odd	$h(x) = 0,\ f_i \in \mathbb{F}_2$	$2I + 154M/S$	$2I + 132M/S$
	our work	general	$h_i \in \mathbb{F}_2,\ f_6 = 0$	2I + 118M + 4S	2I + 106M + 19S
		two	$h_i \in \mathbb{F}_2,\ f_6 = 0$	2I + 110M + 4S	2I + 98M + 13S
		two	$h(x) = 1,\ f_6 = 0$	2I + 110M + 4S	**I + 14M + 11S**
				Table A.8	Table A.9
improvements				24%	78%
4	Cantor [Nag00]	general		$6I + 386M/S$	$6I + 359M/S$
	Nagao [Nag00]	odd	$h(x) = 0,\ f_i \in \mathbb{F}_2$	$3I + 286M/S$	$3I + 260M/S$
	our work	general	$h_i \in \mathbb{F}_2,\ f_8 = 0$	3I + 222M + 6S	3I + 206M + 17S
		two	$h_i \in \mathbb{F}_2,\ f_8 = 0$	3I + 204M + 6S	3I + 181M + 14S
		two	$h(x) = x,\ f_8 = 0$	3I + 204M + 6S	2I + 76M + 13S
				Table A.13	Table A.14
improvements				26%	64%

Table 4.3: Group operations using Harley's Algorithm.

genus		field charac.	curve properties	cost addition	cost doubling
2	Harley [Har00]	odd	$h(x) = 0$	$2I + 27M/S$	$2I + 30M/S$
	Lange [Lan01]	general	$h_2 = 1$	$2I + 24M + 3S$	$2I + 26M + 6S$
	Matsuo et al. [MCT01]	odd	$h(x) = 0$	$2I + 25M/S$	$2I + 27M/S$
	Miyamoto et al. [MDM+02]	odd	$h(x) = 0$, $f_4 = 0$	$I + 26M/S$	$I + 27M/S$
	Takahashi [Tak02]	odd	$h(x) = 0$	$I + 25M/S$	$I + 29M/S$
	Sugizaki et al. [SMCT02]	even	$h_i \in \mathbb{F}_2$	$I + 25M + 2S$	$I + 27M + 1S$
	Lange [Lan02a, Lan03]	general	$f_4 = 0$	$I + 22M + 3S$	$I + 22M + 5S$
		two	$f_4 = 0$	$I + 21M + 3S$	$I + 20M + 5S$
		two	$h_2 = 0$		
			$f_4 = 0$	**I+21M+3S** Table A.1	$I + 17M + 5S$ Table A.2
	our work	two	$h(x) = x$ $f_4 = f_3 = f_2 = 0$	–	**I+9M+6S** Table A.3
improvements				-	35%
3	Kuroki et al. [KGM+02]	odd	$h(x) = 0$, $f_6 = 0$	$I + 81M/S$	$I + 74M/S$
	Gonda et al. [GMA+04]	general	h(x)=0, $f_6 = 0$	$I + 70M/S$	$I + 71M/S$
	our work	general	$h_i \in \mathbb{F}_2$, $f_6 = 0$	$I + 70M + 6S$	$I + 62M + 10S$
		two	$h_i \in \mathbb{F}_2$, $f_6 = 0$	$I + 65M + 6S$	$I + 53M + 10S$
		two	$h(x) = 1$, $f_6 = 0$	**I+65M+6S** Table A.6	$I + 22M + 7S$ Table A.7
improvements				-	54%
4	our work	general	$h_i \in \mathbb{F}_2$, $f_8 = 0$	$2I + 158M + 6S$	$2I + 193M + 17S$
		two	$h_i \in \mathbb{F}_2$, $f_8 = 0$	$2I + 146M + 6S$	$2I + 144M + 17S$
		two	$h(x) = x$, $f_8 = 0$	**2I+146M+6S** Table A.11	**2I+72M+13S** Table A.12
improvements				46%	66%

Table 4.4: Inversion-free group operations for genus-2 HEC.

	charac.	curve properties	addition	doubling	mix addition	affine doubling
[MDM+02]	N.A.	N.A.	67M	42M	-	-
[Lan02b]	odd	$h_2 = 0$, $f_4 = 0$	47M + 4S	38M + 6S	40M + 4S	25M + 5S
	two	$h_2 = 0$, $f_4 = 0$	49M + 4S	38M + 7S	39M + 4S	
our work	two	$h(x) = x$ $f_4 = f_3 = 0$	**45M + 5S** Table A.15	**31M + 6S** Table A.16	**38M + 5S** Table A.17	**18M + 5S** Table A.18
improv.			8%	18%	3%	28%

Table 4.5: Most efficient group operations for HECC.

genus	addition	doubling
2	I+21M+3S Table A.1 [Lan03]	I+9M+6S Table A.3
	inversion-free	
	45M + 4S Table A.15 (mixed: 38M + 4S, Table A.17)	31M + 7S Table A.16 (affine input: 18M + 5S, Table A.18)
3	I+65M+6S Table A.6	I + 14M + 11S Table A.10
4	2I+146M+6S Table A.11	2I+72M+13S Table A.12

5 A New Complexity Metric for HECC and ECC

In the past, providing complexity measures and, thus, comparisons between ECC and HECC was a difficult undertaking. The operations involved in both systems were very different (different field orders, field operations vs. operations with polynomials, etc.). Furthermore, measures such as the bit complexity often provide very little information about the *de facto* complexity in actual implementations. The underlying motivation for the work described in the following was the development of a more accurate metric for practical purposes. Part of this work was presented in [PWGP03].

5.1 Previous Theoretical Comparisons

In [SSI98], the authors clarified practical advantages of hyperelliptic cryptosystems when compared to ECC and to RSA. To our knowledge this is the first and only contribution that investigates in detail the theoretical complexity of ECC and HECC. They estimated the cost of different cryptosystems based on the number of bit operations. In their work they used Cantor's formula and the cost of one multiplication in $\mathbb{F}_{2^n}$ was assumed to take n^2 bit operations. One of the estimated theoretical results shows that genus-3 curves needed three times as many bit operations as elliptic curves. We want to

point out that this publication used supersingular curves [Gal01] and curves of genus higher than 4 which today are believed to be insecure due to the attacks presented in [FR94, Gau00b, Gal01, Thé03].

In the following years further analysis of the complexity of HECC were published. A theoretical analysis of the computational efficiency of the arithmetic on hyperelliptic curves is derived in [Eng99b]. In [SS00], the authors implemented hyperelliptic curve cryptosystems and analyzed the complexity of the group law on Jacobians $\mathbb{J}_C(\mathbb{F}_p)$ and $\mathbb{J}_C(\mathbb{F}_{2^n})$. Moreover, they verified their theoretical complexity estimates with a HECC implementation and with the theoretical analysis done by Enge in [Eng99b]. More recent papers present timings for HECC using explicit formulae and compare HECC to ECC [Lan02a, Ava04]. However, these comparisons were based on the implementation timings.

In [Ste01], the author compared the arithmetic of hyperelliptic function fields. The author was able to present upper bounds on the number of operations in various situations. Furthermore, the author was able to show that the group operation in imaginary quadratic function fields and the corresponding infrastructure operation in real quadratic function fields have identical complexity. Note that the hyperelliptic function field can be represented as a real quadratic function field.

To our knowledge, there is no theoretical complexity comparison between ECC and HECC published that uses the explicit formulae for HECC and compares HECC and ECC in terms of processor instructions, such as shift and XOR operations. Hence, this comparison is processor independent and can be adapted to any platform.

5.2 Our Metric

In this section, we introduce our new metric for HECC and ECC over characteristic two fields which is based on an atomic operation count rather than on the (theoretical) bit complexity or specific timings.

All operations which are computationally expensive will be expressed in terms of *atomic operations* (AOPS), such as processor word-SHIFTs and XORs. In particular, we will decompose field multiplications into AOPS. This provides a metric which allows a comparison of fields of different sizes which is crucial for comparing ECC and HECC with equal level of security. The approach possesses the advantage that it accurately counts the actual elementary processor operations (as opposed to the more theoretical bit complexity), while at the same time avoiding processor and implementation-dependent "tricks" which can skew comparisons that are merely based on timings.

In summary, we developed a method which allows accurate predictions of the performance on a given processor without the laborious task of actually implementing the cryptosystem. The accuracy of the new metric is demonstrated by the small difference between our theoretical and practical results (see Section 6.3.5).

In our comparison we make the following assumptions:

1. We only consider fields of characteristic two and, thus, do not need integer multiplications.

2. We neglect the cost of the field addition and field squaring.

3. We perform field multiplications with Algorithm 5 published[1] in [LD00]. This algorithm requires $3 + 2(w/4 - 1)$ word-SHIFTs and $s(11 + n/4) + 8(2s - 1)$ word-

[1] To our knowledge this is the fastest published multiplication algorithm for finite fields of characteristic two.

XORs, where w is the word size of processor and $s = \lceil \frac{n}{w} \rceil$ is the number of words needed to represent an element of the underlying field $\mathbb{F}_{2^n}$.

4. We express the cost of one field inversion as x field multiplications and denote the ratio of multiplications to inversions as MI-ratio. Note that for different underlying fields one gets different MI-ratios. Hence, we use in the following different variables to express different rations because of the varying field sizes for ECC, genus-2, genus-3, and genus-4 HECC.

5.3 Comparing ECC and HECC

Based on the assumptions stated in Section 5.2, the complexity of the group operations of HEC and EC are summarized.

Referring to Table 4.5, a divisor addition for a genus-2 curve requires $1I + 21M$ and doubling needs $1I + 9M$. Assuming that the cost of one field inversion is equivalent to l field multiplications, leads to $(21 + l)M$ and $(9 + l)M$ for addition and doubling, respectively. The variable l corresponds to the MI-ratio for genus-2 HECC. When using the projective coordinates for genus-2 curves no field inversion is required and therefore the computational complexity is not conditioned by the MI-ratio. Thus, we get $45M$ for the group addition and $31M$ for the group doubling (see Table 4.5).

Due to the lower extension of the underlying field used for genus-3 curves, a different MI-ratio m is used. This leads to $(65+m)M$ for a divisor addition and $(14+m)M$ for a divisor doubling. In the case of genus-4 curves one gets $(146+2r)M$ and $(72+2r)M$ for addition and doubling, respectively, where r denotes the MI-ration for genus-4 HEC.

The number of inversions and multiplications for a group operation on EC heavily depends on the chosen coordinate system (like in the case of hyperelliptic curves). For

the sake of completeness we summarize the number of required operations in Table 5.1. One notices that by using projective coordinates we do not need to provide inversion and therefore the MI-ratio is not necessary. In contrary, considering affine coordinates, we need another MI-ratio for the corresponding field sizes, denoted as j.

Table 5.1: Field operations required for ECC in each coordinate system [HHM00].

Coordinate system	EC Addition		EC Doubling
	general	mixed coord.	
Affine	$1I + 2M$ $= (2 + j)M$		$1I + 2M$ $= (2 + j)M$
Standard projective [CC87]	$13M$	$12M$	$7M$
Jacobian projective [CC87]	$15M$		$5M$
New projective [LD98]	$14M$	$9M$	$4M$

Table 5.2 states the total number of AOPS for the group operations of the cryptosystems with different MI-ratios. In terms of ECC, affine coordinates, Jacobian projective coordinates, and new projective coordinates are considered. The variables n_i represent the extension of the underlying field $\mathbb{F}_{2^{n_i}}$. To provide a fair comparison, one needs to consider the same security level for all cryptosystems. Hence, we take into account the latest attack presented in [Thé03] and therefore increase the group order for genus-3 and genus-4 curves (for more details see Table 2.1). For a given processor, Table 5.2 allows an immediate, fairly accurate prediction of the ECC and HECC performance. We would like to remark that we did not improve the new projective coordinates for HECC as proposed in [Lan03] and therefore we did not include them in the given comparison (it is not fair to compare optimized formulae with non optimized formulae).

Figure 5.1 illustrates the number of operations for a scalar multiplication on a 32-bit processor depending on the MI-ratios, considering the findings in [Thé03]. Hence, we used the fields $\mathbb{F}_{2^{81}}$, $\mathbb{F}_{2^{57}}$, and $\mathbb{F}_{2^{53}}$ for HEC of genus two, three, and four, respectively, considering 163bit security. Some of the implementations might have potential crypto-

Table 5.2: Total number of atomic operations for ECC and HECC (underlying field $\mathbb{F}_{2^{n_i}}$, processor word w, MI-ratios j, l, m, r).

	ECC		
	affine	Jacobian projective	new projective (mixed)
Add	$(2+j)\cdot$ $\lceil \frac{2w}{4} + (\frac{n_1}{4} + 27)\lceil \frac{n_1}{w} \rceil - 7 \rceil$	$15\cdot$ $\lceil \frac{2w}{4} + (\frac{n_1}{4} + 27)\lceil \frac{n_1}{w} \rceil - 7 \rceil$	$9\cdot$ $\lceil \frac{2w}{4} + (\frac{n_1}{4} + 27)\lceil \frac{n_1}{w} \rceil - 7 \rceil$
Doub	$(2+j)\cdot$ $\lceil \frac{2w}{4} + (\frac{n_1}{4} + 27)\lceil \frac{n_1}{w} \rceil - 7 \rceil$	$5\cdot$ $\lceil \frac{2w}{4} + (\frac{n_1}{4} + 27)\lceil \frac{n_1}{w} \rceil - 7 \rceil$	$4\cdot$ $\lceil \frac{2w}{4} + (\frac{n_1}{4} + 27)\lceil \frac{n_1}{w} \rceil - 7 \rceil$
	HECC		
	genus-2	genus-3	genus-4
Add	$(21+l)\cdot$ $\lceil \frac{2w}{4} + (\frac{n_2}{4} + 27)\lceil \frac{n_2}{w} \rceil - 7 \rceil$	$(65+m)\cdot$ $\lceil \frac{2w}{4} + (\frac{n_3}{4} + 27)\lceil \frac{n_3}{w} \rceil - 7 \rceil$	$(146+2r)\cdot$ $\lceil \frac{2w}{4} + (\frac{n_4}{4} + 27)\lceil \frac{n_4}{w} \rceil - 7 \rceil$
Doub	$(9+l)\cdot$ $\lceil \frac{2w}{4} + (\frac{n_2}{4} + 27)\lceil \frac{n_2}{w} \rceil - 7 \rceil$	$(14+m)\cdot$ $\lceil \frac{2w}{4} + (\frac{n_3}{4} + 27)\lceil \frac{n_3}{w} \rceil - 7 \rceil$	$(72+2r)\cdot$ $\lceil \frac{2w}{4} + (\frac{n_4}{4} + 27)\lceil \frac{n_4}{w} \rceil - 7 \rceil$
	Inversion free HECC		
Add	$45\cdot$ $\lceil \frac{2w}{4} + (\frac{n_2}{4} + 27)\lceil \frac{n_2}{w} \rceil - 7 \rceil$	mixed add: $38\cdot$ $\lceil \frac{2w}{4} + (\frac{n_2}{4} + 27)\lceil \frac{n_2}{w} \rceil - 7 \rceil$	
Doub	$31\cdot$ $\lceil \frac{2w}{4} + (\frac{n_2}{4} + 27)\lceil \frac{n_2}{w} \rceil - 7 \rceil$	$31\cdot$ $\lceil \frac{2w}{4} + (\frac{n_2}{4} + 27)\lceil \frac{n_2}{w} \rceil - 7 \rceil$	

graphical weaknesses according to [Fre98, GHS02], however, we would like to emphasize that the implementation techniques are also applicable to fields with prime extensions, see Page 4. The scalar multiplication with a k-bit scalar is realized by the sliding window method with an approximated cost of $n \times$ doublings $+ 0.2 \times k \times$ additions for a 4-bit window size [BSS99]. Figure 5.1 allows to estimate the efficiency of an ECC or a HECC built on top of a given field library by comparing the different MI-ratios.

In general we can draw the following conclusions from this comparison:

1. ECC with new projective coordinates is in almost all cases the most efficient cryptosystem.

2. Scalar multiplication of genus-2 and genus-3 HEC and affine ECC have very similar performance for small MI-ratios. If one uses a software library with a high ratio, genus-2 and genus-3 HEC are preferable compared to affine ECC.

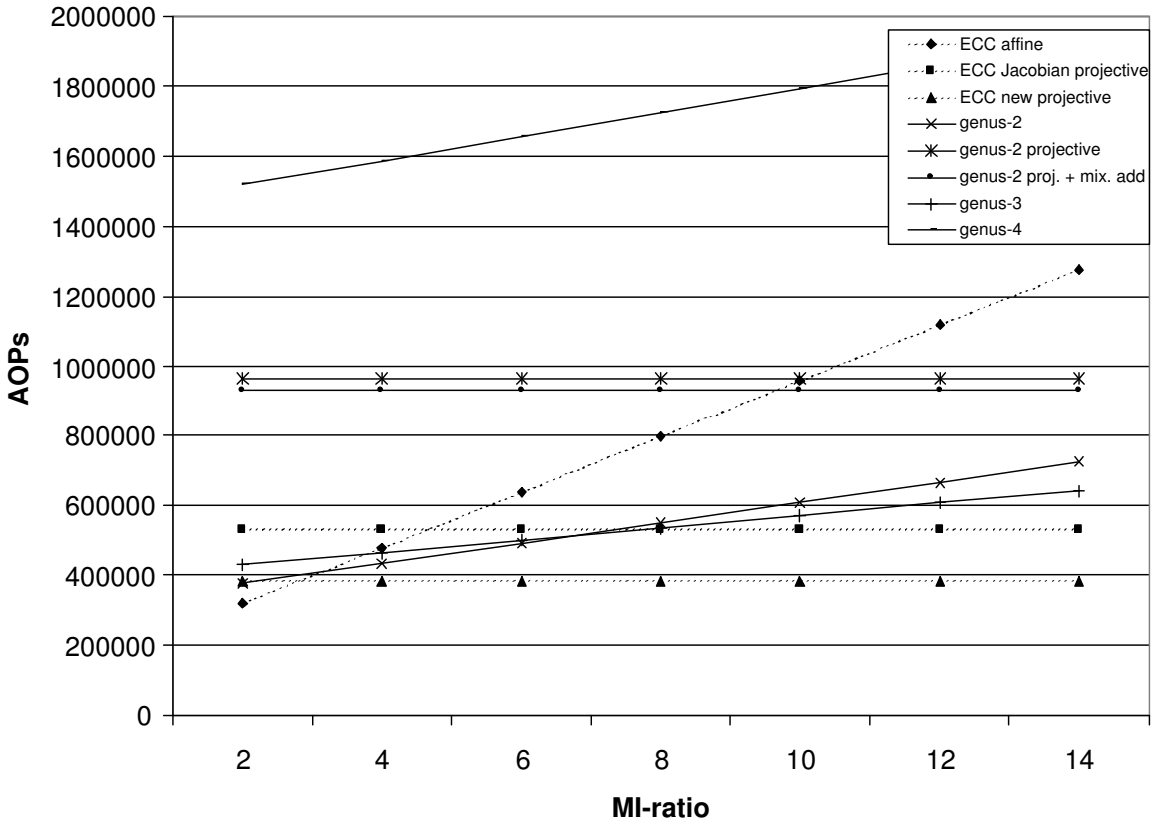


Figure 5.1: Cost of a scalar multiplication for different MI-ratios and cryptosystems in AOPS (32-bit μP, group order $\approx 2^{163}$).

3. Genus-2 curves are more efficient for low MI-rations, whereas genus-3 curves use a smaller amount of AOPs for a higher ratio.

4. Genus-4 HEC is the cryptosystems using the most time to complete a scalar multiplication.

Note that performance of HECC using projective coordinates could be improved by using the new projective coordinates.

6 Software Implementation of Hyperelliptic Curve Cryptosystem

In this chapter we analyze the software performance of HEC cryptosystems. Part of the work presented in this chapter was published in [PWGP03,PWP03,WPW+04,PWP04b, PWP04a]. We implemented HECC on a general purpose processor (Section 6.2), namely the Intel Pentium, as well as on popular embedded processors (ARM, ColdFire, and PowerPC — see Section 6.3). We also implemented ECC for comparison reasons.

The overall performance of EC and HEC cryptosystems depends not only on the specific algorithms but also on the underlying implementation and the processor type used. In particular, we analyzed how different EC and HEC cryptosystems perform with respect to certain settings of both the software routines and the hardware components.

6.1 Methodology for the Software Implementation

We implemented different variants of EC and HEC cryptosystems. The characteristic of the underlying fields is two and the cardinality of the groups ranges between 2^{160} and 2^{252}. All operations are implemented for 32-bit microprocessors using the C programming language. To allow portability, the implementation was not optimized for a

specific platform[1]. Compiler settings for optimal speed were used depending on the tools available.

In the following, we are going to describe the hardware platforms used, the finite field arithmetic and the group arithmetic for ECC and HECC.

6.1.1 Processors Used for the Software Implementation

The cryptosystems were implemented on the Pentium4@1.8GHz and on different hardware architectures for embedded systems. ARM7, ColdFire, and PowerPC were chosen as testing platforms for the extensive analysis of ECC and HECC. The embedded platforms and features will be explained in the following paragraphs and are summarized in Table 6.1.

ARM: ARM (*Advanced RISC Machine*) processors are typically used for embedded applications such as small network devices, controllers and mobile phones.

On the ARM microprocessor [ARM00], instruction decoding is performed with static (i.e., hard-wired) logic for a faster result. The ARM7TDMI is based on von Neuman Architecture and is licensed by ARM Ltd. All instructions have a fixed uniform length to simplify the decoding procedure. Since direct manipulation of data in the memory is not possible, a load/store architecture handles data processing through registers. The simple address mode allows to determine all load/store addresses from the register contents and the instruction parameters. For low power consumption, the ARM7 possesses the *Thumb Instruction Set* which is restricted to 16-bit and allows compact code and, thus, is feasible for small hand held devices such as PDAs.

The ARM7TDMI consists of a program control unit, an address generator, an integer data path, and a general-purpose register bank. The data path contains a 32-bit integer

[1]Significant speed gains can be achieved by implementing the core routines in assembly using processor specific operations.

ALU, a multiply-add unit, and a barrel shifter. The 32-bit arithmetic logical unit (ALU) performs simple integer arithmetic operations such as addition and subtraction. The core features a multi-cycle 32x32 to 64-bit multiplier. It has a total of 37 registers: 31 general-purpose 32-bit registers, and 6 status registers. Speed-critical control signals are pipelined so that system control functions can be implemented in standard low-power logic. The ARM7TDMI does not support floating-point arithmetic in hardware and does not have any DSP-specific features.

ColdFire: The ColdFire microprocessor is the successor of the 68000 series. Besides the use as low cost controller (laser printers), the ColdFire is used in general purpose industry applications such as network elements (routers, bridges).

In version 3, the processor consists of two independent, decoupled pipeline structures [Mot00a]. The instruction fetch pipeline is a six-stage pipeline for prefetching instructions and contains logic for branch prediction. To maximize the performance, the 4KByte on-chip SRAM provides one-cycle access for the ColdFire core. This SRAM can store processor stack and critical code or data segments to maximize performance. The processor core possesses a hardware integer divide unit and supports a 16x16 and 32x32 bit multiplication. The ColdFire features sixteen 32-bit general-purpose registers.

PowerPC: Typical applications for embedded PowerPCs include powerful general purpose microcontrollers, data acquisition systems, applications in robotics, automotive and consumer electronics.

The standard PowerPC architecture has a fully static design that consists of three functional blocks: the integer block, hardware multiplier/divider, and load/store block [Mot00b]. The core supports integer operations on a 32-bit internal data path and 32-bit arithmetic hardware. Its interface to the internal and external busses is 32 bits. The PowerPC integer block supports 32 x 32-bit fixed-point general-purpose registers and can execute one integer instruction per clock cycle. The core is integrated with the memory

management units, an instruction cache, and a data cache. The 8KByte data cache allows single-cycle accesses. The two way 16KByte instruction cache is set-associative.

The PowerPC offers the possibility to disable the data cache as well as the instruction cache separately. For this reason we investigate four different options for the cache: cache enabled, data cache only, instruction cache only, and cache disabled. The timings under these options provide information about the performance depending on the cache. For programs with intensive memory access, the data cache may play a more significant role than the instruction cache, whereas for projects consisting of small functions which get called several times, the instruction cache might be more important.

The PowerPC allows disabling of the pipelining mode. If the core is in *non-serialized* mode, no pipelining is applied. Hence, the next processor command is executed only after the previous has been processed completely. In *serialized* mode, full pipelining is enabled.

Table 6.1: Hardware platforms used.

	board & processor	clock rate [MHz]	memory/cache [kByte]	tools
ARM	Evaluator-7T KS32C50100	50	512 flash EPROM 512 SRAM 8 cache	ARM Developer Suite 1.2
ColdFire	SBC5307 Arnewsh MFC5307	90	4 SRAM 8 cache	SingleStep Deb. 7.6.2 Diab Data Comp. 4.3f.
PowerPC	TQsystem MPC823E	50	8 data cache 16 instruction cache	SingleStep Deb. 7.6.2 Diab Data Comp. 4.3f.

6.1.2 Finite Field Arithmetic

The speed of the underlying implementation of the field arithmetic is crucial for the overall performance of the cryptosystem. Restricting oneself to a certain field with a fixed

field extension polynomial offers the possibility to benefit from special field reduction routines. In our work, we investigated the performance gain of such special routines versus general routines and the resulting benefit to the overall performance. A brief summary of the algorithms used for the field arithmetic is given below.

Field addition, multiplication, squaring, inversion, and reduction are the basis for the group operations on elliptic and hyperelliptic curves. Adding elements in $\mathbb{F}_{2^n}$ is simply accomplished by a bitwise XOR of the components. A field multiplication of m_1 words times m_2 words is split up into several multiplications with a smaller number of words. The algorithm is a modified version of Karatsuba [KO63]. For fields in even characteristic, squaring can be done very fast by table lookups [SOOS95]. The modified Extended Euclidean Algorithm (EEA) is applied for inversion in $\mathbb{F}_{2^n}$ [HHM00]. Furthermore, we were able to speed up this algorithm with a small modification concerning the calculation of the degree difference.

To represent elements of the extension field $\mathbb{F}_{2^n} = \mathbb{Z}_2/p(x)$, we need to choose an irreducible polynomial. In [vzGN00], the authors conjecture that the minimal number of terms $\sigma_q(n)$ in irreducible polynomials of degree n over $GF(q)$, where q is a prime power, is for all $n \geq 1$, $\sigma_2(n) \leq 5$ and $\sigma_q(n) \leq 4$ for $q \geq 3$. This conjecture has been verified for $q = 2$ and $n \leq 10000$ [BGL93,Gol67,vzGN00,Zie70,ZB68,ZB69] and for $q = 3$ and $n \leq 539$ [vzG01]. Hence, we found extension polynomials that are either trinomials of the form $p(x) = x^n + x^k + 1$ or pentanomials of the form $p(x) = x^n + x^{k_1} + x^{k_2} + x^{k_3} + 1$.

The implemented standard reduction function considers tri- and pentanomials and is able to treat arbitrary values k_i. The reduction itself is done word-wise according to [HHM00, Algorithm 6]. In order to achieve a higher speed-up we additionally implemented a special reduction function for each underlying field, where the k_i are fixed. For the remainder of this contribution, we refer to *special* reduction when using a fixed field extension polynomial. Whereas the term *standard* reduction is used for a generically

implemented reduction routine and the extension polynomial is not known in advance. For a server with different cryptographic applications, standard routines have to be implemented whereas implementations on constrained platforms need only specialized settings.

6.1.3 Group Arithmetic

Group Arithmetic on Elliptic Curves

The implementation of the high level elliptic curve group operations uses projective coordinates according to the standard IEEE P1363 [P1399]. The operations performed are as follows:

- point addition – in general this algorithm requires 5 field squarings, 14 general field multiplications
- point doubling – this algorithm requires 5 field squarings, 5 general field multiplications

Group Arithmetic on Hyperelliptic Curves

For the group operations on hyperelliptic curves of genus-2, 3, and 4, the explicit formulae were implemented. We applied the fastest explicit formulae (at the time of implementation) for the group operations.

Remark: The implementations were done over the last three years and at the same time a lot of work was done in improving the group operations. Hence, we used slightly different group operations for some implementations.

The extensive comparison of our implementation on the three embedded processors, was done with the following underlying group operations: group addition and doubling

for genus-2 curves as presented in Tables A.1 and Table A.2; group addition and doubling for genus-3 curves are presented in Table A.6 and Table A.7, respectively. The implementation results are presented in sections 6.3.1, 6.3.2, and 6.3.3

After we improved the group doubling for genus-2 (Table A.3) and genus-3 (Table A.10) hyperelliptic curves, we implemented them on the ARMulator@80MHz and Pentium@1.8GHz. For the genus-4 implementation we used the explicit formulae for the group addition and group doubling, Table A.11 and Table A.12, respectively. The implementation results on the Pentium@1.8GHz can be found in Section 6.2 and for the ARMulator@80MHz in Section 6.3.4.

For the main operation of the cryptosystem, the repeated addition of a divisor (or scalar multiplication), we used the sliding window exponentiation algorithm [MvOV97, Section 14.6.1].

6.2 Hyperelliptic Curve Cryptosystem on General Purpose Processors

In this section we present our implementation results on the Pentium@1.8GHz. We introduce our timings for ECC as well as genus-2, 3, and 4 HECC. We implemented the best explicit formulae for characteristic two fields as summarized in Section 4. In order to be able to compare our results with an elliptic curve cryptosystem, we implemented ECC for some cases with the same methodology.

In Table 6.2, the performance of the different cryptosystems targeting a variety of different underlying fields are presented. We list our timings for the group addition, group doubling as well as for the scalar multiplication. Some of the implementations might have potential cryptographical weaknesses according to [Fre98, GHS02], however,

Table 6.2: Timings for ECC and HECC on the Pentium4 @1.8GHz.

Genus	Field	Group order	Group addition in μs	Group doubling in μs	Scalar. mult. in ms
1	$\mathbb{F}_{2^{163}}$	2^{163}	18.3	9.4	2.60
	$\mathbb{F}_{2^{167}}$	2^{167}	19.2	8.5	2.43
	$\mathbb{F}_{2^{179}}$	2^{179}	16.9	9.8	2.80
	$\mathbb{F}_{2^{191}}$	2^{191}	15.4	8.7	2.78
2	$\mathbb{F}_{2^{81}}$	2^{162}	18.7	11.7	2.73
	$\mathbb{F}_{2^{83}}$	2^{166}	20.2	12.7	3.01
	$\mathbb{F}_{2^{88}}$	2^{176}	20.5	13.2	3.30
	$\mathbb{F}_{2^{91}}$	2^{182}	21.1	13.7	3.50
	$\mathbb{F}_{2^{95}}$	2^{190}	19.0	12.6	3.41
3	$\mathbb{F}_{2^{54}}$	2^{162}	24.8	8.9	2.56
	$\mathbb{F}_{2^{55}}$	2^{165}	25.2	9.0	2.69
	$\mathbb{F}_{2^{57}}$	2^{171}	25.5	9.4	2.95
	$\mathbb{F}_{2^{59}}$	2^{177}	28.5	10.3	3.22
	$\mathbb{F}_{2^{60}}$	2^{180}	24.8	9.2	2.97
	$\mathbb{F}_{2^{61}}$	2^{183}	28.5	10.5	3.34
	$\mathbb{F}_{2^{63}}$	2^{189}	25.3	9.2	3.10
4	$\mathbb{F}_{2^{40}}$	2^{160}	58.6	32.9	7.76
	$\mathbb{F}_{2^{41}}$	2^{164}	53.2	29.7	7.50
	$\mathbb{F}_{2^{44}}$	2^{176}	53.4	30.0	7.90
	$\mathbb{F}_{2^{46}}$	2^{184}	53.3	29.8	8.43
	$\mathbb{F}_{2^{47}}$	2^{188}	53.8	30.7	8.59
	$\mathbb{F}_{2^{53}}$	2^{212}	64.2	36.0	10.18
	$\mathbb{F}_{2^{59}}$	2^{236}	65.2	37.9	10.36
	$\mathbb{F}_{2^{63}}$	2^{252}	63.7	33.1	9.50

we would like to emphasize that the implementation techniques are also applicable to fields with prime extensions, see Page 4.

It is only useful to compare cryptosystems with the same security level. Hence, we have to take the recent progress in computing the discrete logarithm of HECC by Thériault [Thé03] into consideration. Note, that this is a very pessimistic estimation, since the attacks presented can not be realized because of the high complexity. The results in [Thé03] suggest to enlarge the underlying fields of genus-3 and genus-4 curves (see Table 2.1 for more details).

According to [Thé03], an identical security level of, for example, 163bit results in the underlying fields $\mathbb{F}_{2^{81}}$, $\mathbb{F}_{2^{57}}$, and $\mathbb{F}_{2^{53}}$ for HEC of genus two, three, and four, respectively. Figures 6.1 and 6.2 compare the implementation results for 160 bits and 180 bits security.

Contrary to common belief, we could show that the performance of a scalar multiplication of ECC, genus-2 HECC, and genus-3 HECC are in the same range. In some cases HECC even outperforms ECC, see Table 6.2.

It is noticeable that with increasing group order the speed of the group operations decreases for certain field sizes. For example, examine the scalar multiplication performance of genus-4 HEC using the fields $\mathbb{F}_{2^{59}}$ and $\mathbb{F}_{2^{63}}$ (two last rows, Table 6.2). Using $\mathbb{F}_{2^{63}}$ results in a slightly better performance. The reason is the dependency of the field operations on the irreducible polynomials. We used trinomials or pentanomials and the groups using trinomials perform relatively better compared to the one using pentanomials.

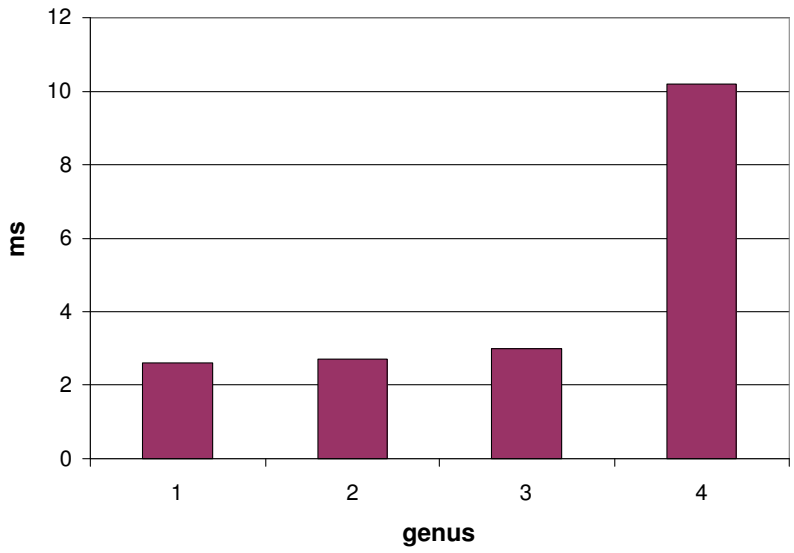


Figure 6.1: Comparison of the scalar multiplication implementation on the Pentium4 @1.8GHz (security $\approx 2^{163}$).

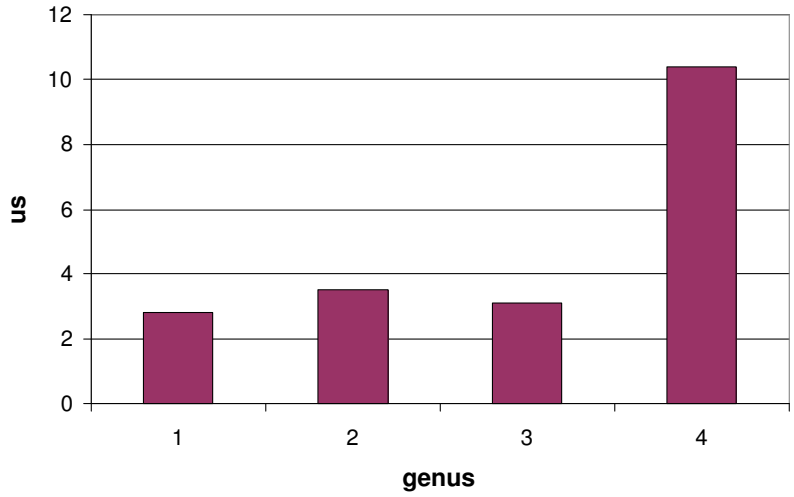


Figure 6.2: Comparison of the scalar multiplication implementation on the Pentium4 @1.8GHz (security $\approx 2^{180}$).

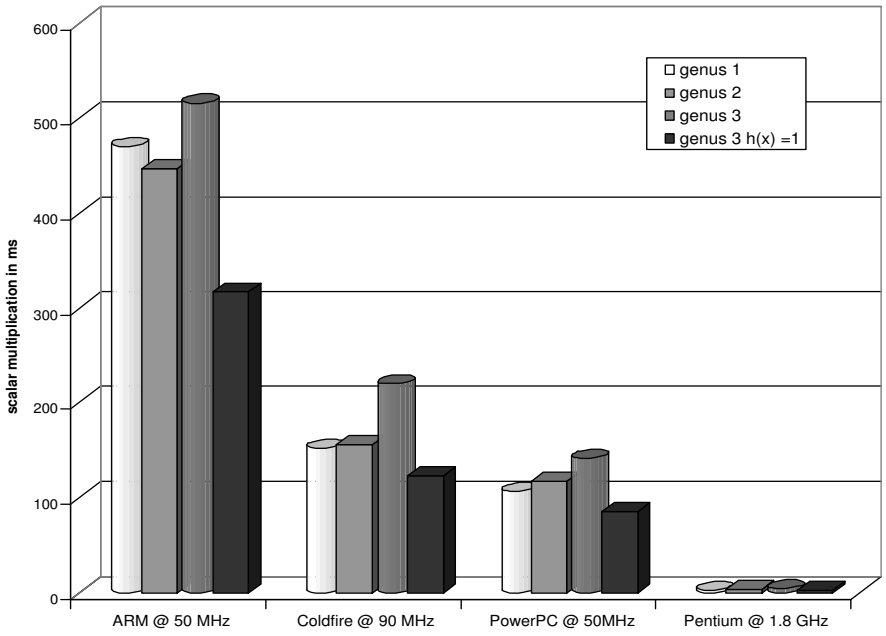
6.3 Hyperelliptic Curve Cryptosystem on Embedded Processors

This section summarizes and analyzes our implementation results on the embedded processors. Most implementation results presented in this section were published in [WPW+04], but in part also appeared in [PWGP03, PWP03, PWP04b, PWP04a]. The emphasis lies on the performance of the different platforms, the comparison of the targeted cryptosystems with different implementation options, and on the influence of the hardware settings. ECC and HECC are implemented using general reduction routines. In addition, we implemented HECC with special (fixed) reduction polynomials to be able to analyze the performance gain.

6.3.1 Implementation Results on Different Embedded Platforms

We implemented ECC and HECC on different embedded platforms with high practical relevance, namely ARM, ColdFire, and PowerPC (Figure 6.3). All timings of the scalar multiplication concerning group orders around 2^{160}, 2^{170}, 2^{180}, and 2^{190} can be found in Table 6.3. For the boards at hand we could achieve the best timings for the HECC implementation on the PowerPC. One scalar multiplication for HECC took 117 ms and 84.9 ms for genus-2 and genus-3 curves, respectively. The scalar multiplication for ECC can be performed fastest on the PowerPC at 50MHz resulting in 106.3 ms.

Figure 6.3: Implementation of ECC and HECC scalar multiplication on different embedded platforms (group order $\approx 2^{160}$).



We want to stress at this point that due to the different hardware architectures of the platforms and the varying board features the actual timings can be quite different, though the processor clock frequency is equal (see Section 6.3.3).

Table 6.3: Timings of the scalar multiplication of ECC and HECC on different embedded platforms (in ms).

group order		ECC	HECC		
			$g = 2$	$g = 3$	$g = 3, h(x) = 1$
$\approx 2^{160}$	ARM @ 50MHz	469.96	446.46	515.46	316.6
	ColdFire @ 90MHz	152.1	155.6	219.4	123.6
	PowerPC @ 50MHz	106.3	117	141.4	84.9
$\approx 2^{170}$	ARM @ 50MHz	397.12	461.36	523.12	321.12
	ColdFire @ 90MHz	132.8	161.5	225.1	126.9
	PowerPC @ 50MHz	94.5	121.2	145.4	87
$\approx 2^{180}$	ARM @ 50MHz	515.95	516.5	577.5	356.99
	ColdFire @ 90MHz	171.7	183.4	246.7	146.2
	PowerPC @ 50MHz	121.8	138.1	160.1	96.8
$\approx 2^{190}$	ARM @ 50MHz	436.01	542.68	581.24	360.24
	ColdFire @ 90MHz	157.8	187.6	258.5	147
	PowerPC @ 50MHz	112.4	141.7	167.8	101.8

6.3.2 Standard versus Special Implementation

There are two major ways of implementing a cryptographic algorithm. One way is to allow all possible input parameters, e.g., arbitrary curves and irreducible polynomials. This form is referred to as standard implementation and is used in server applications or cryptographic libraries. Furthermore, it is sufficient to target specific implementations of algorithms when constrained in memory and processor power (e.g., allowing only standardized curves or even a single fixed curve). The more specific the implementation the higher the efficiency. In this subsection, we focus on the impact of using the specific versus the standard implementation.

Performance of Underlying Field Arithmetic

We implemented the frequently used finite field functions, namely modular multiplication and modular squaring in two different ways. First we used a *standard* implementation with a reduction function capable of handling arbitrary irreducible polynomials. Second,

we used a fixed (*special*) polynomial and therefore had to program separate reduction routines for each of the finite fields used. Table 6.4 shows the timings for multiplication and squaring with different underlying fields using standard and special reduction routines on the ARM microprocessor. Some of the implementations might have potential cryptographical weaknesses according to [Fre98, GHS02], however, we would like to emphasize that the implementation techniques are also applicable to fields with prime extensions, see Page 4.

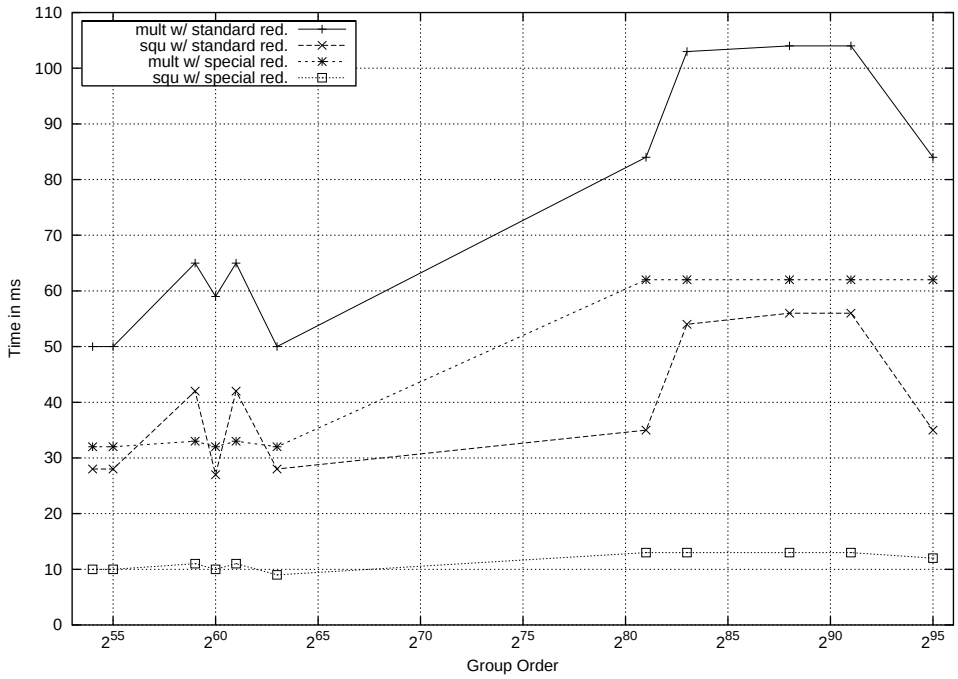
Table 6.4: Influence of special and standard field reduction (all timings in μs, platform: ARM @50MHz).

	general red.		special red.		general / special	
field	mult	squ	mult	squ	mult	squ
2^{54}	50	28	32	10	1.56	2.8
2^{55}	50	28	32	10	1.56	2.8
2^{59}	65	42	33	11	1.97	3.82
2^{60}	59	27	32	10	1.75	2.7
2^{61}	65	42	33	11	1.97	3.82
2^{63}	50	28	32	9	1.56	3.11
2^{81}	84	35	62	13	1.35	2.69
2^{83}	103	54	62	13	1.66	4.15
2^{88}	104	56	62	13	1.68	4.31
2^{91}	104	56	62	13	1.68	4.31
2^{95}	84	35	62	12	1.35	2.92

Analyzing the throughput of the functions the special modular multiplication routine is up to two times faster compared to the standard implementation. In the case of squaring, the gain is even higher and an increase in performance by a factor of 4 can be achieved. The difference in the performance gain relies on the reduction routine, playing a crucial role in the squaring routine.

Figure 6.4 depicts the timings of the modular arithmetic for different fields. The evaluation of this figure leads to the following conclusions:

Figure 6.4: Comparison of standard versus special implementation of the field arithmetic (platform: ARM@50MHz).



1. We implemented two sets of fields: first set considers field sizes smaller than $\mathbb{F}_{2^{63}}$ and the second set consists of field sizes larger than $\mathbb{F}_{2^{81}}$. There is an unusually rise of the speed between these sets of fields. The increase results from the fact that the implementation is targeted for 32-bit processors. The field elements smaller $\mathbb{F}_{2^{63}}$ can be represented with two words, whereas in the case of at least $\mathbb{F}_{2^{63}}$ three words have to be provided.

2. In the specific implementation uses fixed parameter, resulting in a nearly monotonic slope for a constant number of words. The standard implementation depends heavily on the chosen irreducible polynomial which can be seen from the non-monotonic slope of the graphs. In our implementation we used trinomials and pentanomials. The latter case applied when there were no irreducible trinomials available. For example in the case of the underlying field $\mathbb{F}_{2^{55}}$, we used a trinomial,

while a pentanomial was used for the field $\mathbb{F}_{2^{59}}$. The larger overhead for a standard routine using a pentanomial instead of a trinomial leads to a decrease in speed for multiplication and squaring.

Influence on the Scalar Multiplication

Table 6.5 shows how the different implementations of the underlying library influence the performance of the HECC. For genus-2 curves, the ratio of the standard implementation to that of the special implementation is in the range of 1.27 to 1.48. In the case of genus-3 curves, scalar multiplication can be accelerated by almost 50%. The performance gain is not as huge as for the plain field operations because of additional overhead and other underlying functions (e.g. inversion) that are not optimized.

Table 6.5: Influence of different reduction routines on HECC scalar multiplication (all timings in ms, platform: ARM @50MHz).

	standard reduction			special reduction			standard / special		
group	g=2	g=3		g=2	g=3		g=2	g=3	
order		h(x)=1			h(x)=1			h(x)=1	
$\approx 2^{160}$	565.97	449.62	749.36	446.46	316.6	515.46	1.27	1.42	1.45
$\approx 2^{170}$	682.86	454.81	758.72	461.36	321.12	523.12	1.48	1.42	1.45
$\approx 2^{180}$	766.64	504.75	837.71	516.5	356.99	577.5	1.48	1.41	1.45
$\approx 2^{190}$	681.36	513.66	852.15	542.68	360.24	581.24	1.26	1.43	1.47

6.3.3 Influence of Cache

The performance of a cryptographic system depends a lot on the processor and on the available resources of the board. In this subsection, we analyze the influence of different cache settings. We choose the PowerPC board to analysis these settings. The PowerPC provides two different caches, namely one to store data and one for instructions. Furthermore, we can disabling the pipelining on the PowerPC referred to as *non-serialized*

mode and *serialized* mode achieving full pipelining.

Table 6.6: Influence of different cache options on ECC and HECC performance (all timings in ms, platform: PowerPC @50MHz, see Table 6.7 for ratios).

		cache, serialized			no cache	
	group order	data + instruction	instruction	data	serialized	not serialized
ECC	2^{163}	106.4	271.9	626.2	828.1	1249
	2^{167}	94.5	241	553.5	732.4	1062.8
	2^{179}	122	311.4	713.7	944.6	1371.4
	2^{191}	112.5	286.3	659	871.7	1264.7
HECC, g=2	2^{162}	117	272.8	695.8	886.1	1293
	2^{166}	121.2	280.7	722.2	916.6	1339
	2^{176}	130.5	300.9	776.2	984.9	1438
	2^{182}	138.1	317.9	821.8	1042	1521
	2^{190}	141.7	328.8	841.7	1071	1562

Table 6.6 shows the influence of the cache targeting ECC and genus-2 HECC implementations on the PowerPC. In the left part of the table we present our results targeting different cache settings using full pipeline mode. The right part of Table 6.6 compares the performance of the scalar multiplication providing pipelining (serialized mode) and non-pipelining (Not serialized mode).

Normalizing these timings with respect to the obtained execution times with disabled cache leads to the ratios stated in Table 6.7. The ratios on the left part of the table show the improvement of adding cache compared to having no cache. The right part of the table indicates the speed-up of using pipeline.

It is noticeable that there is almost no difference in the impact of the cache setting for ECC and HECC. The data cache is advantageous when intensive memory access is necessary. The utilization of the instruction cache dominates in projects consisting of small functions which get called frequently. In our case, the latter applies: the code size is relatively small and the functions called most frequently consist of only few commands. This is confirmed by the timings on the PowerPC. It can be seen that the performance

Table 6.7: Ratios of the ECC and HECC scalar multiplication using different cache settings (platform: PowerPC @50MHz, see Table 6.6 for the timings).

	group order	no cache / data + instruction	no cache / instruction	no cache / data	no serialized / serialized (no cache)
ECC	2^{163}	7.78	3.05	1.32	1.51
	2^{167}	7.75	3.04	1.32	1.45
	2^{179}	7.74	3.03	1.32	1.45
	2^{191}	7.75	3.04	1.32	1.45
HECC, g=2	2^{162}	7.57	3.25	1.27	1.46
	2^{166}	7,56	3.27	1.27	1.46
	2^{176}	7.55	3.27	1.27	1.46
	2^{182}	7.55	3.28	1.27	1.46
	2^{190}	7.56	3.26	1.27	1.46

increases by a factor of 1.3 when using only data cache and by a factor of 3.3 when only using instruction cache. The computation of a scalar multiplication is about a factor of 7.7 faster if using instruction and data cache. Since we are using a 16KByte cache, the most relevant subroutines are permanently cached. In addition, the serialized mode compared to the non-serialized mode can speed up the design by almost 50%.

Hence, we advise using at least an instruction cache, or even better, both kinds of cache when running ECC or HECC.

6.3.4 Comparing the Performance of the Different Cryptosystems

In this section we compare ECC and HECC on the ARM microprocessor. In order to provide a fair comparison, we consider:

1. the same security level (analogous to Section 6.2, where we did the same using the Pentium processor);

2. use the most efficient group operations for all cryptosystems (as proposed in Table 4.5); and

3. implement general routines for the field arithmetic.

We enlarge the field sizes as proposed by the findings in [Thé03] for equal security. Hence, we have to consider underlying fields $\mathbb{F}_{2^{81}}$, $\mathbb{F}_{2^{57}}$, and $\mathbb{F}_{2^{53}}$ for HEC of genus two, three, and four, respectively, considering 163bit security. We proceeded in the same way with the security level of about 180bit.

Table 6.8 presents the our timings for HECC on the ARMulator. Figure 6.5 and Figure 6.6 show, that the performance of a scalar multiplication of ECC, genus-2 HECC, and genus-3 HECC are in the same range. Note that some of the selected underlying fields might have some cryptographical weaknesses and the reader is referred to Section 2 for more details.

Considering genus-4 HEC, one notices that this cryptosystem is very inefficient compared to ECC, genus-2, and genus-3 HECC. Furthermore one notices that in some cases HECC even outperforms ECC, see Table 6.8. The reason for the non linear performance decrease with larger fields, is the use of different reduction polynomials (trinomial and pentanomials).

Hyperelliptic curve cryptosystems show very good performance on embedded processors and are therefore well suited for applications in constrained environments.

To speed up the presented implementation results one can use special reduction routines, as investigated in earlier sections. Using the fastest group operations known today, a genus-2 scalar multiplication for group orders of $\approx 2^{160}$ to $\approx 2^{190}$ would take 69ms to 86ms.

6.3.5 Theoretical Metric Applied to the ARM Implementation

In this subsection, we show how to use the theoretical metric introduced in Section 5 for our ARM implementation in the previous section.

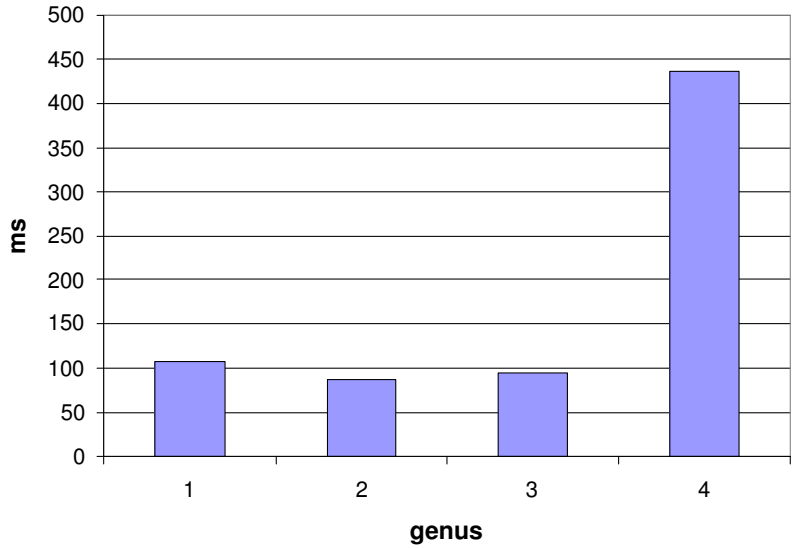


Figure 6.5: Comparison of the scalar multiplication implementation on the ARMulator ARM7TDMI @80MHz (security $\approx 2^{163}$).

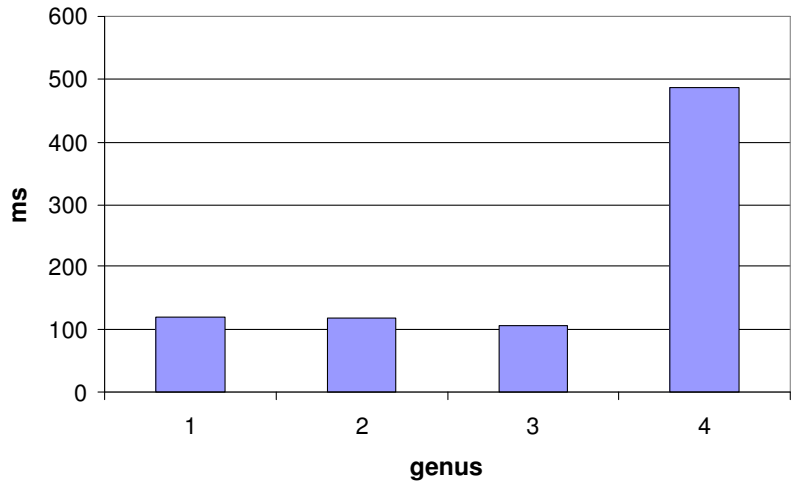


Figure 6.6: Comparison of the scalar multiplication implementation on the ARMulator ARM7TDMI @80MHz (security $\approx 2^{180}$).

Table 6.8: Timings for ECC and HECC on the ARMulator ARM7TDMI @80MHz.

Genus	Field	Group order	Group addition in μs	Group doubling in μs	Scalar. mult. in ms
1	$\mathbb{F}_{2^{163}}$	2^{163}	746	406	108
	$\mathbb{F}_{2^{167}}$	2^{167}	579	342	90
	$\mathbb{F}_{2^{179}}$	2^{179}	720	451	120
	$\mathbb{F}_{2^{191}}$	2^{191}	598	358	100
2	$\mathbb{F}_{2^{81}}$	2^{162}	600	376	87
	$\mathbb{F}_{2^{83}}$	2^{166}	715	449	105
	$\mathbb{F}_{2^{88}}$	2^{176}	732	463	114
	$\mathbb{F}_{2^{91}}$	2^{182}	736	468	119
	$\mathbb{F}_{2^{95}}$	2^{190}	623	399	107
3	$\mathbb{F}_{2^{54}}$	2^{162}	914	317	90
	$\mathbb{F}_{2^{55}}$	2^{165}	917	319	91
	$\mathbb{F}_{2^{57}}$	2^{171}	918	321	94
	$\mathbb{F}_{2^{59}}$	2^{177}	1180	415	126
	$\mathbb{F}_{2^{60}}$	2^{180}	921	324	100
	$\mathbb{F}_{2^{61}}$	2^{183}	1183	417	130
	$\mathbb{F}_{2^{63}}$	2^{189}	925	329	106
4	$\mathbb{F}_{2^{40}}$	2^{160}	2106	1176	272
	$\mathbb{F}_{2^{41}}$	2^{164}	1748	980	222
	$\mathbb{F}_{2^{44}}$	2^{176}	1756	987	250
	$\mathbb{F}_{2^{46}}$	2^{184}	1760	993	262
	$\mathbb{F}_{2^{47}}$	2^{188}	1763	995	268
	$\mathbb{F}_{2^{53}}$	2^{212}	2561	1442	437
	$\mathbb{F}_{2^{59}}$	2^{236}	2575	1456	487
	$\mathbb{F}_{2^{63}}$	2^{252}	2021	1154	413

Table 6.9 introduces the timings of the field multiplications, field inversion, and the computed MI-ratios. To determine the most efficient cryptosystem theoretically based on the timings given in Table 6.9, one can either use Figure 5.1 to find the number of AOPS or calculate the necessary number using Table 5.2. In order to be able to do so, one needs the MI-ratios for the different underlying fields used by the cryptosystems. Some of the implementations might have potential cryptographical weaknesses according to [Fre98,GHS02], however, we would like to emphasize that the implementation techniques are also applicable to fields with prime extensions, see Page 4.

Table 6.9: Timings of the field library and corresponding MI-ratios (in μsec, ARM @80MHz).

Field	Inversion	Multiplication	MI-ratio
$\mathbb{F}_{2^{163}}$	$456.3\mu s$	$57.9\mu s$	$j = 7.9$
$\mathbb{F}_{2^{81}}$	$119.7\mu s$	$19.2\mu s$	$l = 6.2$
$\mathbb{F}_{2^{57}}$	$66.2\mu s$	$11.4\mu s$	$m = 5.8$
$\mathbb{F}_{2^{53}}$	$66.4\mu s$	$15.1\mu s$	$r = 4.4$

Considering a finite field $\mathbb{F}_{2^{163}}$ for an ECC using Jacobian projective coordinates, 531,840 AOPS are needed to calculate one scalar multiplication. HECC of genus 2 with the underlying field $\mathbb{F}_{2^{81}}$ will take 497,837 AOPS, and genus-3 over $\mathbb{F}_{2^{57}}$ requires 497,174 AOPS. Finally, we find that genus-4 curves ($\mathbb{F}_{2^{53}}$) need 1,600,403 AOPS.

Thus, we expect HECC of genus-2 to be a factor of 1.06 and genus-3 HECC a factor of 1.07 faster than ECC. Genus-4 HECC is expected to be 3.01-times slower than ECC.

Using the timings presented in Table 6.8, we can compute the performance difference between the cryptosystems accordingly. The implementation timings for a scalar multiplication of genus-2 curves over $\mathbb{F}_{2^{81}}$, genus-3 curves over $\mathbb{F}_{2^{57}}$, and genus-4 curves over $\mathbb{F}_{2^{53}}$ are compared with the performance of the ECC scalar multiplication over $\mathbb{F}_{2^{163}}$. HECC of genus 2 is a factor of 1.23, genus-3 a factor of 1.15 faster, and HECC of genus-4 is a factor of 4.05 slower than ECC.

In Table 6.10 we summarized all the given numbers above and show the deviation of our implementation and the theoretical findings. The deviation of is at most 26%. Note, that we normalized all the given number with respect to the ECC performance.

Similarly, we can provide the theoretical metric for any security level. In the following we show another example using the group order $\approx 2^{180}$. Table 6.11 shows the corresponding MI-rations and Table 6.12.

Thus, we can conclude that our theoretical estimates were quite accurate.

Table 6.10: Derivation of our implementation and the theoretical matrix (security level $\approx 2^{160}$).

	$\frac{genus-2}{ECC}$	$\frac{genus-3}{ECC}$	$\frac{genus-4}{ECC}$
theory	0.94	0.93	3.01
ARM	0.81	0.87	4.05
deviation	14%	6%	26%

Table 6.11: Timings of the field library and corresponding MI-ratios (in μsec, ARM @80MHz).

Field	Inversion	Multiplication	MI-ratio
$\mathbb{F}_{2^{179}}$	$428.0\mu s$	$63.5\mu s$	$j = 6.7$
$\mathbb{F}_{2^{91}}$	$150.6\mu s$	$24.2\mu s$	$l = 6.2$
$\mathbb{F}_{2^{63}}$	$69.4\mu s$	$11.4\mu s$	$m = 6.1$
$\mathbb{F}_{2^{59}}$	$71.1\mu s$	$15.1\mu s$	$r = 4.7$

Table 6.12: Derivation of our implementation and the theoretical matrix (security level $\approx 2^{180}$).

	$\frac{genus-2}{ECC}$	$\frac{genus-3}{ECC}$	$\frac{genus-4}{ECC}$
theory	0.93	0.92	2.96
ARM	0.99	0.88	4.06
Derivation	6%	4%	27%

6.3.6 Low Cost Security

In many low cost embedded applications lower security margins are adequate. In practice, if a group order of 2^{128} is sufficient, the operations can be performed with an operand length of 32-bit in the case of genus-4 HEC. Thus, the underlying field operations can be implemented very efficiently using one word arithmetic if working with 32-bit microprocessors (e.g., ARM). For genus-2 and genus-3 curves however, we need to perform the underlying field operation using two words of a 32-bit processor. It is important to point out that the small field sizes and the resulting short operand size of HECC compared to other cryptosystems makes HECC specially promising for the use

in embedded environments.

Security of 128-bit HECC

In [LV01], Lenstra and Verheul argue that for commercial security in the year 2004, 138-bit ECC should be considered. Furthermore, the authors state that ECC using 138-bit keys are as secure as 1108-bit keys for RSA or DSS. This notion of commercial security is based on the hypothesis that a 56-bit block cipher offered adequate security in 1982.

Recently, Certicom announced that Chris Monico and his team of mathematicians from the University of Notre Dame solved the Certicom ECCp-109 Challenge [Cor02]. The underlying curve was randomly generated over prime fields p of order 2^{109}. The challenge was solved by utilizing 10,000 computers running 549 days. Note that in our low security HECC implementations we used a group order of 2^{128}. Therefore, HECC is by a factor of $\sqrt{2^{19}} \approx 724$ harder to break than the ECCp-109 challenge which would mean 1089 years, assuming the computational resources of the ECCp-109 attack.

It is also worth to point out that the factorization of the 512-bit RSA challenge took only about 2% of the time required to break the ECC2K-108 [Cor02] challenge (or to break DES). This implies that ECC or HECC in groups of order 2^{128} offer far more security than a 512-bit RSA system. Nevertheless, RSA with a 512-bit key is still in use, for example, in fielded smart card applications.

Remark: we did not considers the attacks on HEC and the underlying fields proposed in [Fre98, GHS02, Thé03] as a threat for low cost application. Low cost security application may require to secure a message only for some minutes or a few hours. The proposed attack does not appear to be a threat because of the high complexity and the huge processor power and memory necessary.

Low Cost Implementation Results

In contrast to the facts mentioned in Section 6.3.4, there is a benefit of using genus-4 HECC over a 32-bit finite field. The processor word is optimally utilized and the cryptosystem does not need additional multi-precision arithmetic. Analyzing the results for a group order of around 2^{128} as presented in Table 6.13, genus-4 low cost implementations are in the same range as genus-2 and genus-3 curves (contrary to the results in Section 6.3.4).

Table 6.13: Timings on the ARM7TDMI@80MHz and Pentium4 @1.8GHz for group order $\approx 2^{128}$ (explicit formulae).

ARMulator ARM7TDMI@80MHz					
Genus	Field	Group order	Group addition in μs	Group doubling in μs	Scalar. mult. in ms
2	$\mathbb{F}_{2^{63}}$	2^{126}	538	326	60
3	$\mathbb{F}_{2^{43}}$	2^{129}	966	327	75
4	$\mathbb{F}_{2^{32}}$	2^{128}	914	524	97
Pentium4@1,8GHz					
2	$\mathbb{F}_{2^{63}}$	2^{126}	16.5	10.2	1.93
3	$\mathbb{F}_{2^{43}}$	2^{129}	25.5	9.0	2.15
4	$\mathbb{F}_{2^{32}}$	2^{128}	21.7	13.3	2.57

As a result of this comparison, we suggest the use of genus-4 HECC for applications with short term security needs, if one is not worried about the attacks mentioned before. Despite the high number of required field operations compared to HEC with genus $g \leq 3$, group operations on genus-4 HEC are easier to implement. This relies on the fact that field arithmetic based on operands of length no longer than 32 bits are relatively simple.

6.4 Summary of the Software Implementation

Software performance depends on the specific algorithms, on the underlying implementation, and the processor type used. The contribution of this chapter was the implemen-

tation of ECC and HECC on general purpose processors and embedded processors using the same methodology. The same methodology allows a relative fair comparison between the two cryptosystems on the given platforms. In the case of HECC we considered curves with genus two, three, and four. Our analysis targets the different cryptosystems with respect to certain software settings and different hardware components.

Considering the same security level, we showed that the performance of a scalar multiplication of ECC, genus-2 HECC, and genus-3 HECC are in the same range. Considering genus-4 HEC, one notices that this cryptosystem is very inefficient compared to ECC, genus-2 and genus-3 HECC. Furthermore one notices that in some cases HECC even outperforms ECC. Considering lower security margins, we were able to implement the underlying field operations efficiently for genus-4 HEC using 32-bit microprocessors (e.g., ARM). However, one should keep in mind the security limitations of genus-4 HECC.

At this point we want to clarify that the performance is based on certain curve parameters and implementation options. Changing these parameters could result in different performance numbers. Furthermore, one should be aware of the fact that processor speeds are increasing rapidly and that one can achieve much higher speeds in the future.

We conclude from this chapter that ECC and HECC show similar performance on general purpose processors as well as on embedded processors. Hence, these cryptosystems are well suited for applications in constrained environments.

7 Finding an Optimized Parallel Architectures for HECC

Most general purpose processors are designed to execute instructions essentially sequentially and the instruction set is bound to the word length of the processor (e.g. 32-bits). Hence, there are only limited option for parallelizing the execution of cryptographic algorithms. In custom-built hardware the designer can choose to perform instruction in parallel and to process larger operands. Exploiting this parallelism can result in a major performance increase. Cryptographic hardware accelerators are built exactly for this reason, i.e., allowing hardware parallelism, as they can provide a high throughput for the arithmetic-intensive public-key cryptographic primitives. An example are high-end smart cards, where a cryptographic coprocessor takes over all the expensive (area and time) computations. However, application areas of cryptographic chips exceed far beyond smart card applications. These hardware accelerators are used for web servers, for Virtual Private Networks, for wireless gateways, for ATM machines, and for secure satellite communications, to name only a few.

In this chapter we present optimized parallel architecture for a HECC considering the most recent explicit formulae to compute group operations. This is achieved by theoretically evaluating a variety of different architectures. In order to do so, we wrote a software tool capable of scheduling the necessary operations. The parallelization of

the scalar multiplication of HECC was investigated at the following three levels: the field operation level, the group operation level, and the scalar multiplication level. At the field operation level we used multipliers that handle different numbers of bits in parallel. At the group operation level we used different numbers of field multipliers. Our investigation of the parallelism reachable for the scalar multiplication level was achieved by overlapping the computation of consecutive group operations, i.e., by pipelining the group operations.

In [MS03] the authors proposed, for the first time, a parallelization of the explicit group operation of HECC. The authors developed a general methodology for obtaining parallel algorithms. The methodology guarantees that the obtained parallel version requires a minimum number of rounds. They show that for the inversion free arithmetic using 4, 8 and 12 multipliers in parallel, scalar multiplication can be carried out in 27, 14 and 10 parallel rounds, respectively. When using affine coordinates and 8 multipliers it can be performed in 11 rounds, including an inversion round.

Note that for an effective implementation it is impractical to use so many multipliers in parallel as stated in [MS03]. The work at hand attempts to consider not only the minimum number of rounds (speed), but also the necessary devices (area) as well as practical applications.

In the following, we focus on the theoretical analysis of the parallelism of HECC and present the optimal architecture for HECC[1]. We first present the methodology leading to our results. Next, we present our results of parallelizing system using affine and projective coordinates. Afterwards, we compare and analyze the two systems in order to find the best architecture realizing HECC. We end this chapter with a short summary and outlook.

[1]This is joint work with Dr. Guido Bertoni and Prof. Luca Breveglieri from the Politecnico di Milano. Guido Bertoni did the implementation of the scheduler in C language.

7.1 Methodology

The discussion in this chapter are based on a standard architecture, see Figure 7.1, consisting of a 3-bus loop scheme connecting a set of function units with a register file. A control unit drives the various function units and the register file. The register file stores temporary intermediate values and final results. The size of each register is the dimension of the field, namely 81 bits. The register file has two output ports to feed the operators to the function units and one input port to receive the result. The processor allows to load two elements and store one element at any clock cycle. This guarantees feasibility and ease of implementation. However, at any given clock cycle only one field operation can start. If the operation is unary, such as inversion, one input bus remains idle.

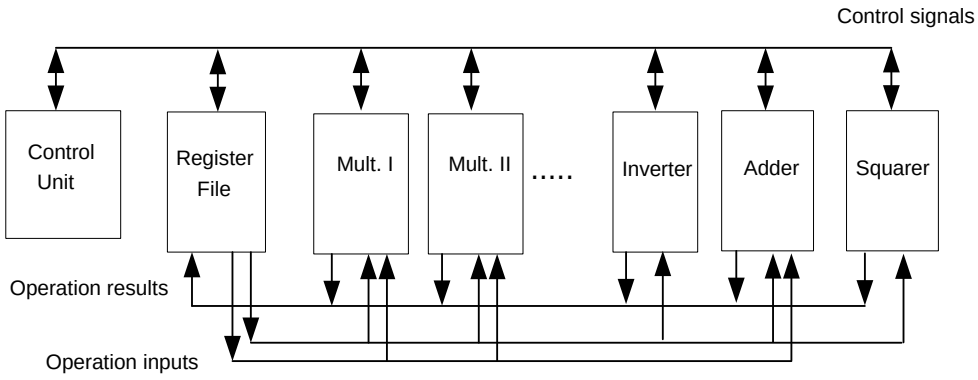


Figure 7.1: Architecture of the HECC coprocessor.

The following list is a summary of how we implemented the field arithmetic operations ($F(x)$ denotes the irreducible polynomial of the field $\mathbb{F}$):

- Addition. Adding two elements requires the modulo 2 sum of the coefficients of the field elements.

- Squaring. Squaring a field element $A = \sum_{i=0}^{m-1} a_i x^i$ is ruled by the following equation: $A^2 \equiv \sum_{i=0}^{m-1} a_i x^{2i} \bmod F(x)$; further details can be found in [OP00].

- Multiplication. We decided to use digit multipliers, introduced in [SP97] for fields of type $\mathbb{F}_{2^m}$. This type of multiplier allows a trade-off between speed, area and power consumption. It works by processing several multiplicand coefficients at the same time. The number of coefficients processed in parallel is the digit-size $D \geq 1$. Given the parameter D, we denote by $d = \lceil m/D \rceil$ the total number of digits in a polynomial of degree $m-1$. Hence, $C \equiv AB \bmod F(x) \equiv A \sum_{i=0}^{d-1} B_i x^{Di} \bmod F(x)$.

- Inversion. It is computed using the algorithm proposed in [BCH93], which is based on a modification of Euclid's algorithm for computing the GCD of two polynomials. The asymptotic complexity is linear with the modulus both in time and area.

In Table 7.1 we give the area and latency for each arithmetic function unit we used. The given estimates assume 2-input gates and optimal field generator polynomials $F(x) = x^m + \sum_{i=0}^{t} f_i x^i$, where $m - t \geq D$.

One observes that the gate-consuming function units are multiplier and inverter, while in comparison the area used for the adder and squarer are negligible. In case of the multiplier and the inverter the number of XOR gates and AND gates are the same. Hence, their ratios are the same and justifies why in our estimate we identify XOR and AND gates.

Now, we describe briefly our approach to design the best suited architecture.

1. Input. First we evaluated the most recent achievements regarding the group operation of HECC. The given formulae were then prepared for the scheduler.

Table 7.1: Area and time complexity of the arithmetic operations (underlying field $\mathbb{F}_{2^m}$, irreducible polynomial $F(x) = x^m + \sum_{i=0}^{t} f_i x^i$, where $m - t \geq D$).

	Area		Latency [clock cycles]
		Total number of gates	
Add	$[m]$ XOR	$[m]$	1
Sqr [OP00]	$[m + t + 1]$ XOR	$[m + t + 1]$	1
Mul [SP97]	$[D \cdot m]$ AND & $[D \cdot m]$ XOR	$[2Dm]$	$\lceil m/D \rceil$
Inv [BCH93]	$[6 \cdot m + log_2 m]$ AND & $[6 \cdot m + log_2 m]$ XOR	$[2(6m + log_2 m)]$	$2 \cdot m$

2. Scheduler. Our own software library, especially developed to schedule the HECC group operations, is the heart of our methodology. The scheduler is based on the method known as Operation Scheduling [Gov03] and works accordingly to the As Soon As Possible (ASAP) policy. There is a list of operations that should be executed by the architecture. The scheduler takes one operation at a time and searches for the earliest time slot where the operation can be executed. It is constrained by the number of available resources and by the different times required to execute each operation. The same methodology is used by programming language compilers for scheduling machine instructions.

   Schedulers fall into two broad families: unconstrained or resource limited. We choose to upper bound the number of resources for busses and arithmetic units, while that of registers is unbounded. Once the operations are scheduled, we count the number of live registers and compute the overall register usage.

   It should be noted that our methodology is heuristic and does not necessarily guaranty optimal results, but it is quite natural, easy to design and efficient. In order to reach the globally optimal scheduling it is necessary to use other methods instead, see [Gov03].

   The scheduler has the following input parameters:

- HECC formulae (i.e., the sequence of operations to schedule).
- Latencies of the function units.
- Number of multiplication units.
- Different digit-sizes of the multiplication units.
- Properties (latency and size) of the busses.
- Memory access time (latency for register read / write).

3. Testing. The results of the scheduler were tested on test vectors. In order to do, so we implemented HECC group operations with the NTL library [Sho01].

4. Analysis. The results were analyzed and, if needed, the underlying architecture was changed in order to find a better structure for the parallel coprocessor.

We investigated different designs in order to find an optimal architecture. We varied the number of multipliers (one, two, three, or four), as well as their digit-size D ($D = 2, 4, 8, 16$, or 32). Representing the parallelism on the field operation level and group operation level, respectively. Furthermore, we investigated the parallelism on the scalar multiplication level, by computing two consecutive group operations in parallel. We will refer to this parallel computation in the following as *overlapping*. We timed the overlapping of two group operations as the difference between a) the time when the last field operation of the former group operation ends execution and b) the time when the first field operation of the latter starts. This is possible because the outputs of the first group operation, namely four field elements in the case of genus-2 HEC, are neither produced at the same time nor necessary to start the next group operation. Hence we start the second group operation after computing one field element. If one considers overlapping and uses the double-and-add algorithm for the scalar multiplication the following consecutive group operations should be scheduled: addition after doubling, doubling after doubling, and doubling after addition.

Our design goal for the optimal architecture is to implement a simple controller computing group operations with a fixed execution order. Hence, we look at a static schedule. The alternative would be to implement a finite state machine executing the schedule directly, controlling the availability of resources and deciding which operation should be executed. This solution is feasible but we consider it as too complex and expensive compared to a simple controller executing the operations in a fixed order.

We supposed that all the different modules of the system work always at the same frequency. This is a worst-case assumption. In fact, usually the complexity of the multiplication unit dominates the frequency, and a smaller digit-size will yield a higher clock frequency and thus will speed-up the system. Furthermore we omitted the register file area in the estimation. We decided to do so after noticing that the required number of registers is almost the same in all the configurations, and that the area consumed by a register can vary considerably depending on the implementation technology.

Our results are based on the following considerations: i) we choose a set of keys to schedule the operations; ii) we scheduled group addition and doubling, accordingly to the values of the key; iii) we considered a long sequence of concatenated group operations. For the latency evaluation of the scalar multiplication kD reported, we examined it in an average case. This means that 80 and 160 additions and doublings were performed, respectively. Half of the 160 doublings are computed after another doubling and half after an addition.

The group operations were implemented using the up-to-date fastest formulae for genus-2 HEC, as are given in Table 4.5. Hence, the curve parameters are given as $h = x$ and $f = x^5 + f_1 x + f_0$ and we used the underlying finite field $\mathrm{GF}(2^{81})$. The implementations of the field $\mathrm{GF}(2^{81})$ might have potential cryptographical weaknesses according to [Fre98, GHS02], however, we would like to emphasize that the implementation techniques are also applicable to fields with prime extensions, see Page 4.

We changed for the different architecture options the processing power by adding resources. Increasing the processing power yields a speed up of the group operations, but also causes a growth in area. Thus there must exist an optimal architecture, where the area-time product is minimal.

7.2 Analysis of Parallel Architectures Using Affine Coordinates

In this section we present the results for the different architectures using affine coordinates. Results of this subsection were partly published in [BBWP04a, BBWP04b]. Genus-2 HEC defined over a field of characteristic 2 was used and the group operations represented in Tables A.1 and A.3 were used. We are going to evaluate different design options and examine the different levels of parallelism. Our software tool is capable of scheduling the necessary operations, resulting in an optimal architecture with respect to area and speed.

The various system options need between 18 and 20 registers in the best and worst case, respectively. If a designer wants to lower register usage, he/she can trade the number of registers for additional latency. In order to do so, the designer should avoid overlapping and start a group operation only when the previous has finished. We noted that a single group operation uses from 8 to 10 registers, while the maximum register number is reached when two group operations overlap.

7.2.1 Parallelism on the Scalar Multiplication Level

In the usual cases the genus-2 HEC group operation outputs one polynomial of degree two and one monic polynomial of degree three. Hence the output consists of four coefficients,

namely four field elements. They are neither produced at the same time nor are all necessary to start the next group operation. This means that when one field element (one coefficient) is computed, it can be used by the next group operation. Hence, we overlap the computation of two consecutive group operations.

Table 7.2: Overlapping of doubling after addition(in clock cycles, affine coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	333	172	169	140
4	173	92	89	80
8	93	48	48	48
16	50	30	33	33

Table 7.3: Overlapping of doubling after doubling (in clock cycles, affine coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	214	96	96	96
4	114	56	56	56
8	64	36	36	36
16	39	26	26	26

Table 7.4: Overlapping of addition after doubling (in clock cycles, affine coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	214	96	96	96
4	114	56	56	56
8	64	36	36	36
16	39	26	26	26

Table 7.2 shows the overlapping time interval for doubling after addition. Overlapping decreases as the speed and the number of multipliers increase. Similar behaviors have been observed in the other two cases: doubling after doubling (Table 7.3) and doubling followed by an addition (Table 7.4). This decrease is due to the increase of the parallelism

in the tail of the operation. In the best scenario we were able to compute the two operations for 333 clock cycles in parallel (Table 7.2, first cell).

7.2.2 Performances of the Group Operations and Scalar Multiplication

Tables 7.5 and 7.6 show the clock cycles necessary for performing a group addition and group doubling, respectively, for different system configurations. Table 7.7 presents the latency numbers for the scalar multiplication.

We examined all different cases of consecutive group operations, in order to determine whether we could schedule them in a way to gain speed and to achieve a higher hardware utilization. Our results show that addition is always scheduled in the same way. In the case of doubling, one should use two different ways to execute it depending on the previous operation. We have to allow negligible extra hardware for the controller to decide which option to choose. Thus, we get two latency numbers for the doubling operation. The leftmost timing in each cell of Table 7.6 is the number of clock cycles necessary to compute doubling after doubling. The rightmost one instead is the time latency of doubling after addition.

Table 7.5: Latency of group addition (in clock cycles, affine coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	1,259	739	664	635
4	739	479	444	435
8	479	349	335	335
16	356	289	288	288

Considering only the performance, one concludes from Tables 7.5 and 7.6 that the design option using one inverter, two multipliers ($D = 16$), one adder and one squarer

Table 7.6: Latency of group doubling after doubling / after addition (in clock cycles, affine coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	724 / 846	486 / 560	458 / 490	458 / 484
4	464 / 526	346 / 380	338 / 350	338 / 344
8	334 / 366	278 / 286	278 / 279	278 / 279
16	274 / 284	248 / 247	248 / 250	248 / 250

is preferable. The architecture can perform group doubling and addition in 289 and 248 clock cycles, respectively. Whereas, when analyzing the scalar multiplication number (Table 7.7), one would chose the architecture using three multipliers. In this setup the scalar multiplication is performed in $56,139$ clock cycles.

Table 7.7: Latency of the scalar multiplication (in clock cycles, group order $\approx 2^{160}$, affine coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	$1659,87$	$113,948$	$100,345$	$99,836$
4	$106,527$	$80,228$	$74,625$	$74,136$
8	$76,797$	$63,524$	$61,844$	$61,844$
16	$62,969$	$56,216$	$56,139$	$56,139$

One can see that by increasing the digit-size and the number of multipliers, the time necessary to execute a group operation decreases, as expected. For group addition, our results show that in some cases the performance does not increase, when augmenting the resources of the system. For example, focusing on the speed of addition, using three rather than four digit-size multipliers with $D = 8$ (Table 7.5, third row), does not result in any performance difference. The same behavior can be observed for 2, 3 and 4 multipliers with digit-size $D = 16$. The reason for this behavior is that the structure of the group operation does not allow additional parallelism. When focusing on doubling, there is almost no performance gain in moving from 3 to 4 multipliers (Table 7.6, the

two rightmost columns). Additionally, there is no performance gain in providing 3 or 4 multipliers of digit-size $D = 8$ or $D = 16$ instead of two. Similar facts hold for the scalar multiplication latency (Table 7.7). Hence, we do not gain any performance when changing from three to four multipliers for all genera. Even adding the third multiplier increases the performance only slightly or not at all.

In an ideal scenario all components, but most importantly the multipliers should be used uniformly. The reason being resource usage and side channel attacks. Hence, when examining our results for the group operations and the scalar multiplication, we conclude that the third and fourth multiplier, and in some cases also the second one, are used very infrequently. Hence, for most applications it will be unreasonable to provide these extra hardware units.

Note that the ASAP scheduling policy used in our software tool does not guaranty an optimal solution; as discussed in Section 7.1. Analysis of the performance of doubling shows that for $D = 16$ the speed drops slightly when increasing the hardware from two to three multipliers.

7.2.3 Area-Time Product

In Table 7.8 we used the normalized area-time product to find the optimal architecture. The figures are derived from those of Table 7.1, where $m = 81$ bits. The optimal architecture will achieve the highest throughput consuming the smallest area (contrary to some traditional cryptographic implementations, where only best performance was evaluated). Table 7.8 shows that the architecture using one inversion, one multiplication ($D = 8$), one addition and one squaring achieves the best area-time product. Note, that there are many architecture options that have very similar area-time product.

Table 7.8: Normalized area-time product (group order $\approx 2^{160}$, affine coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	1.3552	1.1148	1.1442	1.3000
4	1.0422	1.0447	1.2133	1.4454
8	1	1.2385	1.6063	2.0067
16	1.2277	1.8241	2.5487	3.2758

7.3 Analysis of Parallel Architectures Using Projective Coordinates

In this section we are analyzing the parallelism of HECC using projective coordinates. We target genus-2 HECs defined over a field of characteristic 2 using the newly derived group operations over projective coordinates, as reported in Tables A.15 and A.16.

All the results presented here target systems directly based on projective coordinates and, therefore, coordinate conversion is unnecessary. For applications where conversions to and from affine coordinates are necessary, discussion and figures will be provided at the end of Section 7.4.

All the considered system configurations require 21 registers for storing temporary values, where each register stores a field element of 81 bits. One could reduce the number of registers at the cost of some additional latency by avoiding the overlapping of two consecutive group operations, exactly as in the case where affine coordinates were used.

7.3.1 Parallelism on the Scalar Multiplication Level

We show the degree of parallelization considering consecutive group operations (parallelism at the scalar multiplication level). When implementing scalar multiplication

using the double-and-add algorithm, one has to evaluate the parallelization of three different cases: i) addition after doubling, ii) doubling after doubling and iii) doubling after addition.

Table 7.9: Overlapping of doubling after doubling (in clock cycles, projective coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	167	85	85	91
4	87	45	30	24
8	47	25	20	23
16	27	17	10	8
32	14	12	7	4

In Table 7.9 we show the number of clock cycles where two consecutive doubling operations can be computed in parallel. The tables for addition after doubling and doubling after addition show similar figures and can be found in Table 7.11 and Table 7.10, respectively. One notices that overlapping decreases as the digit size and the number of multipliers increase. Larger digit sizes and numbers of multiplier units yield a higher parallelism at this level, which results in a smaller overlap for the group operations. In the best case we could schedule two consecutive group operations for 167 clock cycles in parallel (Table 7.9, first cell).

Table 7.10: Overlapping of addition after doubling (in clock cycles, projective coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	164	82	46	41
4	84	42	26	21
8	44	22	30	21
16	24	23	17	4
32	11	4	2	4

Table 7.11: Overlapping of doubling after addition (in clock cycles, projective coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	126	51	44	45
4	66	33	24	25
8	36	21	25	12
16	21	12	10	10
32	11	5	7	6

7.3.2 Performances of the Group Operations and Scalar Multiplication

In this subsection we present our results targeting the latency of group addition, group doubling and of the complete scalar multiplication. Table 7.12 provides the figures for group addition and doubling, and Table 7.13 those for the scalar multiplication. Note that we did not include a second performance number for doubling and addition, as we did for doubling in our analysis using affine coordinates (Section 7.2). The reason being that for projective coordinates the group operations are almost scheduled in the same way. Hence, it is not worth to chose the sequence of field operations dependent on the previous group operation.

The conclusion one can draw from the two tables is that with increasing hardware resources (more multiplier units and higher digit size), the latency drops. Thus we can compute the group operations and the scalar multiplication of HECC in a shorter time. Without constraints in terms of silicon area, group addition and group doubling can be executed most efficiently by using three or four multipliers ($D = 32$), respectively. These group operations can be computed in 99 and 81 clock cycles, respectively (Table 7.12, bottom two rows). Scalar multiplication can be performed most efficiently in $19,769$ clock cycles providing three field multipliers (of type $D = 32$), one field adder and one

Table 7.12: Latency of group addition and doubling (in clock cycles, projective coordinates).

Digit-size		Number of multipliers			
		1	2	3	4
2	addition	1876	1008	704	553
	doubling	1465	793	545	417
4	addition	976	528	384	293
	doubling	765	413	286	237
8	addition	526	290	220	170
	doubling	415	221	165	138
16	addition	296	166	127	115
	doubling	241	149	106	99
32	addition	170	113	101	99
	doubling	132	90	81	94

field squarer (Table 7.13, bottom row).

Table 7.13: Latency of the scalar multiplication (in clock cycles, group order $\approx 2^{160}$, projective coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	348210	188693	127690	96726
4	181670	97355	69730	55646
8	98400	53243	38935	31953
16	56525	32835	24747	22974
32	31702	21209	19769	20490

Our scheduler is based on the method known as Operation Scheduling and works according the ASAP scheduling policy thus does not guarantee an optimal solution [Gov03]; this is evident in the case of 4 multipliers of type $D = 32$ (Table 7.13, 5-th row). Whereas latency is higher than with only 3 multipliers. This happens as the scheduler could not find an efficient way of performing group doubling, as it can be seen in the bottom row of Table 7.12.

When carefully inspecting the results, we see that adding extra hardware resources might be unreasonable. For example, consider the group addition using multipliers of

type $D = 2$ (Table 7.12, first row). We get a 46 % speed-up when using 4 instead of 2 multipliers. Hence, in this case we double the number of multipliers and get a 50% improvement in the performances. Examine the same setting for multipliers of type $D = 32$ (Table 7.12, second to bottom row): we get only a 8% improvement when moving from 2 to 4 multipliers. In real applications where hardware resources are limited the latter scenario would be inefficient. Thus, we provide in the next section the area-time products to find the optimal solutions.

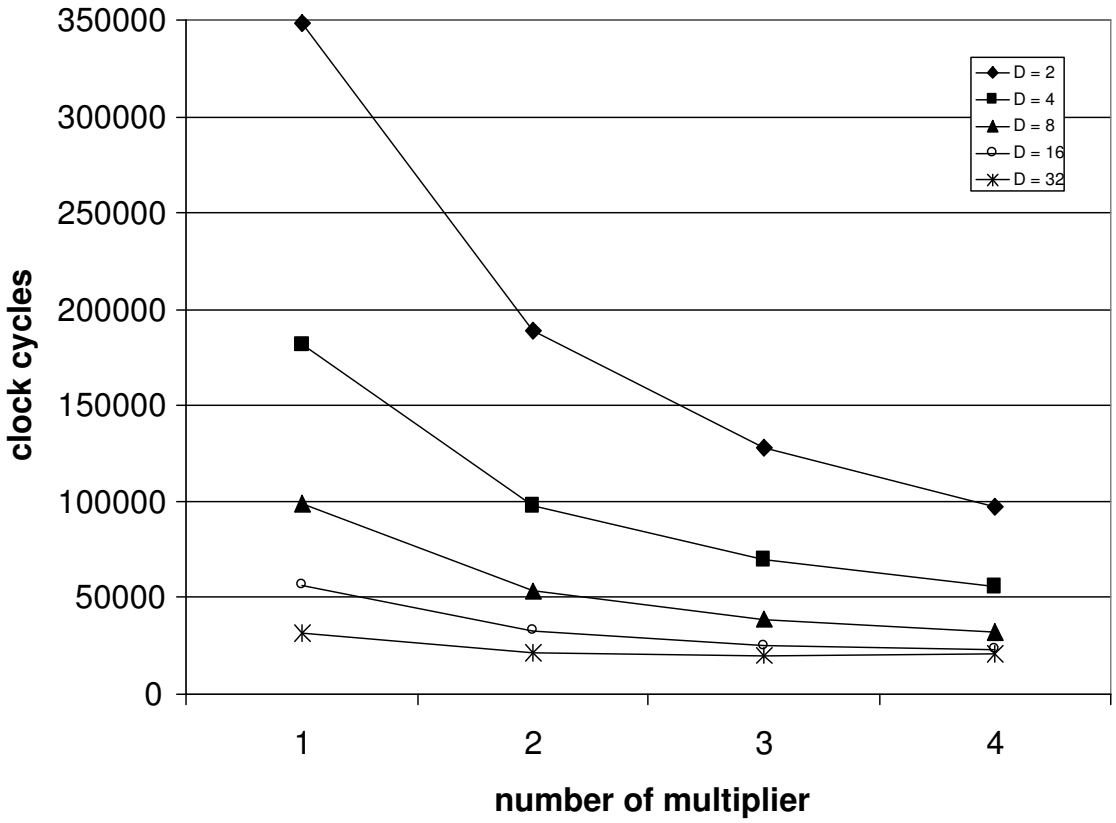


Figure 7.2: Latency of the scalar multiplication using projective coordinates (group order $\approx 2^{160}$).

In order to find an easy way of examining the performance improvement considering the different architectures, we added Figure 7.2. The figure shows the latency of the scalar multiplication (x-axis) with respect to different numbers of multipliers (y-axis).

Each line corresponds to a different digit size. Hence, if the line joining two values declines sharply, it can be interpreted as a considerable decrease of the computation time obtained by means of the additional resources. Going back to our example, the line for $D = 2$ falls more steeply than that for $D = 32$.

7.3.3 Area-Time Product

The optimal implementation will achieve the highest throughput consuming the smallest area. Hence, we consider both the hardware requirements and the time constraints of the cryptographic application.

Table 7.14: Normalized area-time product for scalar multiplication (in clock cycles, group order $\approx 2^{160}$, projective coordinates).

Digit-size	Number of multipliers			
	1	2	3	4
2	1.4533	1.1813	1.0659	1.0093
4	1.1374	1.0158	1.0186	1.0451
8	1.0267	1	1.0563	1.1336
16	1.0616	1.1649	1.2911	1.5821
32	1.1247	1.4606	2.0215	2.7794

In Table 7.14 we show the area-time product for the different design options. The analysis uses the normalized area-time product with respect to the lowest area-time product. Table 7.14 shows that the architecture using two multipliers (of type $D = 8$), one adder and one squarer achieves the best area-time product and therefore is the optimal architecture.

7.4 Optimum Architecture

The main contribution of this section is to identify which coordinate system, affine or projective, is best for the architecture for a parallel architecture. In order to do so we need to compare the different architectures for the affine and projective group operations.

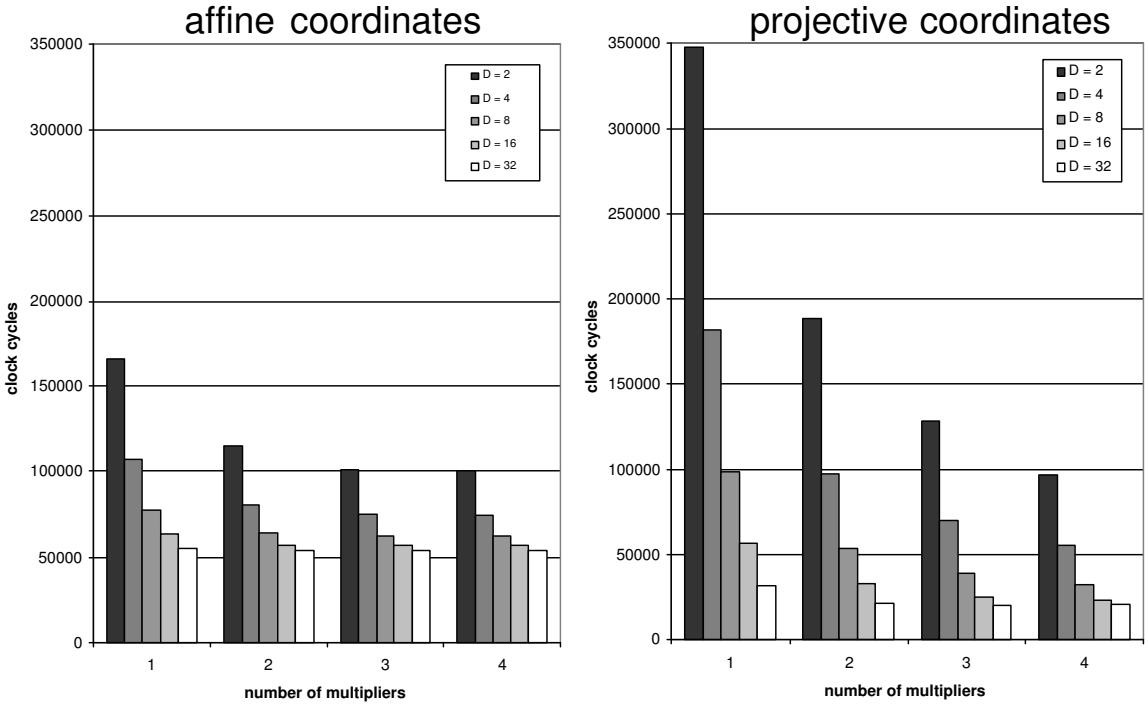


Figure 7.3: Latency comparison of the scalar multiplication between the architectures based on affine and projective coordinates (group order $\approx 2^{160}$).

Figure 7.3 compares the latency between the architectures based on affine and projective coordinates. The lowest bars corespond to lower latency, hence better performance. One can draw the following conclusions:

- In both coordinate systems, latency drops by providing additional hardware resources (increasing the digit size and the number of multiplier units)
- If the context allows to use projective coordinates, they are always preferable in terms of latency and area. However, affine coordinates might find an appropriate

use in low area applications.

- One very important result is that high speed implementations should definitely based on projective coordinates. As it can be seen in Figure 7.3, projective coordinates and large digit sizes result in the lowest latency.

In Figure 7.4 we compare the five best area-time product figures using the two different coordinate systems. For each system configuration the area-time product and the latency is reported and both measures are normalized to the minimum value (in our case the configuration with the best area-time product is even the fastest). The leftmost bars show the figures for affine coordinates and the right-most bars show those for the projective case. The design option used is given under each bar (M denotes the multipliers).

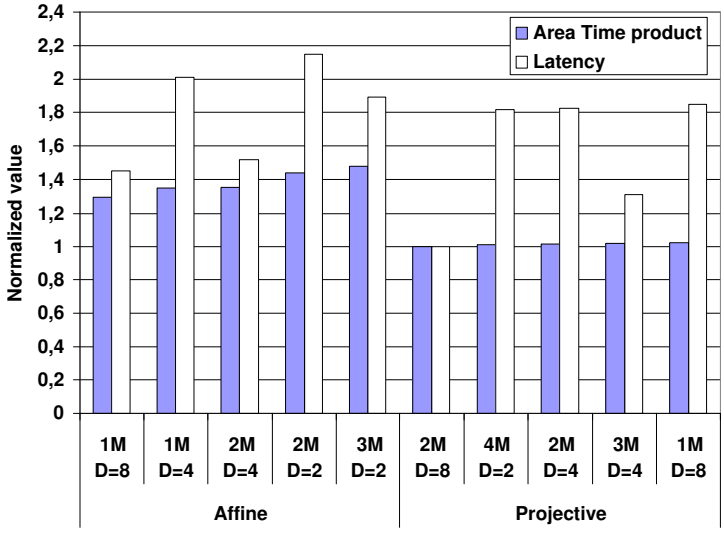


Figure 7.4: Comparison of the best five area-time product figures using affine and projective coordinates (group order $\approx 2^{160}$).

It is interesting to analyze the implicit parallelism using projective formulae. Analysis of the best five area-time products shows there is only one system configuration with one multiplier (in addition this is the worst architecture under the five best using projective

coordinates). In the case of affine coordinates we see that the two best solutions contain only one multiplier. Hence, potentially we cannot parallelize as many field operations in the affine case.

Examine the two best design options of using one multiplier of type $D = 8$ for affine coordinates versus two multipliers of type $D = 8$ for the projective case. In Figure 7.4 these cases are represented by the two leftmost bars for each coordinate system. We realize that there is a gap of about 30% in the area-time product between affine and projective coordinates. Comparing now for the same configurations the two bars representing the latency (second bar for each coordinate system), we realize that in the projective case we get a performance almost 50% better than in the affine case. This speed-up was obtained on the basis of an already 30% better area-time product. We can safely conclude that projective coordinates are preferable at parallel hardware is available.

All the results considering projective coordinate system presented so far target an application using only projective coordinates. Thus, one does not need to convert between coordinate systems. If a change of coordinates is strictly required one has to compute an inversion and four additional multiplication operations. In terms of latency one inversion costs ≈ 160 clock cycles, while the inclusion of an inversion function unit increases the area from 4% to 152%, depending on the number and type of multipliers in the architecture.

Figure 7.5 shows the area-time product of the two coordinate systems, whereas in the figures related to projective coordinates we included the conversion. Note that the top five design options using projective coordinates are not the same as the previous case, where the coordinate conversion was not included. The reason is the relatively high area increase for small system, due to the presence of inversion. However, the total computation time remains practically unchanged since the conversion has a very low

impact on the scalar multiplication time. Hence, fast and large implementations exhibit a relatively better area-time product.

Analyzing Figure 7.5 we see that the best system configurations using affine and projective coordinates have almost the same area-time product (1% difference for the best case). Comparing the two latency bars for the best solutions of the two coordinate systems we realize that the projective case is 50% faster than the affine case. This speed-up was obtained on the basis of the same area-time product. Hence, even when including the conversion between coordinate systems, the use of projective coordinates is still preferable.

This result of this theoretical comparison is quite surprising considering the fact that using projective instead of affine coordinates one has to trade one inversion operation with up to 24 multiplications. Hence, projective coordinates are not at all attractive for sequential processing like in most embedded and general purpose processors. However, our results show that the larger number of multiplications allows a higher degree of parallelization. Therefore, HECC realized in hardware can result in a better performance and in a lower area-time product.

7.5 Summary and Outlook

In this section we analyzed the different parallel architectures for genus-2 HECC. In the first part of the chapter we described the methodology used. After an extensive treatment of a variety of architectural options using affine and projective coordinates in the second and third part of the chapter, respectively, we compared the architecture options in last part of the chapter. Hence, we were able to find the best architecture realizing HECC.

The *main* finding of this theoretical analysis is that architecture based on *projective*

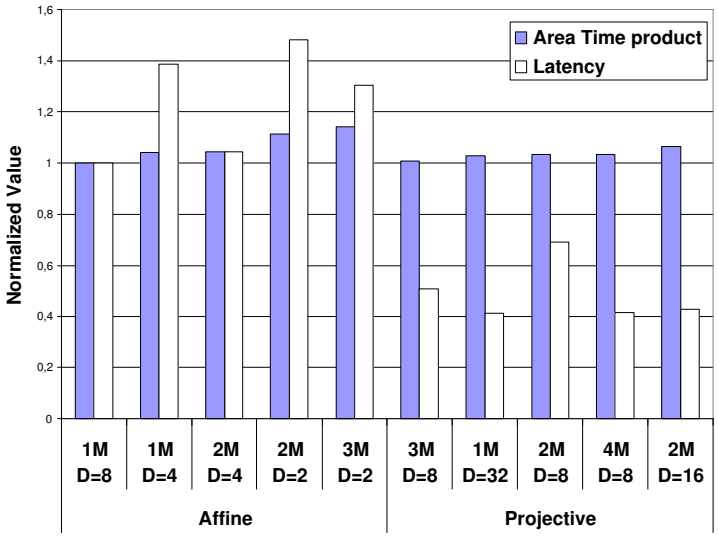


Figure 7.5: Comparison of the best five area-time products using affine and projective coordinates (projective numbers includes the conversion of coordinates, group order $\approx 2^{160}$).

coordinates are more flexible than those based on affine coordinates, allowing the designer to chose the best compromise in terms of required latency and silicon area. If the application imposes data conversion between the different coordinate systems, the architecture based on affine coordinates become attractive for their low area requirements.

We want to point out that the above presented results are based on our scheduling optimization software. We did not implement any of the architectures introduced in actual hardware. In general, hardware designers face the problem that VLSI designs and the manufacturing of prototypes are very complex and costly. Hence, the given results provide a theoretical basis, on which one can built when starting to design a HECC VLSI coprocessor. We would also want to mention that we fixed the underlying field and used certain curve parameters.

In the future, further analysis using different field sizes as well as different genera and curve parameters should be done. Furthermore, one should try to actually realize a HECC coprocessor prototype. This prototype can help to investigate further improve-

ments to the HECC coprocessor design.

8 A High Speed HECC Coprocessor on FPGAs

FPGA (Field Programmable Gate Arrays) are an interesting target platform for cryptographic algorithms. Not surprisingly, one can find many companies already developing cryptographic IP cores for FPGAs (for example, ALMA Technologies, Amphion, Bisquare Systems Private Ltd., Helion Technologies, Ocean Logic Pty Ltd). In this chapter, we present our implementation results of HECC on a FPGA. We propose a genus-2 hyperelliptic curve cryptographic coprocessor using affine coordinates. We provide three prototype implementations of the coprocessor aiming for high performance.

FPGAs have some potential advantages compared to ASIC implementations in cryptographic applications: algorithm agility, algorithm upload, architecture efficiency, resource efficiency, algorithm modification, throughput, and cost efficiency. More details and a description to each item can be found in [WGP04, WP04]. However, the listed advantages of FPGAs for cryptographic applications can only be exploited if the potential security shortcomings of FPGAs have been addressed. Potential shortcomings are Black Box Attack, Cloning of SRAM FPGAs, Readback Attack, Reverse-Engineering of the Bitstreams, Side Channel Attacks, and Physical Attacks [WGP04, WP04].

Black box attacks do not seem to be feasible for state-of-the-art FPGAs. However, it seems very likely for an attacker to get the secret information stored in a FPGA,

when combining readback and fault induction attacks. Cloning of SRAM FPGA and reverse engineering depend on the specifics of the system under attack, will probably involve a lot of effort, but are not entirely impossible. Physical attacks against FPGAs are very complex due to the physical properties of the semiconductors in the case of flash/SRAM/EEPROM FPGAs and the small size of AF cells. It appears that such attacks are even harder than the analogous ones against ASICs. Even though FPGA have different internal structures than ASICs with the same functionality, we believe that side-channel attacks against FPGAs, in particular power-analysis attacks, will be feasible too. More detailed information about security aspects of FPGAs and the prevention of them can be found in [WP03b, WGP04, WP04].

It seems from our previous remarks that FPGAs might be out of question for security applications. We do not think that is the right conclusion, however. It should be noted that many commercial ASICs with cryptographic functionality are also vulnerable to attacks similar to the ones discussed here. A common approach to prevent these attacks is to put the ASIC in a secure environment. A secure environment could, for instance, be a box with tamper sensors which triggers what is called "zeroization" of cryptographic keys, when an attack is being detected. Similar approaches are certainly possible for FPGAs too.

In the following we present our approach to implementing HECC on a FPGA[1]. We first present the HECC coprocessor (Section 8.1). Then, we are going to outline our methodology and the different design options (Section 8.2). At the end of this chapter we put our results in perspective to previous ECC and HECC FPGA implementations.

[1]This is joint work with Dr. HoWon Kim who did the VHDL implementation of the HECC coprocessor.

8.1 HECC Coprocessor on FPGA

The coprocessor has to compute the scalar multiplication kD for the HECC. Efficient algorithms are needed to implement the group operations as well as the field arithmetic. This work proposes three different processor architectures that are well suited for implementation on Field Programmable Gate Arrays (FPGAs).

The HEC coprocessor consists of three main components: main control unit (MC), arithmetic unit (AU), and register file or memory (RF), see Figure 8.1. MC is the main control unit of the coprocessor. It computes the scalar multiplication kD for HECC and interacts with the host system as well as with the other components. It generates control signals for RF, interconnection block, and AU. The interconnection block, which is mainly composed of multiplexers, takes care of the data exchange between the RF and the AU. The AU performs the field and group operations. In the following, we are going to describe the individual components in more detail.

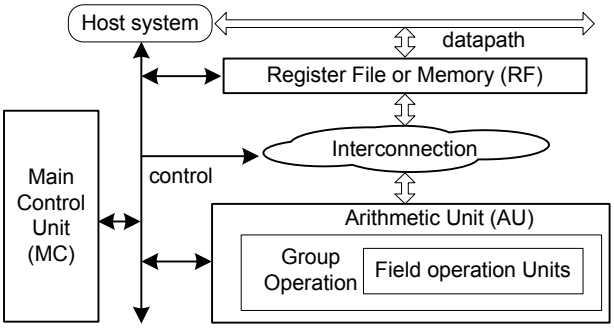


Figure 8.1: Architecture of the HECC coprocessor on FPGAs.

8.1.1 Field Operation Units

The AU in the HECC coprocessor has field operation units for addition, squaring, multiplication and inversion over $\mathbb{F}_{2^{81}}$. The implementations of the field $\mathbb{F}_{2^{81}}$ might have potential cryptographical weaknesses according to [Fre98, GHS02], however, we would

like to emphasize that the implementation techniques are also applicable to fields with prime extensions, see Page 4.

In existing literature one can find a variety of ways to implement the field operations. We provide a short description of the implementation and give references pointing to more detailed information.

FIELD ADDER: The addition of two elements requires the modulo 2 operation of the coefficients of the field elements. Hence, we use 81 exclusive OR gates to add two field elements in one clock cycle.

FIELD SQUARER: The squaring of a field element $A = \sum_{i=0}^{m-1} a_i x^i$ is ruled by the following equation: $A^2 \equiv \sum_{i=0}^{m-1} a_i x^{2i} \bmod F(x)$, where $F(x)$ denotes the defining polynomial of the field $\mathbb{F}$. The squarer is implemented with combinatorial logic and we use a fixed underlying polynomial $F(x)$. Squaring can be performed within one clock cycle and further details can be found in [OP00].

FIELD INVERSION: The inversion was implemented using the Modified Almost Inverse Algorithm (MAIA) [HHM00] rewritten in Algorithm 1. In addition, we increased the performance by using the techniques described in [CKK02, SK98]. These techniques allow executing of Steps 2 to 5 of Algorithm 1 in nearly one clock cycle. In order to do so, we a) unrolled the loop (Step 2) and b) merged Step 5 in Step 2. To realize the improvement a) we had to replicate and expand the conditional statement for all cases. The replication number of the loop body was decided by considering the tradeoff between the hardware complexity and performance.

The field inversion can be performed in an average time of $1.01\mu s$ (see Table 8.1). The input data that was used to measure the execution time had an hamming weight of $n/2$, where n is the bit size of the input data. The reason being the performance of the inversion varies with the input data.

Algorithm 1. Modified Almost Inverse Algorithm for inversion in $\mathbb{F}_{2^m}$

INPUT : $a \in \mathbb{F}_{2^m}, a \neq 0$.
OUTPUT : $a^{-1} \bmod F(x)$
1. $b \leftarrow 1$, $c \leftarrow 0$, $u \leftarrow a$, $v \leftarrow f$.
2. While x divides u do:
   2.1 $u \leftarrow u/x$
   2.2 If x divides b then $b \leftarrow b/x$; $else\, b \leftarrow (b+f)/x$.
3. If $u = 1$ then return(b).
4. If $\deg(u) < \deg(v)$ then: $u \leftrightarrow v, b \leftrightarrow c$.
5. $u \leftarrow u + v, b \leftarrow b + c$.
6. Goto step 2.

Table 8.1: Performance of field multiplication and field inversion logic (FPGA Xilinx Virtex II XC2V4000 ff1517-6, $\mathbb{F}_{2^{81}}$).

Type	DigitSize	slices	Frequency [MHz]	Clock cycles	Time [μs]
LSD	D=4	219	98.0	21	0.214
	D=8	342	108.7	11	0.101
	D=16	554	87.5	6	0.069
	D=27	882	71.0	3	0.042
	D=32	1,010	70.0	3	0.043
MSD	D=4	215	98.6	21	0.213
	D=8	339	109.9	11	0.100
	D=16	551	71.1	6	0.084
	D=27	868	70.1	3	0.043
	D=32	1,066	69.6	3	0.043
MAIA	-	663	87.8	89	1.014

Field Multiplication: For the field multiplication we use the digit multiplier introduced in [SP97]. This kind of multiplier allows a trade-off between speed, area and power consumption. This can be achieved by varying the number of multiplicand coefficients that are processed in parallel, denoted as digit-size D. Given D, we denote by $d = \lceil m/D \rceil$ the total number of digits in a polynomial of degree $m-1$. Hence, $C \equiv AB \equiv A \sum_{i=0}^{d-1} B_i x^{Di} \bmod F(x)$.

We investigated the performance of the field multiplication in more depth because it is a crucial field operation in the HECC. The digit multipliers come in two flavors: Least

Significant Digit (LSD) first and Most Significant Digit (MSD) first multipliers [SP97]. Additionally, one can vary the digit size depending on the application. Table 8.1 shows the latency and the area requirement of the two different types of multiplier as well as varying digit sizes.

One would expect the frequency of the multipliers to be decreased with higher digit size. However, examining Table 8.1 one notices the lower frequency for D=4 multiplier compared to D=8. Careful analysis shows that the critical path of the D=4 multipliers are dominated by the control logic, more specifically from the 5 bit comparator logic and additional logic (such as multiplexers). For all the other multipliers the critical path is conditioned by the data path. The data path includes the parallel multiplication, the accumulation of the partial results and the reduction (for more detail see [SP97]).

The choice of the ideal multiplier for our HECC coprocessor largely depends on frequency of the total design and secondly, on the latency of the multiplier. The frequency of the coprocessor is limited by the interconnection network and does not exceed 61 MHz (see Table 8.3). Hence, the multipliers using digit size D=27 are the best choice. The reason being it can be clocked with a high enough frequency and computes the multiplication in only three cycles, at the same time requiring lower area compared to the D=32 multiplier. Note that LSD and MSD multipliers using digit size 27 have very similar performance and, therefore, both types are applicable.

8.1.2 Arithmetic Unit

Figure 8.2 shows the arithmetic unit (AU) that performs the group addition and group doubling operations. The AU consists of the field operation units and the group operation logic. The control logic is in charge of scheduling the right sequence of operations to perform the HEC group operations. Necessarily, the control unit has to interact with the main control unit (MC) and the register files (RF). In addition, we had to provide

internal control logic for the multiplier and inversion logic.

Field addition and field squaring are the least expensive operations and, therefore, we decided to integrate them into the data path between RF and the multipliers. However, we still can compute the operation without consecutive multiplication by setting the right multiplexer options.

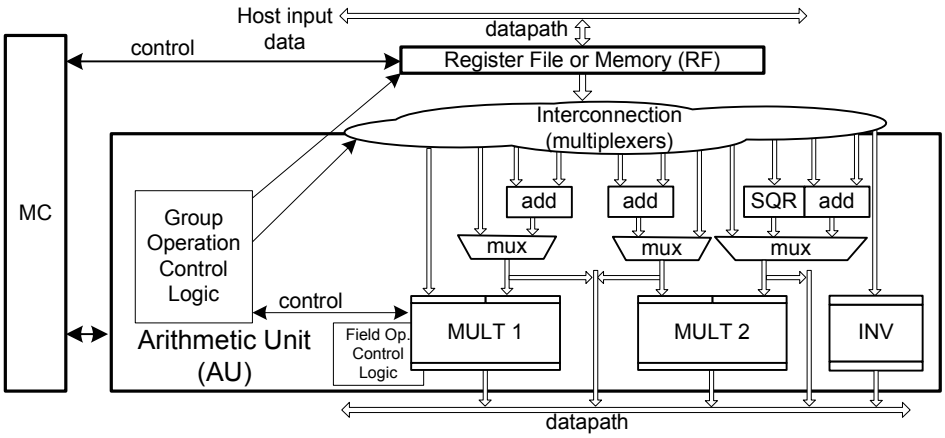


Figure 8.2: Arithmetic Unit of the HECC coprocessor on FPGA.

8.1.3 Interconnection Network

The interconnection network, consisting of a multiplexer, handles the data transfer between the RF and AU. Note that for our high performance design (see Section 8.2), we included the capability to load the input data in parallel. This results in the possibility to start two multipliers and the inversion logic at the same time.

8.2 Design Methodology for the HECC Coprocessor on FPGA

In this section, we will briefly describe our design methodology that resulted in the HECC coprocessor. The primary goal for all presented design options was to reach the best

possible performance. However, we tried (where possible) at the same time to reduce overall hardware complexity. In order to do so, we examined the parallelism within the group operations, minimized the number of registers, and reduced the complexity of the interconnection network.

8.2.1 Parallel Architecture for the Group Operations

We used the Data Dependence Graph (DDG) to design the architecture for computing the group operations. We found that the multiplication and the inversion operations are the dominant components and, therefore, crucial for the overall performance. Hence, we designed our graph where the nodes of the DDG represent the multiplications and the inversions and the addition and squaring are an edge of the DDG.

We can describe the DDG as $G(F) = (V, E)$, where V is the set of multiplication and inversion operations, F is the HECC explicit formulae, and E is the data flow between multiplication, inversion, and registers.

Our analysis using the DDG resulted in the choice of using two parallel multipliers for the genus-2 HECC coprocessor. This choice considered the hardware complexity as well as the performance. The utilization rate of the two multipliers for group addition and group doubling is 91% and 50%, respectively. Therefore, adding additional multipliers would not provide any advantages considering the increasing hardware complexity. Hence, we used two multipliers and sped up the performance of the HECC coprocessor by choosing a large digit size (D=27).

As mentioned already, there has been a previous contribution studying the parallelism of the genus-2 HECC group operations [MS03]. In [MS03], the authors develop a general methodology for obtaining parallel algorithm for the HECC. However, this work only focused on theoretical aspects on the parallelism of the HECC. Furthermore, they did

not consider the register allocation and interconnection network complexity problems, which are important factors in practical implementations.

8.2.2 Minimizing the Number of Registers

We have designed the HECC coprocessor with an effort to minimize the number of registers. We used eight 81-bit registers to store two divisor values which are updated during the computation of the group operations with the output result. Additionally, we needed some extra registers to store intermediate values. We found the optimal number of extra registers by efficiently reusing these resources. This was done manually with the help of register allocation tables.

8.2.3 Reduction of the Complexity of the Interconnection Network

After we found the minimum number of registers, we tried to reduce the complexity of the interconnection network. We did this by looking at the inputs and outputs of the different units (field operations and RF) involved in the computation. The minimum complexity would be achieved by a) always storing a designated output into the same register and b) loading a certain input value every time from the same register. Looking at the complexity of the HECC group operations, it is obvious that there will not be a fixed register for loading inputs or storing the output of one unit. However, by trying to use this method we were able to reduce the size of the interconnection multiplexers.

8.2.4 Various Design Options for the HECC Coprocessor

We now introduce the three prototype implementations for HECC coprocessors. Our Type 1 coprocessor is designed for high performance, whereas in the case of the Types 2

and 3 we also tried to lower the hardware complexity. The characteristics of the various types of HECC coprocessors are summarized in Table 8.2.

At this point we would like to stress, that the two main factors limiting the performance of the HEC coprocessor are the AU and the interconnection network (see discussion in previous sections).

Type 1 Design: High Performance

Our Type 1 design aims for a high performance implementation and is shown in Figure 8.3(a). This design has two independent arithmetic units: one for group addition and one for group doubling. The independency of the two group operations results from separate field operation units, registers, interconnection networks, and control units. Thus, we can compute the group operations in parallel by using a modified version of the double-and-add scalar multiplication. In the case of group addition we used two multipliers and one inversion logic, whereas for group doubling we provided only one multiplier and one inversion logic.

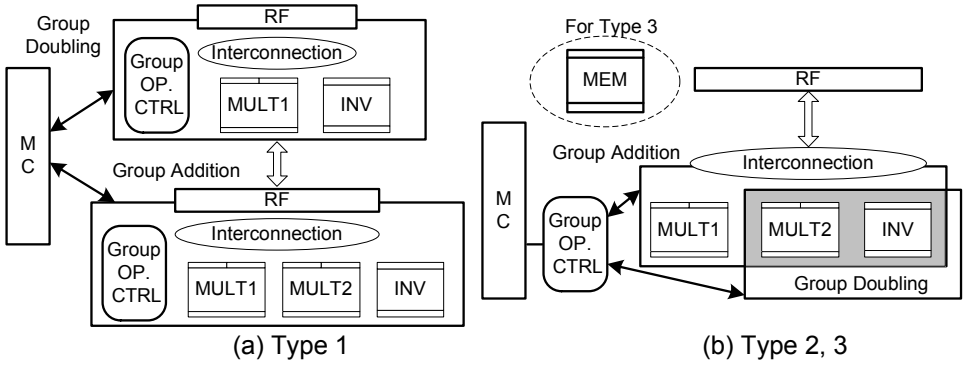


Figure 8.3: Various design options for the HECC coprocessor.

Type 2 Design: Resource Sharing

Our Type 2 HECC coprocessor provides only one AU shared by the group addition and group doubling (see Figure 8.3(b)). Hence, the two group operations share the field arithmetic unit, the register file, the interconnection network, and the control logic.

Another difference is that a more complex interconnection network is needed to handle the computation. These differences result in a slower operating frequency, however, the total hardware complexity is smaller compared to Type 1.

Type 3 Design: Reduced Area

Type 3 HECC coprocessor also uses, like in the case of the Type 2 design, shared resources (see Figure 8.3(b)). The difference between Type 2 and Type 3 design is the usage of memory for storing the intermediate data needed during the computation of the group operations and the scalar multiplication. When we use memory instead of registers, the decoding logic for reading (writing) data from (to) the RF unit is intrinsically implemented inside the memory.

We have used distributed memory with dual ports (DIST_MEM_V6), which is internally provided by the Xilinx FPGAs. We have chosen the smallest memory block available, namely of the size 1,536 bits, however we are currently using only 1,134 bits to store 14 field elements.

Trading memory versus registers, results in higher frequency and smaller area, however, there are two disadvantages: a) we use specific memory which will be costly when converting the system to an ASIC and b) the number of clock cycles for the overall computation increases, because of the expensive data movement from and to the memory. The latter disadvantage was partially removed by applying pipelining to the HECC coprocessor. For example, one can move data to or from the memory while simultaneously performing a field multiplication.

In Chapter 7, we provided a theoretical analysis of the parallelism in HECC in order to find the optimal architecture. The implementation results presented in this chapter primarily aim high performance. This was achieved by using a more efficient inversion and multiplication. Additionally, we adjusted the architecture to obtain a higher performance by including the addition/squaring into the data path and by using input buses to

Table 8.2: Architectural characteristics of the different HECC coprocessor types.

	logic		interconnection	scalar mult.	Storage for RF
Type 1	addition:	2 MUL, 1 INV	Multiplexers	Right to Left (parallel)	13 registers
	doubling:	1 MUL, 1 INV			10 registers
Type 2	2 MUL, 1 INV (shared)		Multiplexers	Left to Right	14 Registers
Type 3	2 MUL, 1 INV (shared)		Multiplexers	Left to Right	Memory (1,536 bits)

transfer the field elements to the arithmetic units. We also targeted the implementation so that we could start more than one field operation in each clock cycle. The previous chapter considered architectures with different numbers of multipliers. However, a general result was using two multipliers is a good choice for high performance and moderate area increase. The FPGA implementations described in this chapter use two or three multipliers and therefore are in the same line with Chapter 7.

8.3 Results of HECC Coprocessor on FPGA

In this section we present our implementation results using the methodology stated above. We show the throughput results of different designs.

We implemented the HECC coprocessor using the best known formulae for the group operations (for more details see Section 4.4). It was modeled using VHDL language and then implemented a Xilinx Virtex II Pro V20 FPGA (XC2V P20ff1152-7). The VHDL code was synthesized using Synplicity's Synplify Pro 7.3.1 and Xilinx Foundation 5.2.03i to implement the modeled HECC coprocessor onto the target FPGA. Our HECC coprocessor used between 43% and 83% of the slices available in the FPGA.

Table 8.3 shows the area and time requirements for the three different design options. Type 1 corresponds to our high performance implementation. We were able to compute

the scalar multiplication in $387\mu s$, which is about 81% faster than the best known implementation presented in [EMY04]. The area requirements decrease almost 50% changing from Type 1 to Type 3. Hence, our Type 3 design utilizes 81% less area than the smallest design described in [EMY04].

Table 8.3: Performance of the scalar multiplication on the HECC coprocessors (Xilinx FPGA XC2VP20 ff1152-7, group order 2^{162}).

	Size [*slices*]	Frequency [MHz]	Clock cycles	Time [μs]
Type 1	7,737	60.7	23,509	387
Type 2	5,674	51.4	34,012	662
Type 3	4,039	57.0	44,848	787

8.4 Summary and Analysis of our FPGA Implementation

In this chapter we presented our approach to implement HECC on FPGA. The HECC coprocessor, our design methodology, and the different design options were introduced. The main focus of this section is to put our results in perspective to previous published ECC and HECC implementations.

We evaluated the performance of cryptographic implementations by comparing not *only* the throughput of the implementation but also using the area-time product. Therefore, the optimal implementation will achieve the highest throughput in the least amount of area and, thus, the *lowest* area-time product. At this point, we must caution the reader against using the area-time product to compare implementations on *different* FPGA devices. This is because even within the same family, one may get different timing results based on available logic and routing resources.

In order to be able to provide a fair comparison between our work and the ones previously presented, we also did the place and routing for the HECC coprocessor with

Table 8.4: Comparison of the ECC and HECC scalar multiplication implementations on FPGAs (target device is Xilinx FPGA XC2V4000, except in [OP00]).

	scalar mult.	group order	digit size	slices slices	f [MHz]	Time [μs]	area-time product
genus-2 HECC							
Clancy [Cla03]	affine coord. binary	2^{166}	D=1	22,000	-	10,000	68.10
		2^{166}	D=4	60,000	-	9,000	167.14
Elias et al. [EMY04]	proj. coord. NAF	2^{226}	D=1	21,550	45.6	7,390	49.29
		2^{226}	D=4	25,271	45.3	2,030	15.88
our work	affine coord.						
Type 1	binary	2^{162}	D=27	7,785	56.7	415	1
Type 2		2^{162}	D=27	5,604	47.0	724	1.26
Type 3		2^{162}	D=27	3,955	54.0	831	1.02
ECC							
Orlando et al. [OP00]	proj. coord. Montgomery	2^{167}	D=16	1,501	76.7	210	-
Gura et al. [EGCS03]	proj. coord. Montgomery	2^{163}	D=64	11,845	66.4	143	0.52

a target FPGA, Xilinx Virtex II FPGA (XC2V4000ff1517-6). In the case of XC2V4000, our designs used between 17% and 34% of the available slices. Table 8.4 shows all HECC hardware numbers and the two best known FPGA implementation of ECC.

Note that we did not calculate the area-time product for one of the ECC implementations, because in [OP00], the authors used a different type of FPGA, namely the Xilinx FPGA XCV400E. However, for completeness and in order to put our work in perspective with state-of-the-art research on ECC we think it is useful to include these numbers.

Analyzing the area-time product numbers of our designs, we noticed that they are fairly similar. Thus, considering different application scenarios, e.g., high speed or low area, we are able to provide an adequate solution. Comparing our results to the previously published shows that our designs are an order of magnitude better. Our implementations perform between a factor of 16 and 167 better than previous implementations. Note that the authors in [EMY04] used a different underlying field and, therefore, the

factor will be smaller when changing the field. Considering our HECC coprocessors we are now approaching the performance range of ECC FPGA implementations. Considering the area-time product, ECC is only about a factor of two better than HECC.

At this point we would like to clarify that this is only a prototype implementation. Higher speed and lower area could most likely be achieved when further optimizing the VHDL code as well as the design methodology. In addition, one will obtain better results with new FPGA releases and tool versions. Furthermore, when designing a HECC ASIC coprocessor the results are expected to be better. However, the presented results give a trend of the performance and area usage of implementing genus-2 HECC on a state-of-the-art FPGA family.

We want to point out that the presented results using a fixed underlying field and the HECC coprocessor is targeted for the genus-2 HEC group operation using certain curve parameters.

9 Discussion

This chapter summarizes research finding of the thesis. A summary of the main results as well as some recommendations for future research will be provided.

9.1 Conclusions

Today most cryptographic protocols use RSA [RSA78] for public key encryption. However, ECC increasingly used in practice as well. The reason being the short operant length of ECC resulting in a much higher speed for the encryption compared to RSA. ECC is the fastest used public-key primitive known today. HECC was thought to be out of scope for any practical use because of the complex structure of the group operations and the resulting low performance.

We were able to increase the performance of HECC. The newly derived group operations are up to 78% more efficient than the previous known operations. Furthermore, we showed the influence of the algorithm, group order, and curve parameters on the performance for a given application. We demonstrated with our implementations in software and hardware that HECC reaches the speed of ECC. In some cases we could even outperform ECC.

Based on the results presented in this contribution, explicit formulae based on Harley's algorithm should be used when implementing genus-2 and genus-4 curves. In the case

of genus-3 curves the doubling operation should be implemented using explicit formulae based on Cantor and for the addition we advise the use of Harley's algorithm.

Our software implementations show that ECC, genus-2, and genus-3 HECC have a similar performance. This was verified with various implementations on embedded and general processors using different underlying fields. Hence, the stated result holds for applications with different security levels. The best timings for the scalar multiplication considering a group order of approximately 2^{163} for HEC cryptosystems could be achieved on the ARM7TDMI@80MHz, resulting in 87 and 94 milliseconds for genus 2 and 3, respectively. The fastest scalar multiplication for ECC on the same platform was performed in 108 ms. For the same security level, the scalar multiplication on an Pentium@1.8GHz could be performed in 2.60 ms, 2.73 ms, and 2.95 ms for ECC, genus-2 HECC, and genus-3 HECC, respectively. Hence, ECC and HECC can be the cryptosystem of choice for general purpose processors as well as for embedded processors.

We did a theoretical investigation of various parallel architectures for a genus-2 HECC using affine and projective coordinate systems. We studied the parallelism of the HECC at the field operation level, the group operation level, and the scalar multiplication level. The main finding of this theoretical analysis is that the architecture based on projective coordinates are preferable compared to affine ones. Our fastest scalar multiplication was performed in $19,769$ clock cycles. It uses three field multipliers (of type $D = 32$), one field adder and one field squarer.

Analyzing the encryption speed of HECC implemented on FPGA leads to the same conclusions as stated before, namely that HECC reaches the performance ECC. Our fast version of HECC implemented on FPGA can perform one scalar multiplication in $415\mu s$ and is therefore 81% faster than the best previous implementation. Thus, one can use HECC coprocessor as an alternative encryption scheme for cryptographic accelerators.

In summary, we presented a variety of engineering aspects of HECC improving their

acceptance. This was achieved by increasing the performance of HECC group operation and by presenting similar speed for ECC and HECC considering software and hardware implementations.

9.2 Further Research

This thesis concentrated on the improvement of the HECC as well as on the implementation in software and hardware. This section will provide the reader with an overview of possible areas in which further work could be pursued. The presented ideas came up during the research and implementation that was done. These recommendations provide opportunities to investigate further the engineering aspects of HECC.

Further optimization of the explicit group operation: In Sections 4.1 and 4.2 we introduced all our techniques used to improve the explicit formulae for the HECC group operation. The resulting efficient group operations are introduced in Section 4.3. The group operation for higher genus HECC are very complex and therefore probably one can improve them further, by a) repeating the proposed techniques or by b) finding new techniques for improvement.

Deriving projective coordinates for genus-3 and genus-4 HECC: In [MDM+02, Lan02b, Lan03], the authors introduced the HEC group operation using projective coordinates for genus-2 curves. Hence, one can encrypt without having to provide inversion operations in the underlying field. In our contribution we improved the genus-2 HECC using projective coordinates, as shown in Table 4.4. Taking this contribution for genus-2 curves as a starting point, one can formulate projective group operations for higher genus HECC. In addition this formulae could be improved, by the techniques presented in Sections 4.1 and 4.2.

FPGA implementation using projective coordinates: In [EMY04], the authors

did a HECC implementation on FPGA using projective coordinates. However, they did not use our optimized formulae. In addition, one could inspect the features of the FPGA to target the implementation specifically for this platform. One should also consider various design options for this implementation targeting different cryptographic applications, similarly to the work presented in Section 8.

Software and hardware implementation of HECC using underlying prime fields: Many high security applications, e.g. for government use, require the use of prime fields when implementing curve based cryptographic algorithms. Hence, it is very important to get reliable performance numbers for HECC using prime fields. The software as well as the hardware implementation presented in this contribution is based on fields of characteristic two. In [Ava04], the author implemented HECC using prime fields on a AMD Athlon running at 1GHz. These results, even though limited to software implementation on a general purpose processor, parallel the results in this theses nicely. In addition, implementations on embedded processors, as well as on more hardware-oriented platforms such as FPGAs could be investigated.

Assembly implementation of HECC on embedded processors: The efficiency of compilers has improved a lot over the last years, and it can be expected that it will further advance in the future. However, the tools will never be perfect and one can always get a significant speed-up by rewriting the core routines in assembly. This is especially important for embedded application, because the performance is already limited by the processor.

Implementation of HECC on 8bit and 16bit processors: Our contribution clearly shows that ECC and HECC are the cryptosystems for embedded processors at least as far as binary fields are used. In Section 6.3 we extensively studied the performance of a variety of important 32-bit processors. We think that HECC can also be implemented with acceptable performance on 8bit and 16bit embedded processors. These kinds of

processors play a important role in many cryptographic applications.

Implementation of HECC using Koblitz Curves: In [Kob91], the author presented Koblitz curves to perform a scalar multiplication in the case of ECC more efficiently. We implemented the Frobenius map using Koblitz curves targeting a group order of approximately 2^{160} for ECC and obtained very competitive timings. Hence, it seems very interesting to investigate the performance of Koblitz curves for HECC.

A Explicit Formulae for the Group Operations over GF(p)

Table A.1: Explicit formulae for adding on a HEC of genus two (Harley) [Lan02a].

Input	Weight two reduced divisors $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$ with $u_1 = x^2 + u_{11}x + u_{10}$; $u_2 = x^2 + u_{21}x + u_{20}$; $v_1 = v_{11}x + v_{10}$; $v_2 = v_{21}x + v_{20}$; furthermore: $h = h_2x^2 + h_1x + h_0$; where $h_i \in \{0, 1\}$; $f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$; where $f_4 \in \{0, 1\}$;	
Output	A weight two reduced divisor $D' = (u', v') = D_1 + D_2$ with $u' = x^2 + u'_1x + u'_0$; $v' = v'_1x + v'_0$;	
Step	Procedure	Cost
1	Compute resultant r of u_1 and u_2: $z_1 = u_{11} - u_{21}$, $z_2 = u_{20} - u_{10}$, $z_3 = u_{11}z_1 + z_2$ $r = z_2z_3 + {z_1}^2u_{10}$	$3M + 1S$
2	Compute almost inverse $inv = r/u_2 \bmod u_1$: $inv_1 = z_1$, $inv_0 = z_3$	$-$
3	Compute $s' = rs \equiv (v_1 - v_2)inv \bmod u_1$: $w_1 = v_{10} - v_{20}$, $w_2 = v_{11} - v_{21}$, $w_3 = inv_0w_1$, $w_4 = inv_1w_2$ $s'_1 = (inv_0 + inv_1)(w_1 + w_2) - w_3 - w_4(1 + u_{11})$, $s'_0 = w_3 - u_{10}w_4$ If $s_1 = 0$ perform Cantor	$5M$
4	Compute $s'' = x + s_0/s_1 = x + s'_0/s'_1$ and s_1: $w_1 = (rs'_1)^{-1}$, $w_2 = rw_1(= 1/s'_1)$, $w_3 = {s'_1}^2w_1(= s_1)$ $w_4 = rw_2(= 1/s_1)$, $w_5 = w_4^2$ $s''_0 = s'_0w_2$	$I + 5M + 2S$
5	Compute $l' = s''u_2 = x^3 + l'_2x^2 + l'_1x + l'_0$: $l'_2 = u_{21} + s''_0$, $l'_1 = u_{20} + u_{21}s''_0$, $l'_0 = u_{20}s''_0$	$2M$

Table A.1: (continued)

Step	Procedure	Cost
6	Compute $u' = (s(l+h+2v_1)-k)u_1^{-1} = x^2 + u'_1 x + u'_0$:	$3M$
	$u'_1 = h_2 w_4 + s''_0 + l'_2 - u_{11} - w_5$	
	$u'_0 = (s''_0 - u_{11})(l'_2 + h_2 w_4 - u_{11}) - u_{10} + l'_1 + (h_1 + 2v_{21})w_4 +$	
	$(u_{11} + u_{21} - f_4)w_5$	
7	Compute $v' \equiv -(h + l + v_2) \bmod u'$:	$4M$
	$w_1 = l'_2 - u'_1$, $w_2 = u'_1 w_1 + u'_0 - l'_1$	
	$v'_1 = w_3 w_2 - v_{21} - h_1 + h_2 u'_1$	
	$w_4 = u'_0 w_1 - l'_0$	
	$v'_0 = w_3 w_4 - v_{20} - h_0 + h_2 u'_0$	
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_4 = 0$	$I + 22M + 3S$
	fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_4 = 0$	$I + 21M + 3S$

Table A.2: Explicit formulae for doubling on a HEC of genus two (Harley) [Lan02a].

Input	Weight two reduced divisors $D = (u, v)$ with	
	$u = x^2 + u_1 x + u_0$;	
	$v = v_1 x + v_0$;	
	furthermore:	
	$h = h_2 x^2 + h_1 x + h_0$; where $h_i \in \{0, 1\}$;	
	$f = x^5 + f_4 x^4 + f_3 x^3 + f_2 x^2 + f_1 x + f_0$; where $f_4 \in \{0, 1\}$;	
Output	A weight two reduced divisor $D' = (u', v') = [2]D$ with	
	$u' = x^2 + u'_1 x + u'_0$;	
	$v' = v'_1 x + v'_0$;	
Step	Procedure	Cost
1	Compute resultant r of u and $h + 2v$:	$3M + 2S$
	let $\tilde{v} \equiv h + 2v \bmod u$:	
	$\tilde{v}_1 = h_1 + 2v_1 - h_2 u_1$; $\tilde{v}_0 = h_0 + 2v_0 - h_2 u_0$;	
	$w_0 = v_1^2$; $w_1 = u_1^2$; $w_2 = \tilde{v}_1^2$; $w_3 = u_1 \tilde{v}_1$;	
	$r = u_0 w_2 + \tilde{v}_0(\tilde{v}_0 - w_3)$;	
2	Compute almost inverse $inv \equiv r/\tilde{v} \bmod u_1$:	$-$
	$inv_1 = -\tilde{v}_1$; $inv_0 = \tilde{v}_0 - w_3$;	
3	Compute $k \equiv [(f - hv - v^2)/u] \bmod u$:	$1M$
	$w_3 = f_3 + w_1$; $w_4 = 2u_0$;	
	$k_1 = 2(w_1 - f_4 u_1) + w_3 - w_4 - v_1 h_2$;	
	$k_0 = u_1(2w_4 - w_3 + f_4 u_1 + v_1 h_2) + f_2 - w_0 - 2f_4 u_0 - v_1 h_1 - v_0 h_2$;	

Table A.2: (continued)

Step	Procedure	Cost
4	Compute $s' = kinv \bmod u$: $w_0 = k_0 inv_0$; $w_1 = k_1 inv_1$; $s'_1 = (inv_0 + inv_1)(k_0 + k_1) - w_0 - w_1 - u_1 w_1$; $s'_0 = w_0 - u_0 w_1$; If $s'_1 = 0$ perform Cantor's Algorithm	$5M$
5	Compute $s = x + s'_0/s'_1$ and s_1: $w_1 = (r s'_1)^{-1}$; $w_2 = r w_1 (= 1/s'_1)$; $w_3 = {s'_1}^2 w_1 (= s_1)$; $w_4 = r w_2 (= 1/s_1)$; $w_5 = w_4^2$; $s_0 = s'_0 w_2$;	$I + 5M + 2S$
6	Compute $l = su$: $l_2 = u_1 + s_0$; $l_1 = u_1 s_0 + u_0$; $l_0 = u_0 s_0$	$2M$
7	Compute $u' = [l^2 + w_4 l(2v + h) - w_5(f - vh - v^2)]/u^2$: $u'_1 = 2s_0 + w_4 h_2 - w_5$; $u'_0 = s_0^2 + w_4(h_2(s_0 - u_1) + 2v_1 + h_1) + w_5(2u_1 - f_4)$;	$2M + 1S$
8	Compute $v' \equiv -(h + w_3 l + v) \bmod u'$: $w_1 = l_2 - u'_1$; $w_2 = u'_1 w_1 + u'_0 - l_1$; $v'_1 = w_2 w_3 - v_1 - h_1 + h_2 u'_1$; $w_4 = u'_0 w_1 - l_0$; $v'_0 = w_3 w_4 - v_0 - h_0 + h_2 u'_0$;	$4M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_4 = 0$ fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_4 = 0$ fields of characteristic two, $h_i \in \mathbb{F}_2, h_2 = 0$, $f_4 = 0$ fields of characteristic two, $h(x) = x$, $f_4 = f_3 = f_2 = 0$, see Table A.3	$I + 22M + 5S$ $I + 20M + 5S$ $I + 17M + 5S$ $I + 9M + 6S$

Table A.3: Optimized explicit formulae for doubling a divisor on special curves of genus two over $\mathbb{F}_{2^n}$ with $h(x) = x$.

Input	Weight two reduced divisors $D = div(u, v)$ with $u = x^2 + u_1 x + u_0$ $v = v_1 x + v_0$ furthermore: $h = x$ and $f = x^5 + f_1 x + f_0$	
Output	A weight two reduced divisor $D' = div(u', v') = [2]D$ with $u' = x^2 + u'_1 x + u'_0$; $v' = v'_1 x + v'_0$;	
Step	Procedure	Cost
1	Compute resultant r of u and $h + 2v$: $r = u_0$;	$-$

Table A.3: (continued)

Step	Procedure	Cost
2	Compute almost inverse $inv \equiv r/\tilde{v} \bmod u_1$: $inv_1 = 1$; $inv_0 = u_1$;	$-$
3	Compute $k \equiv [(f - hv - v^2)/u] \bmod u$: $w_0 = v_1^2$; $w_1 = u_1^2$; $k_1 = w_1$; $t_1 = u_1 k_1$; $k_0 = t_1 + w_0 + v_1$;	$1M + 2S$
4	Compute $s' = kinv \bmod u$: $t_2 = u_0 k_0$; $s'_1 = k_0$; $s'_0 = (u_0 + u_1)(k_0 + k_1) + t_1 + t_2$; If $s'_1 = 0$ perform Cantor's Algorithm	$2M$
5	Compute s_1 and $s_0 u_1$: $t_3 = t_2^{-1} (= 1/(rs'_1))$; $w_3 = r^2 t_3 (= 1/s_1)$; $w_4 = w_3^2$; $s_1 = s_1'^2 t_3$; $t_6 = t_1 + k_1 s_1 (= s_0 u_1)$;	$I + 3M + 3S$
6	Compute $z = su$ (Karatsuba): $z_0 = s'_0$; $z_1 = t_6 + s'_1$; $z_2 = w_1$; $z_3 = s_1$;	$-$
7	Compute $u' = 1/s_1^2((su + h + v)^2 + f)/u^2$: $u'_2 = 1$; $u'_1 = w_4$; $u'_0 = w_4 k_1^2 + k_1 + w_3$;	$M + S$
8	Compute $v' \equiv h + z + v \bmod u'$ (Karatsuba): $t_4 = w_3$; $t_7 = t_4 + z_2$; $t_5 = t_7 u'_0$; $v'_1 = (z_3 + t_7)(u'_0 + u'_1) + t_4 + t_5 + 1 + z_1 + v_1$; $v'_0 = t_5 + z_0 + v_0$;	$2M$
Total		$I + 9M + 6S$

Table A.4: Explicit formulae for adding on a HEC of genus two (Cantor).

Input	Weight two reduced divisors $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$ with $u_1 = x^2 + ax + b$; $u_2 = x^2 + cx + d$; $v_1 = ix + j$; $v_2 = kx + l$; furthermore: $h = h_2 x^2 + h_1 x + h_0$; where $h_i \in \{0, 1\}$; $f = x^5 + f_4 x^4 + f_3 x^3 + f_2 x^2 + f_1 x + f_0$; where $f_4 \in \{0, 1\}$;	
Output	A weight two reduced divisor $D' = (u', v') = D_1 + D_2$ with $u' = x^2 + a_3 x + b_3$; $v' = i_3 x + j_3$;	
Step	Procedure	Cost
1	Compute $gcd(u_1, u_2) = 1 = s_1 u_1 + s_2 u_2$: $r_{21} = a - c$; $r_{20} = b - d$; $t_{20} = -1$; $r_{31} = r_{21} \cdot c - r_{20}$; $r_{30} = r_{21} \cdot d$; $t_{30} = r_{21}$; $t_{31} = 1$;	$I + 8M + 2S$

Table A.4: (continued)

Step	Procedure	Cost
	$r_{40} = r_{31} \cdot r_{20} - r_{21} \cdot r_{30}$; $s_{20} = -r_{31} - r_{21}^2$; $s_{21} = -r_{21}$; $t_0 = r_{40}^{-1}$; $s_{20} = s_{20} \cdot t_0$; $s_{21} = s_{21} \cdot t_0$; $s_{11} = -s_{21}$; $s_{10} = -s_{21} \cdot c - s_{20} - a \cdot s_{11}$;	
2	Compute $u' = u_1 u_2 = x^4 + u'_3 x^3 + u'_2 x^2 + u'_1 x + u'_0$: $t_0 = b \cdot d$; $t_1 = a \cdot c$; $t_2 = (a + b) \cdot (c + d)$; $u'_0 = t_0$; $u'_1 = t_2 - t_0 - t_1$; $u'_2 = t_1 + b + d$; $u'_3 = a + c$;	$2M$
3	Compute $v' = s_1 u_1 v_2 + s_2 u_2 v_1 \bmod u'$: $t_0 = b \cdot s_{10}$; $t_1 = a \cdot s_{11}$; $t_2 = (a + b) \cdot (s_{11} + s_{10})$; $su_{10} = t_0$; $su_{11} = t_2 - t_0 - t_1$; $su_{12} = t_1 + s_{10}$; $su_{13} = s_{11}$; $t_0 = l \cdot su_{10}$; $t_1 = k \cdot su_{11}$; $t_2 = (k + l) \cdot (su_{10} + su_{11})$; $d_{00} = t_0$; $d_{01} = t_2 - t_0 - t_1$; $d_{02} = t_1$; $t_0 = l \cdot (su_{10} + su_{12})$; $t_1 = k \cdot (su_{11} + su_{13})$; $t_2 = (k+l) \cdot (su_{10} + su_{11} + su_{12} + su_{13})$; $d_{20} = t_0$; $d_{21} = t_2 - t_0 - t_1$; $d_{22} = t_1$; $suv_{10} = d_{00}$; $suv_{11} = d_{01}$; $suv_{12} = d_{02} + d_{20} - d_{00}$; $suv_{13} = d_{21} - d_{01}$; $suv_{14} = d_{22} - d_{02}$; $t_0 = d \cdot s_{20}$; $t_1 = c \cdot s_{21}$; $t_2 = (c + d) \cdot (s_{21} + s_{20})$; $su_{20} = t_0$; $su_{21} = t_2 - t_0 - t_1$; $su_{22} = t_1 + s_{20}$; $su_{23} = s_{21}$; $t_0 = j \cdot su_{20}$; $t_1 = i \cdot su_{21}$; $t_2 = (i + j) \cdot (su_{20} + su_{21})$; $d_{00} = t_0$; $d_{01} = t_2 - t_0 - t_1$; $d_{02} = t_1$; $t_0 = j \cdot (su_{20} + su_{22})$; $t_1 = i \cdot (su_{21} + su_{23})$; $t_2 = (i + j) \cdot (su_{20} + su_{21} + su_{22} + su_{23})$; $d_{20} = t_0$; $d_{21} = t_2 - t_0 - t_1$; $d_{22} = t_1$; $suv_{20} = d_{00}$; $suv_{21} = d_{01}$; $suv_{22} = d_{02} + d_{20} - d_{00}$; $suv_{23} = d_{21} - d_{01}$; $suv_{24} = d_{22} - d_{02}$; $v'_3 = suv_{13} + suv_{23} - u'_3 \cdot (suv_{14} + suv_{24})$; $v'_2 = suv_{12} + suv_{22} - u'_2 \cdot (suv_{14} + suv_{24})$; $v'_1 = suv_{11} + suv_{21} - u'_1 \cdot (suv_{14} + suv_{24})$; $v'_0 = suv_{10} + suv_{20} - u'_0 \cdot (suv_{14} + suv_{24})$;	$22M$
4	Compute monic $u3 = (f - v'h - v'^2)/u' = x^2 + a_3 x + b_3$: $t_1 = -{v'_3}^2$; $a_3 = -2 \cdot v'_3 \cdot v'_2 + 1 - v'_3 \cdot h_2 - t_1 \cdot u'_3$; $b_3 = -v'_2 \cdot h_2 - v'_3 \cdot h_1 - 2 \cdot v'_3 \cdot v'_1 + f_4 - {v'_2}^2 - t_1 \cdot u'_2 - a_3 \cdot u'_3$; $t_0 = {t_1}^{-1}$; $a_3 = a_3 \cdot t_0$; $b_3 = b_3 \cdot t_0$;	$I + 7M + 2S$
5	Compute $v_3 = -(h + v') \bmod u_3 = i_3 x + j_3$: $t_0 = -v'_3$; $t_1 = -(v'_2 + h_2) - t_0 \cdot a_3$; $i_3 = -(v'_1 + h_1) - (t_0 + t_1) \cdot (b_3 + a_3) + t_1 \cdot b_3 + t_0 \cdot a_3$; $j_3 = -(v'_0 + h_0) - t_1 \cdot b_3$;	$5M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_4 = 0$ fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_4 = 0$	$2I + 44M + 4S$ $2I + 42M + 4S$

Table A.5: Explicit formulae for doubling on a HEC of genus two (Cantor).

Input	Weight two reduced divisors $D = (u, v)$ with $u = x^2 + ax + b$; $v = ix + j$; furthermore: $h = h_2x^2 + h_1x + h_0$; where $h_i \in \{0, 1\}$; $f = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$; where $f_4 \in \{0, 1\}$;	
Output	A weight two reduced divisor $D' = (u', v') = [2]D$ with $u' = x^2 + a_2x + b_2$; $v' = i_2x + j_2$;	
Step	Procedure	Cost
1	Compute $gcd(u_1, h + 2v_1) = 1 = s_1u_1 + s_3(h + 2v_1)$: $r_{11} = h_1 + 2 \cdot i - h_2 \cdot a$; $r_{10} = h_0 + 2 \cdot j - h_2 \cdot b$; $r_{21} = r_{11} \cdot a - r_{10}$; $r_{20} = r_{11} \cdot b$; $r_{30} = r_{21} \cdot r_{10} - r_{11} \cdot r_{20}$; $s_{31} = r_{11}$; $s_{30} = r_{21}$; $t_0 = r_{30}^{-1}$; $s_{31} = s_{31} \cdot t_0$; $s_{30} = s_{30} \cdot t_0$; $s_{11} = -s_{31} \cdot h_2$; $s_{10} = -2 \cdot s_{31} \cdot i - s_{31} \cdot h_1 - s_{30} \cdot h_2 - a \cdot s_{11}$;	$I + 8M$
2	Compute $u' = {u_1}^2 = x^4 + u'_3x^3 + u'_2x^2 + u'_1x + u'_0$: $t_0 = b^2$; $t_1 = a^2$; $t_2 = (a + b)^2$; $u'_0 = t_0$; $u'_1 = t_2 - t_0 - t_1$; $u'_2 = 2 \cdot b + t_1$; $u'_3 = 2 \cdot a$;	$3S$
3	Compute $v' = s_1u_1v_1 + s_3({v_1}^2 + f) \bmod u'_1$: $t_0 = j \cdot s_{10}$; $t_1 = i \cdot s_{11}$; $t_2 = (i + j) \cdot (s_{10} + s_{11})$; $sv_0 = t_0$; $sv_1 = t_2 - t_1 - t_0$; $sv_2 = t_1$; $t_0 = b \cdot sv_0$; $t_1 = a \cdot sv_1$; $t_2 = (a + b) \cdot (sv_0 + sv_1)$; $d_{00} = t_0$; $d_{01} = t_2 - t_0 - t_1$; $d_{02} = t_1$; $d_{10} = sv_2$; $t_0 = (b + 1) \cdot (sv_0 + sv_2)$; $t_1 = a \cdot sv_1$; $t_2 = (a+b+1) \cdot (sv_0 + sv_1 + sv_2)$; $d_{20} = t_0$; $d_{21} = t_2 - t_0 - t_1$; $d_{22} = t_1$; $suv_0 = d_{00}$; $suv_1 = d_{01}$; $suv_2 = d_{02} + d_{20} - d_{00} - d_{10}$; $suv_3 = d_{21} - d_{01}$; $suv_4 = d_{10} + d_{22} - d_{02}$; $t_0 = j^2$; $t_1 = i^2$; $t_2 = (i + j)^2$; $vq_0 = t_0$; $vq_1 = t_2 - t_0 - t_1$; $vsq_2 = t_1$; $vsf_3 = f_3 - u'_2 - f_4 \cdot u'_3 + {u'_3}^2$; $vsf_2 = vsq_2 + f_2 - u'_1 -$ $f_4 \cdot u'_2 + u'_3 \cdot u'_2$; $vsf_1 = vsq_1 + f_1 - u'_0 - f_4 \cdot u'_1 + u'_3 \cdot u'_1$; $vsf_0 = vsq_0 + f_0 - f_4 \cdot u'_0 + u'_3 \cdot u'_0$; $t_0 = s_{30} \cdot vsf_0$; $t_1 = s_{31} \cdot vsf_1$; $t_2 = (s_{30} + s_{31}) \cdot (vsf_0 + vsf_1)$; $d_{00} = t_0$; $d_{01} = t_2 - t_0 - t_1$; $d_{02} = t_1$; $t_0 = (vsf_2 + vsf_0) \cdot s_{30}$; $t_1 = (vsf_3 + vsf_1) \cdot s_{31}$; $t_2 = (vsf_3 + vsf_2 + vsf_1 + vsf_0) \cdot (s_{30} + s_{31})$; $d_{20} = t_0$; $d_{21} = t_2 - t_0 - t_1$; $d_{22} = t_1$; $svf_0 = d_{00}$; $svf_1 = d_{01}$; $svf_2 = d_{02} + d_{20} - d_{00}$; $svf_3 = d_{21} - d_{01}$; $svf_4 = d_{22} - d_{02}$; $v'_3 = suv_3 + svf_3 - u'_3 \cdot (suv_4 + svf_4)$; $v'_2 = suv_2 + svf_2 -$ $u'_2 \cdot (suv_4 + svf_4)$; $v'_1 = suv_1 + svf_1 - u'_1 \cdot (suv_4 + svf_4)$; $v'_0 = suv_0 + svf_0 - u'_0 \cdot (suv_4 + svf_4)$;	$22M + 4S$

Table A.5: (continued)

Step	Procedure	Cost
4	Compute monic $u_3 = (f - v'h - v'^2)/u' = x^2 + a_2x + b_2$: $t_1 = -v'_3$; $a_2 = -2 \cdot v'_3 \cdot v'_2 + 1 - v'_3 \cdot h_2 - t_1 \cdot u'_3$; $b_2 = -v'_2 \cdot h_2 - v'_3 \cdot h_1 - 2 \cdot v'_3 \cdot v'_1 + f_4 - {v'_2}^2 - t_1 \cdot u'_2 - a_2 \cdot u'_3$; $t_0 = {t_1}^{-1}$; $a_2 = a_2 \cdot t_0$; $b_2 = b_2 \cdot t_0$;	$I + 7M + S$
5	Compute $v_3 = -(h + v') \bmod u_3 = i_2x + j_2$: $t_0 = -vs3$; $t_1 = -(v'_2 + h_2) - t_0 \cdot a_2$; $i_2 = -(v'_1 + h_1) - (t_0 + t_1) \cdot (b_2 + a_2) + t_1 \cdot b_2 + t_0 \cdot a_2$; $j_2 = -(v'_0 + h_0) - t_1 \cdot b_2$;	$5M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_4 = 0$ fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_4 = 0$ fields of characteristic two, $h(x) = x$, $f_4 = 0$	$2I + 42M + 8S$ $2I + 40M + 8S$ $I + 23M + 6S$

Table A.6: Explicit formulae for adding on a HEC of genus three (Harley).

Input	Weight three reduced divisors $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$ with $u_1 = x^3 + ax^2 + bx + c$; $u_2 = x^3 + dx^2 + ex + f$; $v_1 = kx^2 + lx + m$; $v_2 = nx^2 + ox + p$; furthermore: $h = h_3x^3 + h_2x^2 + h_1x + h_0$ where $h_i \in \{0, 1\}$; $f = x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ where $f_6 \in \{0, 1\}$;	
Output	A weight three reduced divisor $D_3 = (u_3, v_3) = D_1 + D_2$ with $u_3 = x^3 + a_3x^2 + b_3x + c_3$; $v_3 = k_3x^2 + l_3x + m_3$;	
Step	Procedure	Cost
1	Compute resultant r of u_1 and u_2 (Bezout): $t_1 = ae$; $t_2 = bd$; $t_3 = bf$; $t_4 = ce$; $t_5 = af$; $t_6 = cd$; $t_7 = (f - c)^2$; $t_8 = (e - b)^2$; $t_9 = (d - a)(t_3 - t_4)$; $t_{10} = (d - a)(t_5 - t_6)$; $t_{11} = (e - b)(f - c)$; $r = (f - c + t_1 - t_2)(t_7 - t_9) + (t_5 - t_6)(t_{10} - 2t_{11}) + t_8(t_3 - t_4)$;	$12M + 2S$
2	Compute almost inverse $inv = r/u_1 \bmod u_2$: $inv_2 = (t_1 - t_2 - c + f)(d - a) - t_8$; $inv_1 = inv_2d - t_{10} + t_{11}$; $inv_0 = inv_2e - d(t_{10} - t_{11}) + t_9 - t_7$	$4M$

Table A.6: (continued)

Step	Procedure	Cost
3	Compute $s' = rs \equiv (v_2 - v_1)inv \bmod u_2$ (Karatsuba): $t_{12} = (inv_1 + inv_2)(n - k + o - l)$; $t_{13} = (o - l)inv_1$; $t_{14} = (inv_0 + inv_2)(n - k + p - m)$; $t_{15} = (p - m)inv_0$; $t_{16} = (inv_0 + inv_1)(o - l + p - m)$; $t_{17} = (n - k)inv_2$; $r'_0 = t_{15}$; $r'_1 = t_{16} - t_{13} - t_{15}$; $r'_2 = t_{13} + t_{14} - t_{15} - t_{17}$; $r'_3 = t_{12} - t_{13} - t_{17}$; $r'_4 = t_{17}$; $t_{18} = dr'_4 - r'_3$; $s'_0 = r'_0 + ft_{18}$; $s'_1 = r'_1 - (e + f)(r'_4 - t_{18}) + er'_4 - ft_{18}$; $s'_2 = r'_2 - er'_4 + dt_{18}$; If $s'_2 = 0$ perform Cantor	$11M$
4	Compute $s = (s'/r)$ and make s monic: $w_1 = (rs'_2)^{-1}$; $w_2 = rw_1$; $w_3 = w_1 {s'_2}^2$; $w_4 = rw_2$; $w_5 = w_4^2$; $s_0 = w_2 s'_0$; $s_1 = w_2 s'_1$;	$I + 6M + 2S$
5	Compute $z = su_1$: $z_0 = s_0 c$; $z_1 = s_1 c + s_0 b$; $z_2 = s_0 a + s_1 b + c$; $z_3 = s_1 a + s_0 + b$; $z_4 = a + s_1$;	$6M$
6	Compute $u' = [s(z + w_4(h + 2v_1))$ $-w_5((f - v_1 h - v_1^2)/u_1)]/u_2$: $u'_3 = z_4 + s_1 - d$; $u'_2 = -du'_3 - e + z_3 + s_0 + w_4 h_3 + s_1 z_4$; $u'_1 = w_4(h_2 + 2k + s_1 h_3) + s_1 z_3 + s_0 z_4 + z_2 - w_5 - du'_2 - eu'_3 - f$; $u'_0 = w_4(s_1 h_2 + h_1 + 2l + 2s_1 k + s_0 h_3) + s_1 z_2 + z_1 + s_0 z_3 +$ $w_5(a - f_6) - du'_1 - eu'_2 - fu'_3$	$15M$
7	Compute $v' = -(w_3 z + h + v_1) \bmod u'$: $t_1 = u'_3 - z_4$; $v'_0 = -w_3(u'_0 t_1 + z_0) - h_0 - m$; $v'_1 = -w_3(u'_1 t_1 - u'_0 + z_1) - h_1 - l$; $v'_2 = -w_3(u'_2 t_1 - u'_1 + z_2) - h_2 - k$; $v'_3 = -w_3(u'_3 t_1 - u'_2 + z_3) - h_3$;	$8M$
8	Reduce u', i.e. $u_3 = (f - v'h - v'^2)/u'$: $a_3 = f_6 - u'_3 - {v'_3}^2 - v'_3 h_3$; $b_3 = -u'_2 - a_3 u'_3 + f_5 - 2v'_2 v'_3 - v'_3 h_2 - v'_2 h_3$; $c_3 = -u'_1 - a_3 u'_2 - b_3 u'_3 + f_4 - 2v'_1 v'_3 - {v'_2}^2 - v'_2 h_2 - v'_3 h_1 - v'_1 h_3$;	$5M + 2S$
9	Compute $v_3 = -(v' + h) \bmod u_3$: $k_3 = -v'_2 + (v'_3 + h_3)a_3 - h_2$; $l_3 = -v'_1 + (v'_3 + h_3)b_3 - h_1$; $m_3 = -v'_0 + (v'_3 + h_3)c_3 - h_0$;	$3M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_6 = 0$ fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_6 = 0$	$I + 70M + 6S$ $I + 65M + 6S$

Table A.7: Explicit formulae for doubling on a HEC of genus three (Harley).

Input	A weight three reduced divisors $D_1 = (u_1, v_1)$ with $u_1 = x^3 + ax^2 + bx + c$; $v_1 = kx^2 + lx + m$; furthermore: $h = h_3x^3 + h_2x^2 + h_1x + h_0$ where $h_i \in \{0, 1\}$; $f = x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ where $f_6 \in \{0, 1\}$;	
Output	A weight three reduced divisor $D_2 = (u_2, v_2) = [2]D_1$ with $u_2 = x^3 + a_2x^2 + b_2x + c_2$; $v_2 = k_2x^2 + l_2x + m_2$;	
Step	Procedure	Cost
1	Compute resultant r of u_1 and $h + 2v_1$ (Bezout): let $\tilde{h} = h + 2v_1$: $t_1 = a\tilde{h}_1$; $t_2 = b\tilde{h}_2$; $t_3 = b\tilde{h}_0$; $t_4 = c\tilde{h}_1$; $t_5 = a\tilde{h}_0$; $t_6 = c\tilde{h}_2$; $t_7 = (\tilde{h}_0 - h_3c)^2$; $t_8 = (\tilde{h}_1 - h_3b)^2$; $t_9 = (\tilde{h}_2 - h_3a)(t_3 - t_4)$; $t_{10} = (\tilde{h}_2 - h_3a)(t_5 - t_6)$; $t_{11} = (\tilde{h}_1 - h_3b)(\tilde{h}_0 - h_3c)$; $r = (\tilde{h}_0 - h_3c + t_1 - t_2)(t_7 - t_9) + (t_5 - t_6)[(t_5 - t_6)(\tilde{h}_2 - h_3a) - 2t_{11}] + t_8(t_3 - t_4)$;	$6M + 2S$
2	Compute almost inverse $inv = r/(h + 2v_1) \bmod u_1$: $inv_2 = -(t_1 - t_2 - h_3c + \tilde{h}_0)(\tilde{h}_2 - h_3a) + t_8$; $inv_1 = inv_2a + t_{10} - t_{11}$; $inv_0 = inv_2b + a(t_{10} - t_{11}) - t_9 + t_7$	$4M$
3	Compute $z = ((f - hv_1 - v_1^2)/u_1) \bmod u_1$: $t_{12} = k^2$; $z'_3 = f_6 - a$; $t_{13} = z'_3b$; $z'_2 = f_5 - h_3k - b - az'_3$; $z'_1 = f_4 - h_2k - h_3l - t_{12} - c - t_{13} - z'_2a$; $z_2 = f_5 - h_3k - 2b + a(a - 2z'_3)$; $z_1 = z'_1 - t_{13} + ab - c$; $z_0 = f_3 - h_2l - h_1k - 2kl - h_3m + c(a - 2z'_3) - z'_2b - z'_1a$;	$7M + 2S$
4	Compute $s' = zinv \bmod u_1$ (Karatsuba): $t_{12} = (inv_1 + inv_2)(z_1 + z_2)$; $t_{13} = z_1inv_1$; $t_{14} = (inv_0 + inv_2)(z_0 + z_2)$; $t_{15} = z_0inv_0$; $t_{16} = (inv_0 + inv_1)(z_0 + z_1)$; $t_{17} = z_2inv_2$; $r'_0 = t_{15}$; $r'_1 = t_{16} - t_{13} - t_{15}$; $r'_2 = t_{13} + t_{14} - t_{15} - t_{17}$; $r'_3 = t_{12} - t_{13} + t_{17}$; $r'_4 = t_{17}$; $t_{18} = ar'_4 - r'_3$; $s'_0 = r'_0 + ct_{18}$; $s'_1 = r'_1 - (b + c)(r'_4 - t_{18}) + br'_4 - ct_{18}$; $s'_2 = r'_2 - br'_4 + at_{18}$; If $s'_2 = 0$ perform Cantor	$11M$

Table A.7: (continued)

Step	Procedure	Cost
5	Compute $s = (s'/r)$ and make s monic: $w_1 = (rs_2')^{-1}$; $w_2 = w_1 r$; $w_3 = w_1(s_2')^2$; $w_4 = w_2 r$; $(= r/s_2')$; $w_5 = w_4^2$ $s_0 = w_2 s_0'$; $s_1 = w_2 s_1'$;	$I + 6M + 2S$
6	Compute $G = su_1$: $g_0 = s_0 c$; $g_1 = s_1 c + s_0 b$; $g_2 = s_0 a + s_1 b + c$; $g_3 = s_1 a + s_0 + b$; $g_4 = a + s_1$;	$6M$
7	Compute $u' = u_1^{-2}[(G + w_4 v_1)^2 + w_4 hG + w_5(hv_1 - f)]$: $u_3' = 2s_1$; $u_2' = s_1^2 + 2s_0 + w_4 h_3$; $u_1' = 2s_0 s_1 + w_4(2k + h_3 s_1 + h_2 - h_3 a) - w_5$; $u_0' = w_4[2l + h_1 + h_3 s_0 - h_3 b + 2ks_1 + a(ah_3 - 2k - h_2 - s_1 h_3) + h_2 s_1] + w5(-f_6 + 2a) + s_0^2$;	$6M + 2S$
8	Compute $v' = -(Gw_3 + h + v_1) \bmod u'$: $t_1 = u_3' - g_4$; $v_3' = -(t_1 u_3' - u_2' + g_3)w_3 - h_3$; $v_2' = -(t_1 u_2' - u_1' + g_2)w_3 - h_2 - k$; $v_1' = -(t_1 u_1' - u_0' + g_1)w_3 - h_1 - l$; $v_0' = -(t_1 u_0' + g_0)w_3 - h_0 - m$;	$8M$
9	Reduce u', i.e. $u_2 = (f - v'h - v'^2)/u'$: $a_2 = f_6 - u_3' - {v_3'}^2 - v_3' h_3$; $b_2 = -u_2' - a_2 u_3' + f_5 - 2v_2' v_3' - v_3' h_2 - v_2' h_3$; $c_2 = -u_1' - a_2 u_2' - b_2 u_3' + f_4 - 2v_1' v_3' - {v_2'}^2 - v_2' h_2 - v_3' h_1 - v_1' h_3$;	$5M + 2S$
10	Compute $v_2 = -(v' + h) \bmod u_2$: $k_2 = -v_2' + (v_3' + h_3)a_2 - h_2$; $l_2 = -v_1' + (v_3' + h_3)b_2 - h_1$; $m_2 = -v_0' + (v_3' + h_3)c_2 - h_0$;	$3M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}$, $f_6 = 0$ fields of characteristic two, $h_i \in \mathbb{F}$, $f_6 = 0$ fields of characteristic two, $h(x) = 1$, $f_6 = 0$	$I + 62M + 10S$ $I + 53M + 10S$ $I + 22M + 7S$

Table A.8: Explicit formulae for adding on a HEC of genus three (Cantor).

Input	Weight three reduced divisors $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$ with $u_1 = x^3 + ax^2 + bx + c$; $u_2 = x^3 + dx^2 + ex + f$; $v_1 = kx^2 + lx + m$; $v_2 = nx^2 + ox + p$; furthermore: $h = h_3x^3 + h_2x^2 + h_1x + h_0$ where $h_i \in \{0, 1\}$; $f = x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ where $f_6 \in \{0, 1\}$;	
Output	A weight three reduced divisor $D_3 = (u_3, v_3) = D_1 + D_2$ with $u_3 = x^3 + a_3x^2 + b_3x + c_3$; $v_3 = k_3x^2 + l_3x + m_3$;	
Step	Procedure	Cost
1	Compute $gcd(u_1, u_2) = s_1u_1 + s_2u_2$ (EEA): $r_{22} = a - d$; $r_{21} = b - e$; $r_{20} = c - f$; $t_{20} = -1$; $r_{32} = r_{22} \cdot d - r_{21}$; $r_{31} = r_{22} \cdot e - r_{20}$; $r_{30} = r_{22} \cdot f$; $t_{31} = 1$; $t_{30} = r_{22}$; $r_{41} = r_{32} \cdot r_{21} - r_{22} \cdot r_{31}$; $r_{40} = r_{32} \cdot r_{20} - r_{22} \cdot r_{30}$; $t_{41} = -r_{22} \cdot t_{31}$; $t_{40} = r_{32} \cdot t_{20} - r_{22}^2$; $r_{51} = r_{41} \cdot r_{31} - r_{32} \cdot r_{40}$; $r_{50} = r_{41} \cdot r_{30}$; $t_{52} = -r_{32} \cdot t_{41}$; $t_{51} = r_{41} \cdot t_{31} - r_{32} \cdot t_{40}$; $t_{50} = r_{41} \cdot t_{30}$; $r_{60} = r_{51} \cdot r_{40} - r_{41} \cdot r_{50}$; $s_{22} = r_{51} \cdot t_{42} - r_{41} \cdot t_{52}$; $s_{21} = r_{51} \cdot t_{41} - r_{41} \cdot t_{51}$; $s_{20} = r_{51} \cdot t_{40} - r_{41} \cdot t_{50}$; $t_0 = r_{60}^{-1}$; $s_{22} = s_{22} \cdot t_0$; $s_{21} = s_{21} \cdot t_0$; $s_{20} = s_{20} \cdot t_0$; $s_{12} = -s_{22}$; $s_{11} = -s_{22} \cdot d - s_{21} - a \cdot s_{12}$; $s_{10} = -s_{22} \cdot e - s_{21} \cdot d - s_{20} - a \cdot s_{11} - b \cdot s_{12}$;	$I + 33M + S$
2	Compute $u' = u_1u_2/d^2 = u_1u_2$ (Karatsuba): $t_0 = c \cdot f$; $t_1 = b \cdot e$; $t_2 = (c + b) \cdot (e + f)$; $d_{00} = t_0$; $d_{01} = t_2 - t_1 - t_0$; $d_{02} = t_1$; $d_{10} = a \cdot d$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (a + c) \cdot (d + f)$; $t_1 = d_{02}$; $t_2 = (a + b + c) \cdot (d + e + f)$; $d_{20} = t_0$; $d_{21} = t_2 - t_1 - t_0$; $d_{22} = t_1$; $u'_0 = d_{00}$; $u'_1 = d_{01}$; $u'_2 = d_{20} - d_{10} - d_{00} + d_{02}$; $u'_3 = d_{21} - d_{01} + c + f$; $u'_4 = d_{10} + d_{22} - d_{02} + b + e$; $u'_5 = a + d$;	$6M$
3	Compute $v' = s_1u_1v_2 + s_2u_2v_1 \bmod u'$: $t_0 = c \cdot s_{10}$; $t_1 = b \cdot s_{11}$; $t_2 = (c + b) \cdot (s_{10} + s_{11})$; $d_{00} = t_0$; $d_{01} = t_2 - t_1 - t_0$; $d_{02} = t_1$; $d_{10} = a \cdot s_{12}$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (a + c) \cdot (s_{12} + s_{10})$; $t_1 = d_{02}$; $t_2 = (a + b + c) \cdot (s_{12} + s_{11} + s_{10})$; $d_{20} = t_0$; $d_{21} = t_2 - t_1 - t_0$; $d_{22} = t_1$; $su_{10} = d_{00}$; $su_{11} = d_{01}$; $su_{12} = d_{20} - d_{10} - d_{00} + d_{02}$; $su_{13} = d_{21} - d_{01} + s_{10}$; $su_{14} = d_{10} + d_{22} - d_{02} + s_{11}$; $su_{15} = s_{12}$;	$45M$

Table A.8: (continued)

Step	Procedure	Cost
	$t_0 = su_{10} \cdot p$; $t_1 = su_{11} \cdot o$; $t_2 = (su_{10} + su_{11}) \cdot (o + p)$; $d_{00} = t_0$; $d_{01} = t_2 - t_1 - t_0$; $d_{02} = t_1$; $d_{10} = su_{12} \cdot n$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (p + n) \cdot (su_{10} + su_{12})$; $t_1 = d_{02}$; $t_2 = (n+o+p) \cdot (su_{12}+su_{11}+su_{10})$; $d_{20} = t_0$; $d_{21} = t_2-t_1-t_0$; $d_{22} = t_1$; $d_{100} = d_{00}$; $d_{101} = d_{01}$; $d_{102} = d_{02} + d_{20} - d_{10} - d_{00}$; $d_{103} = d_{21} - d_{01}$; $d_{104} = d_{10} + d_{22} - d_{02}$; $t_0 = (su_{10} + su_{13}) \cdot p$; $t_1 = (su_{11} + su_{14}) \cdot o$; $t_2 = (su_{10} + su_{13} + su_{11} + su_{14}) \cdot (o + p)$; $d_{00} = t_0$; $d_{01} = t_2 - t_1 - t_0$; $d_{02} = t_1$; $d_{10} = (su_{12} + su_{15}) \cdot n$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (su_{10}+su_{13}+su_{12}+su_{15}) \cdot (n+p)$; $t_1 = d_{02}$; $t_2 = (su_{10}+su_{13}+ su_{11}+su_{14}+su_{12}+su_{15}) \cdot (n+o+p)$; $d_{20} = t_0$; $d_{21} = t_2-t_1-t_0$; $d_{22} = t_1$; $d_{120} = d_{00}$; $d_{121} = d_{01}$; $d_{122} = d_{02} + d_{20} - d_{10} - d_{00}$; $d_{123} = d_{21} - d_{01}$; $d_{124} = d_{10} + d_{22} - d_{02}$; $suv_{10} = d_{100}$; $suv_{11} = d_{101}$; $suv_{12} = d_{102}$; $suv_{13} = d_{120} - d_{100} + d_{103}$; $suv_{14} = d_{121} - d_{101} + d_{104}$; $suv_{15} = d_{122} - d_{102}$; $suv_{16} = d_{123} - d_{103}$; $suv_{17} = d_{124} - d_{104}$; $t_0 = f \cdot s_{20}$; $t_1 = e \cdot s_{21}$; $t_2 = (f + e) \cdot (s_{20} + s_{21})$; $d_{00} = t_0$; $d_{01} = t_2 - t_1 - t_0$; $d_{02} = t_1$; $d_{10} = d \cdot s_{22}$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (d + f) \cdot (s_{22} + s_{20})$; $t_1 = d_{02}$; $t_2 = (d + e + f) \cdot (s_{22} + s_{21} + s_{20})$; $d_{20} = t_0$; $d_{21} = t_2 - t_1 - t_0$; $d_{22} = t_1$; $su_{20} = d_{00}$; $su_{21} = d_{01}$; $su_{22} = d_{20}-d_{10}-d_{00}+d_{02}$; $su_{23} = d_{21}-d_{01}+s_{20}$; $su_{24} = d_{10} + d_{22} - d_{02} + s_{21}$; $su_{25} = s_{22}$; $t_0 = su_{20} \cdot m$; $t_1 = su_{21} \cdot l$; $t_2 = (su_{20} + su_{21}) \cdot (l + m)$; $d_{00} = t_0$; $d_{01} = t_2 - t_1 - t_0$; $d_{02} = t_1$; $d_{10} = su_{22} \cdot k$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (m + k) \cdot (su_{20} + su_{22})$; $t_1 = d_{02}$; $t_2 = (k + l + m) \cdot (su_{22} + su_{21} + su_{20})$; $d_{20} = t_0$; $d_{21} = t_2 - t_1 - t_0$; $d_{22} = t_1$; $d_{100} = d_{00}$; $d_{101} = d_{01}$; $d_{102} = d_{02} + d_{20} - d_{10} - d_{00}$; $d_{103} = d_{21} - d_{01}$; $d_{104} = d_{10} + d_{22} - d_{02}$; $t_0 = (su_{20}+su_{23}) \cdot m$; $t_1 = (su_{21}+su_{24}) \cdot l$; $t_2 = (su_{20}+su_{23}+ su_{21} + su_{24}) \cdot (l + m)$; $d_{00} = t_0$; $d_{01} = t_2 - t_1 - t_0$; $d_{02} = t_1$; $d_{10} = (su_{22} + su_{25}) \cdot k$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (su_{20} + su_{23} + su_{22} + su_{25}) \cdot (k + m)$; $t_1 = d_{02}$; $t_2 = (su_{20} + su_{23} + su_{21} + su_{24} + su_{22} + su_{25}) \cdot (k + l + m)$; $d_{20} = t_0$; $d_{21} = t_2 - t_1 - t_0$; $d_{22} = t_1$; $d_{120} = d_{00}$; $d_{121} = d_{01}$; $d_{122} = d_{02} + d_{20} - d_{10} - d_{00}$; $d_{123} = d_{21} - d_{01}$; $d_{124} = d_{10} + d_{22} - d_{02}$; $suv_{20} = d_{100}$; $suv_{21} = d_{101}$; $suv_{22} = d_{102}$; $suv_{23} = d_{120} - d_{100} + d_{103}$; $suv_{24} = d_{121} - d_{101} + d_{104}$; $suv_{25} = d_{122} - d_{102}$; $suv_{26} = d_{123} - d_{103}$; $suv_{27} = d_{124} - d_{104}$;	

Table A.8: (continued)

Step	Procedure	Cost
	$c_0 = suv_{10} + suv_{20}$; $c_1 = suv_{11} + suv_{21}$; $c_2 = suv_{12} + suv_{22}$; $c_3 = suv_{13} + suv_{23}$; $c_4 = suv_{14} + suv_{24}$; $c_5 = suv_{15} + suv_{25}$; $c_6 = suv_{16} + suv_{26}$; $c_7 = suv_{17} + suv_{27}$; $t_0 = c_7$; $t_2 = t_0 \cdot u'_5$; $t_1 = c_6 - t_2$; $t_3 = t_1 \cdot u'_4$; $t_4 = t_0 \cdot u'_3$; $t_5 = t_1 \cdot u'_2$; $t_6 = t_0 \cdot u'_1$; $t_7 = t_1 \cdot u'_0$; $v'_5 = c_5 - (t_0 + t_1) \cdot (u'_4 + u'_5) + t_3 + t_2$; $v'_4 = c_4 - t_4 - t_3$; $v'_3 = c_3 - (t_0 + t_1) \cdot (u'_2 + u'_3) + t_5 + t_4$; $v'_2 = c_2 - t_6 - t_5$; $v'_1 = c_1 - (t_0 + t_1) \cdot (u'_0 + u'_1) + t_7 + t_6$; $v'_0 = c_0 - t_7$;	
4	Compute $u_3 = (f - v'h - v'^2)/u' = u_{34}x^4 + u_{33}x^3 + u_{32}x^2 + u_{31}x + u_{30}$ and make u_3 monic: $u_{34} = -v'^2_5$; $u_{33} = -2 \cdot v'_5 \cdot v'_4 - u_{34} \cdot u'_5$; $u_{32} = -v'^2_4 - v'_5 \cdot h_3 - 2 \cdot v'_5 \cdot v'_3 - u_{34} \cdot u'_4 - u_{33} \cdot u'_5$; $u_{31} = -2 \cdot v'_5 \cdot v'_2 + 1 - 2 \cdot v'_4 \cdot v'_3 - v'_4 \cdot h_3 - v'_5 \cdot h_2 - u_{33} \cdot u'_4 - u_{32} \cdot u'_5 - u_{34} \cdot u'_3$; $u_{30} = -v'_3 - 2 \cdot v'_5 \cdot v'_1 - v'_3 \cdot h_3 - v'_5 \cdot h_1 + f_6 - 2 \cdot v'_4 \cdot v'_2 - v'_4 \cdot h_2 - u_{34} \cdot u'_2 - u_{32} \cdot u'_4 - u_{31} \cdot u'_5 - u_{33} \cdot u'_3$; $t_0 = u_{34}^{-1}$; $u_{33} = u_{33} \cdot t_0$; $u_{32} = u_{32} \cdot t_0$; $u_{31} = u_{31} \cdot t_0$; $u_{30} = u_{30} \cdot t_0$;	$I + 20M + 2S$
5	Compute $v_3 = -(v' + h) \bmod u_3$, where $v_3 = v_{33}x^3 + v_{32}x^2 + v_{31}x + v_{30}$: $t_0 = -v'_5$; $t_2 = t_0 \cdot u_{33}$; $t_1 = -v'_4 - t_2$; $t_3 = t_1 \cdot u_{32}$; $t_4 = t_0 \cdot u_{31}$; $t_5 = t_1 \cdot u_{30}$; $v_{33} = -(v'_3 + h_3) - (t_0 + t_1) \cdot (u_{32} + u_{33}) + t_3 + t_2$; $v_{32} = -(v'_2 + h_2) - t_4 - t_3$; $v_{31} = -(v'_1 + h_1) - (t_0 + t_1) \cdot (u_{30} + u_{31}) + t_5 + t_4$; $v_{30} = -(v'_0 + h_0) - t_5$;	$6M$
6	Compute $u4 = (f - v_3h - v_3^2)/u_3 = x^3 + a_3x^2 + b_3x + c_3$: $a_3 = -v_{33} \cdot h_3 - v_{33} + f_6 - u_{33}$; $b_3 = -v_{32} \cdot h_3 - v_{33} \cdot h_2 - 2 \cdot v_{33} \cdot v_{32} + f_5 - u_{32} - a_3 \cdot u_{33}$; $c_3 = -v_{32} \cdot h_2 - 2 \cdot v_{33} \cdot v_{31} - v_{33} \cdot h_1 - v_{31} \cdot h_3 + f_4 - v_{32}^2 - u_{31} - a_3 \cdot u_{32} - b_3 \cdot u_{33}$;	$5M + S$
7	Compute $v_4 = -(v_3 + h) \bmod u_4 = k_3x^2 + l_3x + m_3$: $t_0 = -(v_{33} + h_3)$; $k_3 = -(v_{32} + h_2) - t_0 \cdot a_3$; $l_3 = -(v_{31} + h_1) - t_0 \cdot b_3$; $m_3 = -(v_{30} + h_0) - t_0 \cdot c_3$;	$3M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_6 = 0$	$2I + 118M + 4S$
	fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_6 = 0$	$2I + 110M + 4S$

Table A.9: Explicit formulae for doubling on a HEC of genus three (Cantor).

Input	A weight three reduced divisors $D_1 = (u_1, v_1)$ with $u_1 = x^3 + ax^2 + bx + c$; $v_1 = kx^2 + lx + m$; furthermore: $h = h_3x^3 + h_2x^2 + h_1x + h_0$ where $h_i \in \{0, 1\}$; $f = x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ where $f_6 \in \{0, 1\}$;	
Output	A weight three reduced divisor $D_2 = (u_2, v_2) = [2]D_1$ with $u_2 = x^3 + a_2x^2 + b_2x + c_2$; $v_2 = k_2x^2 + l_2x + m_2$;	
Step	Procedure	Cost
1	Compute $gcd(u_1, 2v_1 + h) = s_1u_1 + s_3(2v_1 + h)$: $r_{12} = h_2 + 2 \cdot k - h_3 \cdot a$; $r_{11} = h_1 + 2 \cdot l - h_3 \cdot b$; $r_{10} = h_0 + 2 \cdot m - h_3 \cdot c$; $r_{22} = r_{12} \cdot a - r_{11}$; $r_{21} = r_{12} \cdot b - r_{10}$; $r_{20} = r_{12} \cdot c$; $t_{20} = 0$; $t_{21} = -1$; $r_{31} = r_{22} \cdot r_{11} - r_{12} \cdot r_{21}$; $r_{30} = r_{22} \cdot r_{10} - r_{12} \cdot r_{20}$; $t_{31} = r_{12}$; $t_{30} = r_{22}$; $r_{41} = r_{31} \cdot r_{21} - r_{22} \cdot r_{30}$; $r_{40} = r_{31} \cdot r_{20}$; $t_{42} = -r_{22} \cdot t_{31}$; $t_{41} = -r_{31} - r_{22} \cdot t_{30}$; $t_{40} = 0$; $r_{50} = r_{41} \cdot r_{30} - r_{31} \cdot r_{40}$; $s_{32} = -r_{31} \cdot t_{42}$; $s_{31} = r_{41} \cdot t_{31} - r_{31} \cdot t_{41}$; $s_{30} = r_{41} \cdot t_{30}$; $t_0 = r_{50}^{-1}$; $s_{32} = s_{32} \cdot t_0$; $s_{31} = s_{31} \cdot t_0$; $s_{30} = s_{30} \cdot t_0$; $s_{12} = -s_{32} \cdot h_3$; $s_{11} = -2 \cdot s_{32} \cdot k - s_{32} \cdot h_2 - s_{31} \cdot h_3 - a \cdot s_{12}$; $s_{10} = -2 \cdot s_{32} \cdot l - s_{32} \cdot h_1 - 2 \cdot s_{31} \cdot k - s_{31} \cdot h_2 - s_{30} \cdot h_3 - a \cdot s_{11} - b \cdot s_{12}$;	$I + 27M$
2	Compute $u_1 = u^2 = x^6 + u_{15}x^5 + u_{14}x^4 + u_{13}x^3 + u_{12}x^2 + u_{11}x + u_{10}$: $d_{00} = c^2$; $d_{02} = b^2$; $d_{01} = (b + c)^2 - d_{00} - d_{02}$; $d_{10} = a^2$; $d_{12} = 1$; $d_{11} = (a + 1)^2 - d_{10} - d_{12}$; $d_{20} = (a + c)^2$; $d_{22} = (b + 1)^2$; $d_{21} = (a + b + c + 1)^2 - d_{20} - d_{22}$; $u_{10} = d_{00}$; $u_{11} = d_{01}$; $u_{12} = d_{02} + d_{20} - d_{10} - d_{00}$; $u_{13} = d_{21} - d_{11} - d_{01}$; $u_{14} = d_{22} - d_{12} - d_02 + d_{10}$; $u_{15} = d_{11}$;	$8S$
3	Compute $v' = s_1u_1v_1 + s_3(v_1^2 + f) \bmod u_1$: $t_0 = m \cdot s_{10}$; $t_1 = l \cdot s_{11}$; $t_2 = (s_{11} + s_{10}) \cdot (l + m)$; $d_{00} = t_0$; $d_{01} = t_2 - t_0 - t_1$; $d_{02} = t_1$; $t_0 = k \cdot s_{12}$; $t_1 = 0$; $t_2 = t_0$; $d_{10} = t_0$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (s_{10} + s_{12}) \cdot (m + k)$; $t_1 = d_{02}$; $t_2 = (s_{12} + s_{11} + s_{10}) \cdot (k + l + m)$; $d_{20} = t_0$; $d_{21} = t_2 - t_0 - t_1$; $d_{22} = t_1$; $sv_0 = d_{00}$; $sv_1 = d_{01}$; $sv_2 = d_{02} + d_{20} - d_{10} - d_{00}$; $sv_3 = d_{21} - d_{01}$; $sv_4 = d_{10} + d_{22} - d_{12} - d_{02}$;	$44M + 7S$

Table A.9: (continued)

Step	Procedure	Cost
	$t_0 = c \cdot sv_0$; $t_1 = b \cdot sv_1$; $t_2 = (sv_1 + sv_0) \cdot (b + c)$; $d_{00} = t_0$; $d_{01} = t_2 - t_0 - t_1$; $d_{02} = t_1$; $d_{10} = a \cdot sv_2$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (sv_0 + sv_2) \cdot (a + c)$; $t_1 = sv_1 \cdot b$; $t_2 = (sv_0 + sv_1 + sv_2) \cdot (a + b + c)$; $d_{20} = t_0$; $d_{21} = t_2 - t_0 - t_1$; $d_{22} = t_1$; $d_{100} = d_{00}$; $d_{101} = d_{01}$; $d_{102} = d_{02} + d_{20} - d_{10} - d_{00}$; $d_{103} = d_{21} - d_{01}$; $d_{104} = d_{10} + d_{22} - d_{12} - d_{02}$; $t_0 = c \cdot (sv_0 + sv_3)$; $t_1 = b \cdot (sv_1 + sv_4)$; $t_2 = (sv_1 + sv_4 + sv_0 + sv_3) \cdot (c + b)$; $d_{00} = t_0$; $d_{01} = t_2 - t_0 - t_1$; $d_{02} = t_1$; $d_{10} = d_{10}$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (sv_0 + sv_3 + sv_2) \cdot (a + c)$; $t_1 = d_{02}$; $t_2 = (sv_2 + sv_1 + sv_4 + sv_0 + sv_3) \cdot (a + b + c)$; $d_{20} = t_0$; $d_{21} = t_2 - t_0 - t_1$; $d_{22} = t_1$; $d_{120} = d_{00}$; $d_{121} = d_{01}$; $d_{122} = d_{02} + d_{20} - d_{10} - d_{00}$; $d_{123} = d_{21} - d_{01}$; $d_{124} = d_{10} + d_{22} - d_{12} - d_{02}$; $suv_0 = d_{100}$; $suv_1 = d_{101}$; $suv_2 = d_{102}$; $suv_3 = d_{103} + d_{120} - d_{100} + sv_0$; $suv_4 = d_{104} + d_{121} - d_{101} + sv_1$; $suv_5 = d_{122} - d_{102} + sv_2$; $suv_6 = d_{123} - d_{103} + sv_3$; $suv_7 = d_{124} - d_{104} + sv_4$; $d_{00} = m^2$; $d_{02} = l^2$; $d_{01} = (l+m)^2 - d_{00} - d_{02}$; $d_{10} = k^2$; $d_{12} = 0$; $d_{11} = 0$; $d_{20} = (m + k)^2$; $d_{22} = d_{02}$; $d_{21} = (k + l + m)^2 - d_{20} - d_{22}$; $vsq_0 = d_{00}$; $vsq_1 = d_{01}$; $vsq_2 = d_{02} + d_{20} - d_{10} - d_{00}$; $vsq_3 = d_{21} - d_{01}$; $vsq_4 = d_{10} + d_{22} - d_{02}$; $vf_5 = f_5 - u_{14} - (f_6 - u_{15}) \cdot u_{15}$; $vf_4 = (vsq_4 + f_4) - u_{13} - (f_6 - u_{15}) \cdot u_{14}$; $vf_3 = (vsq_3 + f_3) - u_{12} - (f_6 - u_{15}) \cdot u_{13}$; $vf_2 = (vsq_2 + f_2) - u_{11} - (f_6 - u_{15}) \cdot u_{12}$; $vf_1 = (vsq_1 + f_1) - u_{10} - (f_6 - u_{15}) \cdot u_{11}$; $vf_0 = (vsq_0 + f_0) - (f_6 - u_{15}) \cdot u_{10}$; $t_0 = s_{30} \cdot vf_0$; $t_1 = s_{31} \cdot vf_1$; $t_2 = (vf_0 + vf_1) \cdot (s_{31} + s_{30})$; $d_{00} = t_0$; $d_{01} = t_2 - t_0 - t_1$; $d_{02} = t_1$; $d_{10} = s_{32} \cdot vf_2$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (vf_0 + vf_2) \cdot (s_{30} + s_{32})$; $t_1 = d_{02}$; $t_2 = (vf_0 + vf_1 + vf_2) \cdot (s_{30} + s_{31} + s_{32})$; $d_{20} = t_0$; $d_{21} = t_2 - t_0 - t_1$; $d_{22} = t_1$; $d_{100} = d_{00}$; $d_{101} = d_{01}$; $d_{102} = d_{02} + d_{20} - d_{10} - d_{00}$; $d_{103} = d_{21} - d_{01}$; $d_{104} = d_{10} + d_{22} - d_{12} - d_{02}$; $t_0 = s_{30} \cdot (vf_0 + vf_3)$; $t_1 = s_{31} \cdot (vf_1 + vf_4)$; $t_2 = (s_{30} + s_{31}) \cdot (vf_0 + vf_3 + vf_1 + vf_4)$; $d_{00} = t_0$; $d_{01} = t_2 - t_0 - t_1$; $d_{02} = t_1$; $d_{10} = s_{32} \cdot (vf_2 + vf_5)$; $d_{11} = 0$; $d_{12} = 0$; $t_0 = (vf_0 + vf_3 + vf_2 + vf_5) \cdot (s_{32} + s_{30})$; $t_1 = (vf_1 + vf_4) \cdot s_{31}$; $t_2 = (vf_0 + vf_1 + vf_2 + vf_3 + vf_4 + vf_5) \cdot (s_{30} + s_{31} + s_{32})$; $d_{20} = t_0$; $d_{21} = t_2 - t_0 - t_1$; $d_{22} = t_1$; $d_{120} = d_{00}$; $d_{121} = d_{01}$; $d_{122} = d_{02} + d_{20} - d_{10} - d_{00}$; $d_{123} = d_{21} - d_{01}$; $d_{124} = d_{10} + d_{22} - d_{12} - d_{02}$; $svf_0 = d_{100}$; $svf_1 = d_{101}$; $svf_2 = d_{102}$; $svf_3 = d_{103} + d_{120} - d_{100}$; $svf_4 = d_{104} + d_{121} - d_{101}$; $svf_5 = d_{122} - d_{102}$; $svf_6 = d_{123} - d_{103}$; $svf_7 = d_{124} - d_{104}$;	

Table A.9: (continued)

Step	Procedure	Cost
	$t_0 = suv_7 + svf_7$; $t_2 = t_0 \cdot u_{15}$; $t_1 = (suv_6 + svf_6) - t_2$; $t_3 = t_1 \cdot u_{14}$; $t_4 = t_0 \cdot u_{13}$; $t_5 = t_1 \cdot u_{12}$; $t_6 = t_0 \cdot u_{11}$; $t_7 = t_1 \cdot u_{10}$; $v_{15} = (suv_5 + svf_5) - (t_0 + t_1) \cdot (u_{14} + u_{15}) + t_2 + t_3$; $v_{14} = (suv_4 + svf_4) - t_4 - t_3$; $v_{13} = (suv_3 + svf_3) - (t_0 + t_1) \cdot (u_{12} + u_{13}) + t_4 + t_5$; $v_{12} = (suv_2 + svf_2) - t_6 - t_5$; $v_{11} = (suv_1 + svf_1) - (t_0 + t_1) \cdot (u_{11} + u_{10}) + t_6 + t_7$; $v_{10} = (suv_0 + svf_0) - t_7$;	
4	Compute $u_3 = (f - v_1 h - v_1^2)/u_1$ and make u_3 monic, $u_3 = x^4 + u_{33}x^3 + u_{32}x^2 + u_{31}x + u_{30}$: $u_{34} = -v_{15}^2$; $u_{33} = -2 \cdot v_{15} \cdot v_{14} - u_{34} \cdot u_{15}$; $u_{32} = -v_{14}^2 - v_{15} \cdot h_3 - 2 \cdot v_{15} \cdot v_{13} - u_{34} \cdot u_{14} - u_{33} \cdot u_{15}$; $u_{31} = -2 \cdot v_{15} \cdot v_{12} + 1 - 2 \cdot v_{14} \cdot v_{13} - v_{14} \cdot h_3 - v_{15} \cdot h_2 - u_{33} \cdot u_{14} - u_{32} \cdot u_{15} - u_{34} \cdot u_{13}$; $u_{30} = -v_{13} - 2 \cdot v_{15} \cdot v_{11} - v_{13} \cdot h_3 - v_{15} \cdot h_1 + f_6 - 2 \cdot v_{14} \cdot v_{12} - v_{14} \cdot h_2 - u_{34} \cdot u_{12} - u_{32} \cdot u_{14} - u_{31} \cdot u_{15} - u_{33} \cdot u_{13}$; $t_0 = u_{34}^{-1}$; $u_{33} = u_{33} \cdot t_0$; $u_{32} = u_{32} \cdot t_0$; $u_{31} = u_{31} \cdot t_0$; $u_{30} = u_{30} \cdot t_0$;	$I + 20M + 2S$
5	Compute $v_3 = -(v' + h) \bmod u_3$, where $v_3 = v_{33}x^3 + v_{32}x^2 + v_{31}x + v_{30}$: $t_0 = -v_{15}$; $t_2 = t_0 \cdot u_{33}$; $t_1 = -v_{14} - t_2$; $t_3 = t_1 \cdot u_{32}$; $t_4 = t_0 \cdot u_{31}$; $t_5 = t_1 \cdot u_{30}$; $v_{33} = -(v_{13} + h_3) - (t_0 + t_1) \cdot (u_{32} + u_{33}) + t_3 + t_2$; $v_{32} = -(v_{12} + h_2) - t_4 - t_3$; $v_{31} = -(v_{11} + h_1) - (t_0 + t_1) \cdot (u_{30} + u_{31}) + t_5 + t_4$; $v_{30} = -(v_{10} + h_0) - t_5$;	$6M$
6	Compute $u_4 = (f - v_3 h - v_3^2)/u_3 = x^3 + a_2x^2 + b_2x + c_2$: $a_2 = -v_{33} \cdot h_3 - v_{33}^2 + f_6 - u_{33}$; $b_2 = -v_{32} \cdot h_3 - v_{33} \cdot h_2 - 2 \cdot v_{33} \cdot v_{32} + f_5 - u_{32} - a_2 \cdot u_{33}$; $c_2 = -v_{32} \cdot h_2 - 2 \cdot v_{33} \cdot v_{31} - v_{33} \cdot h_1 - v_{31} \cdot h_3 + f_4 - v_{32}^2 - u_{31} - a_2 \cdot u_{32} - b_2 \cdot u_{33}$;	$6M + 2S$
7	Compute $v_4 = -(v_3 + h) \bmod u_4 = k_2x^2 + l_2x + m_2$: $t_0 = -(v_{33} + h_3)$; $k_2 = -(v_{32} + h_2) - t_0 \cdot a_2$; $l_2 = -(v_{31} + h - 1) - t_0 \cdot b_2$; $m_2 = -(v_{30} + h - 0) - t_0 \cdot c_2$;	$3M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_6 = 0$	$2I + 106M + 19S$
	fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_6 = 0$	$2I + 98M + 13S$
	fields of characteristic two, $h(x) = 1$, $f_6 = 0$, see Table A.10	$I + 14M + 11S$

Table A.10: Explicit formulae for doubling on special curves of genus three over $\mathbb{F}_{2^n}$ with $h(x) = 1$ (Cantor).

Input	A weight three reduced divisors $D_1 = (u_1, v_1)$ with $u_1 = x^3 + ax^2 + bx + c$; $v_1 = kx^2 + lx + m$; furthermore: $h = h_0 = 1$ $f = x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ where $f_6 \in \{0, 1\}$;	
Output	A weight three reduced divisor $D_2 = (u_2, v_2) = [2]D_1$ with $u_2 = x^3 + a_2x^2 + b_2x + c_2$; $v_2 = k_2x^2 + l_2x + m_2$;	
Step	Procedure	Cost
1	Compute $d = gcd(u_1, 1) = 1 = s_1a + s_3h$ $(s3 = 1, s1 = 0)$: $s_3 = 1$; $s_1 = 0$;	$-$
2	Compute $u' = u_1^2$: $t_1 = a^2$; $t_2 = b^2$; $t_3 = c^2$;	$3S$
3	Compute $v' = v_1^2 + f \bmod u'$: $t_4 = k^2$; $t_5 = l^2$; $t_6 = m^2$; $v'_5 = f_5 + t_1$; $v'_4 = f_4 + t_4$; $v'_3 = f_3 + t_2$; $v'_2 = f_2 + t_5$; $v'_1 = f_1 + t_3$; $v'_0 = f_0 + t_6$;	$3S$
4	Compute $u'' = ((f - hv' - v'^2)/u')$: $u'_4 = {v'_5}^2$; $u'_3 = 0$; $u'_2 = {v'_4}^2 + t_1 \cdot u'_4$; $u'_1 = 1$; $u'_0 = v'_3 + t_2 \cdot u'_4 + t_1 \cdot u'_2$;	$3M + 3S$
5	Compute $u_2 = u''$ made monic $= x^4 + u_{22}x^2 + u_{21}x + u_{20}$: $u_{21} = {u'_4}^{-1}$; $u_{22} = u'_2 \cdot u_{21}$; $u_{20} = u'_0 \cdot u_{21}$;	$1I + 2M$
6	Compute $v_2 = -(v' + h) \bmod u_2$, where $v_2 = v_{23}x^3 + v_{22}x^2 + v_{21}x + v_{20}$: $t_1 = v'_5 \cdot u_{22}$; $t_2 = v'_4 \cdot u_{21}$; $t_3 = (v'_5 + v'_4) \cdot (u_{22} + u_{21})$; $v_{23} = v'_3 + t_1$; $v_{22} = t_3 + t_1 + t_2 + v'_2$; $v_{21} = t_2 + v'_5 \cdot u_{20} + v'_1$; $v_{20} = v'_4 \cdot u_{20} + v'_0 + 1$;	$5M$
7	Compute $u_3 := (f - v_2h - v_2^2)/u_2 = x^3 + u_{32}x^2 + u_{31}x + u_{30}$: $a_2 = v_{23}^2$; $b_2 = u_{22} + f_5$; $c_2 = u_{22} \cdot a_2 + f_4 + v_{22}^2 + u_{21}$;	$1M + 2S$
8	Compute $v_3 := -(v_2 + h) \bmod u_3 = v_{32}x^2 + v_{31}x + v_{30}$: $k_2 = v_{22} + v_{23} \cdot a_2$; $l_2 = v_{21} + v_{23} \cdot b_2$; $m_2 = v_{20} + v_{23} \cdot c_2 + 1$;	$3M$
Total		$I + 14M + 11S$

Table A.11: Explicit formulae for adding on a HEC of genus four (Harley).

Input	Weight four reduced divisors $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$ $u_1 = x^4 + ax^3 + bx^2 + cx + d$; $v_1 = ix^3 + jx^2 + kx + l$ $u_2 = x^4 + ex^3 + fx^2 + gx + h$; $v_2 = mx^3 + nx^2 + ox + p$ $h = h_4x^4 + h_3x^3 + h_2x^2 + h_1x + h_0$ where $h_i \in \{0, 1\}$; $f = x^9 + f_8x^8 + f_7x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ where $f_8 \in \{0, 1\}$;	
Output	A weight four reduced divisor $D_3 = (u_3, v_3) = D_1 + D_2$ $u_3 = x^4 + a_3x^3 + b_3x^2 + c_3x + d_3$; $v_3 = i_3x^3 + j_3x^2 + k_3x + l_3$	
Step	Procedure	Cost
1	Almost inverse $inv = r/u_1 \bmod u_2$, wher $u_2 = inv_3x^3 + inv_2x^2 + inv_1x + inv_0$: $r_{23} = a - e$; $r_{22} = b - f$; $r_{21} = c - g$; $r_{20} = d - h$; $r_{33} = r_{23} \cdot a - r_{22}$; $r_{32} = r_{23} \cdot b - r_{21}$; $r_{31} = r_{23} \cdot c - r_{20}$; $r_{30} = r_{23} \cdot d$; $r_{42} = r_{33} \cdot r_{22} - r_{23} \cdot r_{32}$; $r_{41} = r_{33} \cdot r_{21} - r_{23} \cdot r_{31}$; $r_{40} = r_{33} \cdot r_{20} - r_{23} \cdot r_{30}$; $t_{41} = r_{23}$; $t_{40} = r_{33} - r_{23}^2$; $r_{52} = r_{42} \cdot r_{32} - r_{33} \cdot r_{41}$; $r_{51} = r_{42} \cdot r_{31} - r_{33} \cdot r_{40}$; $r_{50} = r_{42} \cdot r_{30}$; $t_{52} = -r_{33} \cdot t_{41}$; $t_{51} = -r_{42} - r_{33} \cdot t_{40}$; $t_{50} = r_{42} \cdot r_{23}$; $r_{61} = r_{52} \cdot r_{41} - r_{42} \cdot r_{51}$; $r_{60} = r_{52} \cdot r_{40} - r_{42} \cdot r_{50}$; $t_{62} = -r_{42} \cdot t_{52}$; $t_{61} = r_{52} \cdot t_{41} - r_{42} \cdot t_{51}$; $t_{60} = r_{52} \cdot t_{40} - r_{42} \cdot t_{50}$; $r_{71} = r_{61} \cdot r_{51} - r_{52} \cdot r_{60}$; $r_{70} = r_{61} \cdot r_{50}$; $t_{73} = -r_{52} \cdot t_{62}$; $t_{72} = r_{61} \cdot t_{52} - r_{52} \cdot t_{61}$; $t_{71} = r_{61} \cdot t_{51} - r_{52} \cdot t_{60}$; $t_{70} = r_{61} \cdot t_{50}$; $r_{80} = r_{71} \cdot r_{60} - r_{61} \cdot r_{70}$; $inv_3 = -r_{61} \cdot t_{73}$; $inv_2 = r_{71} \cdot t_{62} - r_{61} \cdot t_{72}$; $inv_1 = r_{71} \cdot t_{61} - r_{61} \cdot t_{71}$; $inv_0 = r_{71} \cdot t_{60} - r_{61} \cdot t_{70}$; If $inv_0 = 0$ call Cantor's Algorithm;	$45M + 1S$
2	$s' = r \cdot s \equiv (v_2 - v_1)inv \bmod u_2 = s'_3x^3 + s'_2x^2 + s'_1x + s_0$ (Karatsuba): $t_a = m - i$; $t_b = n - j$; $t_c = o - k$; $t_d = p - l$; $t_e = inv_3$; $t_f = inv_2$; $t_g = inv_1$; $t_h = inv_0$; $t_0 = t_c \cdot t_g$; $t_1 = t_b \cdot t_f$; $t_2 = t_a \cdot t_e$; $t_3 = t_b \cdot t_g$; $t_4 = t_c \cdot t_f$; $t_{10} = t_d \cdot t_h$; $t_{11} = (t_c + t_d) \cdot (t_g + t_h) - t_0 - t_{10}$; $t_{12} = (t_b + t_d) \cdot (t_f + t_h) - t_{10} - t_1 + t_0$; $t_{13} = (t_a + t_d) \cdot (t_e + t_h) - t_{10} - t_2 + t_3 + t_4$; $t_{14} = (t_a + t_c) \cdot (t_e + t_g) - t_2 - t_0 + t_1$; $t_{15} = (t_a + t_b) \cdot (t_e + t_f) - t_2 - t_1$; $t_{16} = t_2$; $t_{17} = t_{15} - e \cdot t_{16}$; $t_{18} = e \cdot t_{17} + t_{16} \cdot f - t_{14}$; $s'_3 = e \cdot t_{18} - f \cdot t_{17} - g \cdot t_{16} + t_{13}$; $s'_2 = f \cdot t_{18} - g \cdot t_{17} - h \cdot t_{16} + t_{12}$; $s'_1 = g \cdot t_{18} - h \cdot t_{17} + t_{11}$; $s'_0 = h \cdot t_{18} + t_{10}$;	$23M$

Table A.11: (continued)

Step	Procedure	Cost
3	$\underline{s = s' \text{ made monic} = x^3 + s_2x^2 + s_1x + s_0:}$ $t_1 = r_{80} \cdot s'_3$; $w_6 = t_1^{-1}$; $w_7 = r_{80} \cdot w_6$; $w_4 = r_{80} \cdot w_7$; $w_3 = s'^2_3 \cdot w_6$; $w_5 = w_4^2$; $s_0 = s'_0 \cdot w_7$; $s_1 = s'_1 \cdot w_7$; $s_2 = s'_2 \cdot w_7$;	$I + 7M + 2S$
4	$\underline{z = s \cdot u_1 = z_7x^7 + z_6x^6 + z_5x^5 + z_4x^4 + z_3x^3 + z_2x^2 + z_1x + z_0:}$ $t_0 = c \cdot s_1$; $t_1 = b \cdot s_2$; $z_0 = s_0 \cdot d$; $z_1 = (c+d) \cdot (s_1+s_0) - t_0 - z_0$; $z_2 = (b+d) \cdot (s_2+s_0) - z_0 - t_1 + t_0$; $z_3 = (a+d) \cdot (1+s_0) - z_0 - a + b \cdot s_1 + c \cdot s_2$; $z_4 = (a+c) \cdot (1+s_1) - a - t_0 + t_1 + s_0$; $z_5 = (a+b) \cdot (1+s_2) - a - t_1 + s_1$; $z_6 = a + s_2$; $z_7 = 1$;	$10M$
5	$\underline{u' = [s(z + w_4(h + 2v_1)) - w_5((f - v_1h - v_1^2)/u_1)]/u_2 = x^6 + u'_5x^5 + u'_4x^4 + u'_3x^3 + u'_2x^2 + u'_1x + u'_0:}$ $t_1 = s_2 \cdot w_4$; $t_2 = s_1 \cdot w_4$; $diff_4 = s_2 \cdot z_6 + z_5 + s_1 - f$; $diff_3 = -g + z_4 + s_0 + w_4 \cdot h_4 + s_2 \cdot z_5 + s_1 \cdot z_6$; $diff_2 = -h + z_3 + s_2 \cdot (w_4 \cdot h_4 + z_4) + w_4 \cdot (h_3 + 2 \cdot i) + s_1 \cdot z_5 + s_0 \cdot z_6$; $diff_1 = t_1 \cdot (h_3 + 2 \cdot i) + s_1 \cdot (w_4 \cdot h_4 + z_4) + w_4 \cdot (h_2 + 2 \cdot j) + s_2 \cdot z_3 + s_0 \cdot z_5 + z_2 - w_5$; $diff_0 = t_1 \cdot (h_2 + 2 \cdot j) + t_2 \cdot (h_3 + 2 \cdot i) + s_0 \cdot (w_4 \cdot h_4 + z_4) + w_4 \cdot (h_1 + 2 \cdot k) + s_2 \cdot z_2 + s_1 \cdot z_3 - w_5 \cdot (f_8 - a) + z_1$; $u'_5 = z_6 + s_2 - e$; $u'_4 = diff_4 - e \cdot u'_5$; $u'_3 = diff_3 - e \cdot u'_4 - f \cdot u'_5$; $u'_2 = diff_2 - e \cdot u'_3 - f \cdot u'_4 - g \cdot u'_5$; $u'_1 = diff_1 - e \cdot u'_2 - f \cdot u'_3 - g \cdot u'_4 - h \cdot u'_5$; $u'_0 = diff_0 - e \cdot u'_1 - f \cdot u'_2 - g \cdot u'_3 - h \cdot u'_4$;	$35M$
6	$\underline{v' = -(w_3z + h + v_1) \bmod u' = v'_5x^5 + v'_4x^4 + v'_3x^3 + v'_2x^2 + v'_1x + v'_0:}$ $t_1 = u'_5 - z_6$; $v'_5 = -w_3 \cdot (u'_5 \cdot t_1 - u'_4 + z_5)$; $v'_4 = -w_3 \cdot (u'_4 \cdot t_1 - u'_3 + z_4) - h_4$; $v'_3 = -w_3 \cdot (u'_3 \cdot t_1 - u'_2 + z_3) - h_3 - i$; $v'_2 = -w_3 \cdot (u'_2 \cdot t_1 - u'_1 + z_2) - h_2 - j$; $v'_1 = -w_3 \cdot (u'_1 \cdot t_1 - u'_0 + z_1) - h_1 - k$; $v'_0 = -w_3 \cdot (u'_0 \cdot t_1 + z_0) - h_0 - l$;	$12M$
7	$\underline{u'_3 = (f - v'h - v'^2)/u' = u_{34}x^4 + u_{33}x^3 + u_{32}x^2 + u_{31}x + u_{30}:}$ $diff_3 = 1 - v'_5 \cdot (h_4 + 2 \cdot v'_4)$; $diff_2 = f_8 - v'_5 \cdot (h_3 + 2 \cdot v'_3) - v'_4 \cdot h_4 - v'^2_4$; $diff_1 = f_7 - v'_5 \cdot (h_2 + 2 \cdot v'_2) - v'_4 \cdot (h_3 + 2 \cdot v'_3) - v'_3 \cdot h_4$; $diff_0 = f_6 - v'_5 \cdot (h_1 + 2 \cdot v'_1) - v'_4 \cdot (h_2 + 2 \cdot v'_2) - v'_3 \cdot h_3 - v'_2 \cdot h_4 - v'^2_3$;	$16M + 3S$

Table A.11: (continued)

Step	Procedure	Cost
	$u_{34} = -v_5'^2$; $u_{33} = diff_3 - u_{34} \cdot u_5'$; $u_{32} = diff_2 - u_{34} \cdot u_4' - u_{33} \cdot u_5'$; $u_{31} = diff_1 - u_{34} \cdot u_3' - u_{33} \cdot u_4' - u_{32} \cdot u_5'$; $u_{30} = diff_0 - u_{34} \cdot u_2' - u_{33} \cdot u_3' - u_{32} \cdot u_4' - u_{31} \cdot u_5'$;	
8	$u_3 = u_3'$ made monic $= x^4 + a_3x^3 + b_3x^2 + c_3x + d_3$: $t_0 = u_{34}^{-1}$; $a_3 = u_{33} \cdot t_0$; $b_3 = u_{32} \cdot t_0$; $c_3 = u_{31} \cdot t_0$; $d_3 = u_{30} \cdot t_0$;	$I + 4M$
9	$v_3 = -(v' + h) \bmod u_3 = i_3x^3 + j_3x^2 + k_3x + l_3$ (Almost Karatsuba): $t_0 = -v_5'$; $t_2 = t_0 \cdot a_3$; $t_1 = -v_4' - h_4 - t_2$; $t_3 = t_1 \cdot b_3$; $t_4 = t_0 \cdot c_3$; $t_5 = t_1 \cdot d_3$; $i_3 = -(t_0 + t_1) \cdot (a_3 + b_3) + t_3 + t_2 - v_3' - h_3$; $j_3 = -t_3 - t_4 - v_2' - h_2$; $k_3 = -(t_0+t_1)\cdot(c_3+d_3)+t_5+t_4-v_1'-h_1$; $l_3 = -t_5-v_0'-h_0$;	$6M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_8 = 0$ fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_8 = 0$	$2I + 158M + 6S$ $2I + 146M + 6S$

Table A.12: Explicit formulae for doubling on a HEC of genus four (Harley).

Input	A weight four reduced divisor $D_1 = (u_1, v_1)$ $u_1 = x^4 + ax^3 + bx^2 + cx + d$; $v_1 = ix^3 + jx^2 + kx + l$ $h = h_4x^4 + h_3x^3 + h_2x^2 + h_1x + h_0$ where $h_i \in \{0, 1\}$; $f = x^9+f_8x^8+f_7x^7+f_6x^6+f_5x^5+f_4x^4+f_3x^3+f_2x^2+f_1x+f_0$ where $f_8 \in \{0, 1\}$;	
Output	A weight four reduced divisor $D_2 = (u_2, v_2) = [2]D_1$ $u_2 = x^4 + a_2x^3 + b_2x^2 + c_2x + d_2$; $v_2 = i_2x^3 + j_2x^2 + k_2x + l_2$	
Step	Procedure	Cost
1	Almost inverse $inv = r_{80}/(h + 2v_1) \bmod u_1$ $= inv_3x^3 + inv_2x^2 + inv_1x + inv_0$ (EEA): $r_{13} = h_3 + 2 \cdot i - h_4 \cdot a$; $r_{12} = h_2 + 2 \cdot j - h_4 \cdot b$; $r_{11} = h_1 + 2 \cdot k - h_4 \cdot c$; $r_{10} = h_0 + 2 \cdot l - h_4 \cdot d$; $r_{23} = r_{13} \cdot a - r_{12}$; $r_{22} = r_{13} \cdot b - r_{11}$; $r_{21} = r_{13} \cdot c - r_{10}$; $r_{20} = r_{13} \cdot d$; $t_{21} = -1$; $t_{20} = 0$; $r_{32} = r_{23} \cdot r_{12} - r_{13} \cdot r_{22}$; $r_{31} = r_{23} \cdot r_{11} - r_{13} \cdot r_{21}$; $r_{30} = r_{23} \cdot r_{10} - r_{13} \cdot r_{20}$; $t_{31} = r_{13}$; $t_{30} = r_{23}$; $r_{42} = r_{32} \cdot r_{22} - r_{23} \cdot r_{31}$;	$I + 44M$

Table A.12: (continued)

Step	Procedure	Cost
	$r_{41} = r_{32} \cdot r_{21} - r_{23} \cdot r_{30}$; $r_{40} = r_{32} \cdot r_{20}$; $t_{42} = -r_{23} \cdot t_{31}$; $t_{41} = r_{32} \cdot t_{21} - r_{23} \cdot t_{30}$; $t_{40} = 0$; $r_{51} = r_{42} \cdot r_{31} - r_{32} \cdot r_{41}$; $r_{50} = r_{42} \cdot r_{30} - r_{32} \cdot r_{40}$; $t_{52} = -r_{32} \cdot t_{42}$; $t_{51} = r_{42} \cdot t_{31} - r_{32} \cdot t_{41}$; $t_{50} = r_{42} \cdot t_{30}$; $r_{61} = r_{51} \cdot r_{41} - r_{42} \cdot r_{50}$; $r_{60} = r_{51} \cdot r_{40}$; $t_{63} = -r_{42} \cdot t_{52}$; $t_{62} = r_{51} \cdot t_{42} - r_{42} \cdot t_{51}$; $t_{61} = r_{51} \cdot t_{41} - r_{42} \cdot t_{50}$; $t_{60} = r_{51} \cdot t_{40}$; $r_{80} = r_{61} \cdot r_{50} - r_{51} \cdot r_{60}$; $inv_3 = -r_{51} \cdot t_{63}$; $inv_2 = r_{61} \cdot t_{52} - r_{51} \cdot t_{62}$; $inv_1 = r_{61} \cdot t_{51} - r_{51} \cdot t_{61}$; $inv_0 = r_{61} \cdot t_{50} - r_{51} \cdot t_{60}$; If $r_{80} = 0$ call Cantor's Algorithm;	
2	$\underline{z = ((f - hv_1 - v_1^2)/u_1) \bmod u_1,}$ $\underline{\text{ where } z = z_4x^4 + z_3x^3 + z_2x^2 + z_1x + z_0\text{:}}$ $diff_3 = f_7 - i \cdot h_4 - b$; $diff_2 = f_6 - i^2 - i \cdot h_3 - j \cdot h_4 - c$; $diff_1 = f_5 - i \cdot h_2 - j \cdot h_3 - k \cdot h_4 - 2 \cdot i \cdot j - d$; $diff_0 = f_4 - j^2 - i \cdot h_1 - j \cdot h_2 - k \cdot h_3 - l \cdot h_4 - 2 \cdot i \cdot k$; $z'_4 = f_8 - a$; $z'_3 = diff_3 - a \cdot z'_4$; $z'_2 = diff_2 - a \cdot z'_3 - b \cdot z'_4$; $z'_1 = diff_1 - a \cdot z'_2 - b \cdot z'_3 - c \cdot z'_4$; $z'_0 = diff_0 - a \cdot z'_1 - b \cdot z'_2 - c \cdot z'_3 - d \cdot z'_4$; $z_3 = a \cdot (a - z'_4) - b + z'_3$; $z_2 = b \cdot (a - z'_4) - c + z'_2$; $z_1 = c \cdot (a - z'_4) - d + z'_1$; $z_0 = d \cdot (a - z'_4) + z'_0$;	$12M + 3S$
3	$\underline{s' = z \cdot inv \bmod u_1 = s'_3x^3 + s'_2x^2 + s'_1x + s'_0 \text{ (Karatsuba):}}$ $t_1 = z_1 \cdot inv_1$; $t_2 = z_2 \cdot inv_2$; $t_{10} = z_0 \cdot inv_0$; $t_{16} = z_3 \cdot inv_3$; $t_{15} = (z_3 + z_2) \cdot (inv_3 + inv_2) - t_{16} - t_2$; $t_{14} = (z_3 + z_1) \cdot (inv_3 + inv_1) - t_{16} - t_1 + t_2$; $t_{13} = (z_3 + z_0) \cdot (inv_3 + inv_0) - t_{10} - t_{16} + z_2 \cdot inv_1 + z_1 \cdot inv_2$; $t_{12} = (z_2 + z_0) \cdot (inv_2 + inv_0) - t_{10} - t_2 + t_1$; $t_{11} = (z_1 + z_0) \cdot (inv_1 + inv_0) - t_1 - t_{10}$; $t_3 = t_{15} - a \cdot t_{16}$; $t_4 = a \cdot t_3 + b \cdot t_{16} - t_{14}$; $s'_0 = d \cdot t_4 + t_{10}$; $s'_1 = c \cdot t_4 - d \cdot t_3 + t_{11}$; $s'_2 = b \cdot t_4 - c \cdot t_3 - d \cdot t_{16} + t_{12}$; $s'_3 = a \cdot t_4 - b \cdot t_3 - c \cdot t_{16} + t_{13}$;	$23M$
4	$\underline{s = s' \text{ made monic} = x^3 + s_2x^2 + s_1x + s_0\text{:}}$ $t_1 = r_{80} \cdot s'_3$; $w_6 = t_1^{-1}$; $w_7 = r_{80} \cdot w_6$; $w_4 = r_{80} \cdot w_7$; $w_3 = s'^2_3 \cdot w_6$; $w_5 = w_4^2$; $s_0 = s'_0 \cdot w_7$; $s_1 = s'_1 \cdot w_7$; $s_2 = s'_2 \cdot w_7$;	$I + 7M + 2S$

Table A.12: (continued)

Step	Procedure	Cost
5	$\underline{G = su_1 = x^7 + g_6x^6 + g_5x^5 + g_4x^4 + g_3x^3 + g_2x^2 + g_1x + g_0 \text{ (Karatsuba):}}$ $t_0 = c \cdot s_1$; $t_1 = b \cdot s_2$; $g_0 = s_0 \cdot d$; $g_1 = (c+d) \cdot (s_1+s_0) - t_0 - g_0$; $g_2 = (b+d) \cdot (s_2+s_0) - g_0 - t_1 + t_0$; $g_3 = (a+d) \cdot (1+s_0) - g_0 - a + b \cdot s_1 + c \cdot s_2$; $g_4 = (a+c) \cdot (1+s_1) - a - t_0 + t_1 + s_0$; $g_5 = (a+b) \cdot (1+s_2) - a - t_1 + s_1$; $g_6 = a + s_2$;	$10M$
6	$\underline{u' = u_1^{-2}[(G + w_4v_1)^2 + w_4hG + w_5(hv_1 - f)] = x^6 + u'_5x^5 + u'_4x^4 + u'_3x^3 + u'_2x^2 + u'_1x + u'_0:}$ $sa = a^2$; $ssa = sa^2$; $sma = sa \cdot a$; $ab = a \cdot b$; $ac = a \cdot c$; $bc = b \cdot c$; $sb = b^2$; $sc = c^2$; $sg_5 = g_5^2$; $sg_6 = g_6^2$; $g_5g_6 = g_5 \cdot g_6$; $diff_4 = 2 \cdot g_5 + sg_6 - 2 \cdot b - sa$; $diff_3 = 2 \cdot g_6 \cdot (g_5 - sa) + 2 \cdot g_4 + w_4 \cdot h_4 - 2 \cdot c - 2 \cdot a \cdot b + 2 \cdot ssa)$; $diff_2 = sa \cdot (-2 \cdot g_5 - sg_6 + 6 \cdot b) + g_6 \cdot (4 \cdot sma + 2 \cdot g_4 - 4 \cdot ab + w_4 \cdot h_4) - 2 \cdot d + sg_5 + 2 \cdot g_3 - 2 \cdot ac - 3 \cdot ssa - sb + w_4 \cdot (2 \cdot i + h_3)$; $diff_1 = sa \cdot (6 \cdot c - 2 \cdot g_4 + 12 \cdot b \cdot g_6 - 2 \cdot g_5g_6 - w_4 \cdot h_4) + sma \cdot (-12 \cdot b + 4 \cdot g_5 + 2 \cdot sg_6) + ssa \cdot (-6 \cdot g_6 + 4 \cdot a) + w_4 \cdot (2 \cdot g_6 \cdot i + h_4 \cdot g_5 + h_3 \cdot g_6 + h_2 + 2 \cdot j) + g_6 \cdot (-4 \cdot ac - 2 \cdot sb + 2 \cdot g_3) + ab \cdot (-4 \cdot g_5 - 2 \cdot sg_6) + a \cdot (-2 \cdot d + 6 \cdot sb) + 2 \cdot g_2 - w_5 + 2 \cdot g_5 \cdot g_4 - 2 \cdot bc$; $diff_0 = g_6 \cdot (-4 \cdot bc - 4 \cdot d \cdot a + w_4 \cdot h_2 + 2 \cdot w_4 \cdot j + 2 \cdot g_2 - 24 \cdot sma \cdot b + 12 \cdot sa \cdot c - 2 \cdot sa \cdot g_4 + 8 \cdot ssa \cdot a - sa \cdot w_4 \cdot h_4 + 12 \cdot a \cdot sb) + sa \cdot (+6 \cdot b \cdot sg_6 + 12 \cdot b \cdot g_5 - 2 \cdot w_4 \cdot i - 2 \cdot g_3 - sg_5 + 6 \cdot d - 18 \cdot sb - w_4 \cdot h_3) + ssa \cdot (-6 \cdot g_5 - 3 \cdot sg_6 + 20 \cdot b) + sma \cdot (-12 \cdot c + 4 \cdot g_4 + 2 \cdot w_4 \cdot h_4 + 4 \cdot g_5g_6 - 5 \cdot sma) + sb \cdot (2 \cdot b - 2 \cdot g_5 - sg_6) + w_4 \cdot (+2 \cdot g_5 \cdot i + h_4 \cdot (g_4 - 2 \cdot ab) + h_1 + 2 \cdot k + h_3 \cdot g_5) + ab \cdot (-4 \cdot g_5g_6 + 12 \cdot c - 4 \cdot g_4) + ac \cdot (-4 \cdot g_5 - 2 \cdot sg_6) + 2 \cdot g_1 - w_5 \cdot f_8 + 2 \cdot g_5 \cdot g_3 - 2 \cdot b \cdot d + g_4^2 - sc$; $u'_5 = 2 \cdot (g_6 - a)$; $u'_4 = diff_4 - 2 \cdot a \cdot u'_5$; $u'_3 = diff_3 - 2 \cdot a \cdot u'_4 - 2 \cdot b \cdot u'_5$; $u'_2 = diff_2 - 2 \cdot a \cdot u'_3 - 2 \cdot b \cdot u'_4 - 2 \cdot c \cdot u'_5$; $u'_1 = diff_1 - 2 \cdot a \cdot u'_2 - 2 \cdot b \cdot u'_3 - 2 \cdot c \cdot u'_4 - 2 \cdot d \cdot u'_5$; $u'_0 = diff_0 - 2 \cdot a \cdot u'_1 - 2 \cdot b \cdot u'_2 - 2 \cdot c \cdot u'_3 - 2 \cdot d \cdot u'_4$;	$56M + 7S$
7	$\underline{v' = -(w_3z + h + v_1) \bmod u' = v'_5x^5 + v'_4x^4 + v'_3x^3 + v'_2x^2 + v'_1x + v'_0:}$ $t_1 = u'_5 - g_6$; $v'_5 = -w_3 \cdot (u'_5 \cdot t_1 - u'_4 + g_5)$; $v'_4 = -w_3 \cdot (u'_4 \cdot t_1 - u'_3 + g_4) - h_4$; $v'_3 = -w_3 \cdot (u'_3 \cdot t_1 - u'_2 + g_3) - h_3 - i$; $v'_2 = -w_3 \cdot (u'_2 \cdot t_1 - u'_1 + g_2) - h_2 - j$; $v'_1 = -w_3 \cdot (u'_1 \cdot t_1 - u'_0 + g_1) - h_1 - k$; $v'_0 = -w_3 \cdot (u'_0 \cdot t_1 + g_0) - h_0 - l$;	$12M$

Table A.12: (continued)

Step	Procedure	Cost
8	$u'_2 = (f - v'h - v'^2)/u' = u_{24}x^4 + u_{23}x^3 + u_{22}x^2 + u_{21}x + u_{20}$: $diff_3 = 1 - v'_5 \cdot (h_4 + 2 \cdot v'_4)$; $diff_2 = f_8 - v'_5 \cdot (h_3 + 2 \cdot v'_3) - v'_4 \cdot h_4 - v'^2_4$; $diff_1 = f_7 - v'_5 \cdot (h_2 + 2 \cdot v'_2) - v'_4 \cdot (h_3 + 2 \cdot v'_3) - v'_3 \cdot h_4$; $diff_0 = f_6 - v'_5 \cdot (h_1 + 2 \cdot v'_1) - v'_4 \cdot (h_2 + 2 \cdot v'_2) - v'_3 \cdot h_3 - v'_2 \cdot h_4 - v'^2_3$; $u_{24} = -v'^2_5$; $u_{23} = diff_3 - u_{24} \cdot u'_5$; $u_{22} = diff_2 - u_{24} \cdot u'_4 - u_{23} \cdot u'_5$; $u_{21} = diff_1 - u_{24} \cdot u'_3 - u_{23} \cdot u'_4 - u_{22} \cdot u'_5$; $u_{20} = diff_0 - u_{24} \cdot u'_2 - u_{23} \cdot u'_3 - u_{22} \cdot u'_4 - u_{21} \cdot u'_5$;	$16M + 3S$
9	$u_2 = u'_2$ made monic $= x^4 + a_2x^3 + b_2x^2 + c_2x + d_2$: $t_0 = u_{24}^{-1}$; $a_2 = u_{23} \cdot t_0$; $b_2 = u_{22} \cdot t_0$; $c_2 = u_{21} \cdot t_0$; $d_2 = u_{20} \cdot t_0$;	$I + 4M$
10	$v_2 = -(v' + h) \bmod u_2 = i_2x^3 + j_2x^2 + k_2x + l_2$ (Almost Karatsuba): $t_0 = -v'_5$; $t_2 = t_0 \cdot a_2$; $t_1 = -v'_4 - h_4 - t_2$; $t_3 = t_1 \cdot b_2$; $t_4 = t_0 \cdot c_2$; $t_5 = t_1 \cdot d_2$; $i_2 = -(t_0 + t_1) \cdot (a_2 + b_2) + t_3 + t_2 - v'_3 - h_3$; $j_2 = -t_3 - t_4 - v'_2 - h_2$; $k_2 = -(t_0 + t_1) \cdot (c_2 + d_2) + t_5 + t_4 - v'_1 - h_1$; $l_2 = -t_5 - v'_0 - h_0$;	$6M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_8 = 0$	$2I + 193M + 17S$
	fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_8 = 0$	$2I + 144M + 17S$
	fields of characteristic two, $h(x) = x$, $f_8 = 0$	$2I + 72M + 13S$

Table A.13: Explicit formulae for adding on a HEC of genus four (Cantor).

Input	Weight four reduced divisors $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$ $u_1 = x^4 + ax^3 + bx^2 + cx + d$; $v_1 = ix^3 + jx^2 + kx + l$ $u_2 = x^4 + ex^3 + fx^2 + gx + h$; $v_2 = mx^3 + nx^2 + ox + p$ $h = h_4x^4 + h_3x^3 + h_2x^2 + h_1x + h_0$ where $h_i \in \{0, 1\}$; $f = x^9 + f_8x^8 + f_7x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ where $f_8 \in \{0, 1\}$;	
Output	A weight four reduced divisor $D_3 = (u_3, v_3) = D_1 + D_2$ $u_3 = x^4 + a_3x^3 + b_3x^2 + c_3x + d_3$; $v_3 = i_3x^3 + j_3x^2 + k_3x + l_3$	
Step	Procedure	Cost
1	Compute $gcd(u_1, u_2) = s_1u_1 + s_2u_2$ (EEA):	$I + 59M + S$

Table A.13: (continued)

Step	Procedure	Cost
	$r_{23} = a - e$; $r_{22} = b - f$; $r_{21} = c - g$; $r_{20} = d - h$; $r_{33} = r_{23} \cdot a - r_{22}$; $r_{32} = r_{23} \cdot b - r_{21}$; $r_{31} = r_{23} \cdot c - r_{20}$; $r_{30} = r_{23} \cdot d$; $r_{42} = r_{33} \cdot r_{22} - r_{23} \cdot r_{32}$; $r_{41} = r_{33} \cdot r_{21} - r_{23} \cdot r_{31}$; $r_{40} = r_{33} \cdot r_{20} - r_{23} \cdot r_{30}$; $t_{41} = r_{23}$; $t_{40} = r_{33} - r_{23}^2$; $r_{52} = r_{42} \cdot r_{32} - r_{33} \cdot r_{41}$; $r_{51} = r_{42} \cdot r_{31} - r_{33} \cdot r_{40}$; $r_{50} = r_{42} \cdot r_{30}$; $t_{52} = -r_{33} \cdot t_{41}$; $t_{51} = -r_{42} - r_{33} \cdot t_{40}$; $t_{50} = r_{42} \cdot r_{23}$; $r_{61} = r_{52} \cdot r_{41} - r_{42} \cdot r_{51}$; $r_{60} = r_{52} \cdot r_{40} - r_{42} \cdot r_{50}$; $t_{62} = -r_{42} \cdot t_{52}$; $t_{61} = r_{52} \cdot t_{41} - r_{42} \cdot t_{51}$; $t_{60} = r_{52} \cdot t_{40} - r_{42} \cdot t_{50}$; $r_{71} = r_{61} \cdot r_{51} - r_{52} \cdot r_{60}$; $r_{70} = r_{61} \cdot r_{50}$; $t_{73} = -r_{52} \cdot t_{62}$; $t_{72} = r_{61} \cdot t_{52} - r_{52} \cdot t_{61}$; $t_{71} = r_{61} \cdot t_{51} - r_{52} \cdot t_{60}$; $t_{70} = r_{61} \cdot t_{50}$; $r_{80} = r_{71} \cdot r_{60} - r_{61} \cdot r_{70}$; $s_{13} = -r_{61} \cdot t_{73}$; $s_{12} = r_{71} \cdot t_{62} - r_{61} \cdot t_{72}$; $s_{11} = r_{71} \cdot t_{61} - r_{61} \cdot t_{71}$; $s_{10} = r_{71} \cdot t_{60} - r_{61} \cdot t_{70}$; If $r_{80} = 0$ call Cantor's Algorithm; $t_0 = r_{80}^{-1}$; $s_{13} = s_{13} \cdot t_0$; $s_{12} = s_{12} \cdot t_0$; $s_{11} = s_{11} \cdot t_0$; $s_{10} = s_{10} \cdot t_0$; $s_{23} = -s_{13}$; $s_{22} = -s_{13} \cdot a - s_{12} - e \cdot s_{23}$; $s_{21} = -s_{13} \cdot b - s_{12} \cdot a - s_{11} - e \cdot s_{22} - f \cdot s_{23}$; $s_{20} = -s_{13} \cdot c - s_{12} \cdot b - s_{11} \cdot a - s_{10} - e \cdot s_{21} - f \cdot s_{22} - g \cdot s_{23}$;	
2	Compute $u' = u_1 \cdot u_2 / d^2 = u_1 \cdot u_2 = x^8 + u'_7 x^7 + u'_6 x^6 + u'_5 x^5 + u'_4 x^4 + u'_3 x^3 + u'_2 x^2 + u'_1 x + u'_0$ (Karatsuba): $t_{00} = d \cdot h$; $t_{01} = c \cdot g$; $t_{02} = (c + d) \cdot (g + h)$; $t_{10} = b \cdot f$; $t_{11} = a \cdot e$; $t_{12} = (a + b) \cdot (e + f)$; $t_{20} = (b + d) \cdot (h + f)$; $t_{21} = (a + c) \cdot (e + g)$; $t_{22} = (a + b + c + d) \cdot (e + f + g + h)$; $u'_0 = t_{00}$; $u'_1 = t_{02} - t_{01} - t_{00}$; $u'_2 = t_{01} + t_{20} - t_{10} - t_{00}$; $u'_3 = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $u'_4 = d + h + t_{21} - t_{11} - t_{01} + t_{10}$; $u'_5 = c + g + t_{12} - t_{10} - t_{11}$; $u'_6 = b + f + t_{11}$; $u'_7 = a + e$;	$9M$
3	Compute $v' = s_1 u_1 v_2 + s_2 u_2 v_1 \bmod u'$ (Karatsuba): $t_{00} = d \cdot s_{10}$; $t_{01} = c \cdot s_{11}$; $t_{02} = (c+d) \cdot (s_{10} + s_{11})$; $t_{10} = b \cdot s_{12}$; $t_{11} = a \cdot s_{13}$; $t_{12} = (a+b) \cdot (s_{12} + s_{13})$; $t_{20} = (b+d) \cdot (s_{12} + s_{10})$; $t_{21} = (a + c) \cdot (s_{11} + s_{13})$; $t_{22} = (a + b + c + d) \cdot (s_{10} + s_{11} + s_{12} + s_{13})$; $su_{10} = t_{00}$; $su_{11} = t_{02} - t_{01} - t_{00}$; $su_{12} = t_{01} + t_{20} - t_{10} - t_{00}$; $su_{13} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $su_{14} = s_{10} + t_{21} - t_{11} - t_{01} + t_{10}$; $su_{15} = s_{11} + t_{12} - t_{10} - t_{11}$; $su_{16} = s_{12} + t_{11}$; $su_{17} = s_{13}$;	$81M$

Table A.13: (continued)

Step	Procedure	Cost
	$t_{00} = p \cdot su_{10}$; $t_{01} = o \cdot su_{11}$; $t_{02} = (su_{11} + su_{10}) \cdot (o + p)$; $t_{10} = n \cdot su_{12}$; $t_{11} = m \cdot su_{13}$; $t_{12} = (su_{13} + su_{12}) \cdot (m + n)$; $t_{20} = (su_{12} + su_{10}) \cdot (n + p)$; $t_{21} = (su_{13} + su_{11}) \cdot (m + o)$; $t_{22} = (su_{10} + su_{11} + su_{12} + su_{13}) \cdot (m + n + o + p)$; $d_{100} = t_{00}$; $d_{101} = t_{02} - t_{00} - t_{01}$; $d_{102} = t_{01} + t_{20} - t_{10} - t_{00}$; $d_{103} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $d_{104} = t_{21} - t_{11} - t_{01} + t_{10}$; $d_{105} = t_{12} - t_{10} - t_{11}$; $d_{106} = t_{11}$; $t_{00} = p \cdot (su_{10} + su_{14})$; $t_{01} = o \cdot (su_{11} + su_{15})$; $t_{02} = (su_{10} + su_{14} + su_{11} + su_{15}) \cdot (p + o)$; $t_{10} = n \cdot (su_{12} + su_{16})$; $t_{11} = m \cdot (su_{13} + su_{17})$; $t_{12} = (su_{13} + su_{17} + su_{12} + su_{16}) \cdot (m + n)$; $t_{20} = (su_{12} + su_{16} + su_{10} + su_{14}) \cdot (n + p)$; $t_{21} = (su_{13} + su_{17} + su_{11} + su_{15}) \cdot (m + o)$; $t_{22} = (su_{10} + su_{11} + su_{12} + su_{13} + su_{14} + su_{15} + su_{16} + su_{17}) \cdot (m + n + o + p)$; $d_{120} = t_{00}$; $d_{121} = t_{02} - t_{00} - t_{01}$; $d_{122} = t_{01} + t_{20} - t_{10} - t_{00}$; $d_{123} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $d_{124} = t_{21} - t_{11} - t_{01} + t_{10}$; $d_{125} = t_{12} - t_{10} - t_{11}$; $d_{126} = t_{11}$; $suv_{10} = d_{100}$; $suv_{11} = d_{101}$; $suv_{12} = d_{102}$; $suv_{13} = d_{103}$; $suv_{14} = d_{104} + d_{120} - d_{100}$; $suv_{15} = d_{105} + d_{121} - d_{101}$; $suv_{16} = d_{106} + d_{122} - d_{102}$; $suv_{17} = d_{123} - d_{103}$; $suv_{18} = d_{124} - d_{104}$; $suv_{19} = d_{125} - d_{105}$; $suv_{1a} = d_{126} - d_{106}$; $t_{00} = h \cdot s_{20}$; $t_{01} = g \cdot s_{21}$; $t_{02} = (g+h) \cdot (s_{20}+s_{21})$; $t_{10} = f \cdot s_{22}$; $t_{11} = e \cdot s_{23}$; $t_{12} = (e+f) \cdot (s_{22}+s_{23})$; $t_{20} = (f+h) \cdot (s_{22}+s_{20})$; $t_{21} = (e+g) \cdot (s_{21}+s_{23})$; $t_{22} = (e+f+g+h) \cdot (s_{20}+s_{21}+s_{22}+s_{23})$; $su_{20} = t_{00}$; $su_{21} = t_{02} - t_{01} - t_{00}$; $su_{22} = t_{01} + t_{20} - t_{10} - t_{00}$; $su_{23} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $su_{24} = s_{20} + t_{21} - t_{11} - t_{01} + t_{10}$; $su_{25} = s_{21} + t_{12} - t_{10} - t_{11}$; $su_{26} = s_{22} + t_{11}$; $su_{27} = s_{23}$; $t_{00} = l \cdot su_{20}$; $t_{01} = k \cdot su_{21}$; $t_{02} = (su_{21} + su_{20}) \cdot (k + l)$; $t_{10} = j \cdot su_{22}$; $t_{11} = i \cdot su_{23}$; $t_{12} = (su_{23} + su_{22}) \cdot (i + j)$; $t_{20} = (su_{22} + su_{20}) \cdot (j + l)$; $t_{21} = (su_{23} + su_{21}) \cdot (i + k)$; $t_{22} = (su_{20} + su_{21} + su_{22} + su_{23}) \cdot (i + j + k + l)$; $d_{100} = t_{00}$; $d_{101} = t_{02} - t_{00} - t_{01}$; $d_{102} = t_{01} + t_{20} - t_{10} - t_{00}$; $d_{103} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $d_{104} = t_{21} - t_{11} - t_{01} + t_{10}$; $d_{105} = t_{12} - t_{10} - t_{11}$; $d_{106} = t_{11}$;	

Table A.13: (continued)

Step	Procedure	Cost
	$t_{00} = l \cdot (su_{20} + su_{24})$; $t_{01} = k \cdot (su_{21} + su_{25})$; $t_{02} = (su_{20} + su_{24} + su_{21} + su_{25}) \cdot (k + l)$; $t_{10} = j \cdot (su_{22} + su_{26})$; $t_{11} = i \cdot (su_{23} + su_{27})$; $t_{12} = (su_{23} + su_{27} + su_{22} + su_{26}) \cdot (i + j)$; $t_{20} = (su_{22} + su_{26} + su_{20} + su_{24}) \cdot (j + l)$; $t_{21} = (su_{23} + su_{27} + su_{21} + su_{25}) \cdot (i + k)$; $t_{22} = (su_{20} + su_{21} + su_{22} + su_{23} + su_{24} + su_{25} + su_{26} + su_{27}) \cdot (i + j + k + l)$; $d_{120} = t_{00}$; $d_{121} = t_{02} - t_{00} - t_{01}$; $d_{122} = t_{01} + t_{20} - t_{10} - t_{00}$; $d_{123} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $d_{124} = t_{21} - t_{11} - t_{01} + t_{10}$; $d_{125} = t_{12} - t_{10} - t_{11}$; $d_{126} = t_{11}$; $suv_{20} = d_{100}$; $suv_{21} = d_{101}$; $suv_{22} = d_{102}$; $suv_{23} = d_{103}$; $suv_{24} = d_{104} + d_{120} - d_{100}$; $suv_{25} = d_{105} + d_{121} - d_{101}$; $suv_{26} = d_{106} + d_{122} - d_{102}$; $suv_{27} = d_{123} - d_{103}$; $suv_{28} = d_{124} - d_{104}$; $suv_{29} = d_{125} - d_{105}$; $suv_{2a} = d_{126} - d_{106}$; $c_0 = suv_{10} + suv_{20}$; $c_1 = suv_{11} + suv_{21}$; $c_2 = suv_{12} + suv_{22}$; $c_3 = suv_{13} + suv_{23}$; $c_4 = suv_{14} + suv_{24}$; $c_5 = suv_{15} + suv_{25}$; $c_6 = suv_{16} + suv_{26}$; $c_7 = suv_{17} + suv_{27}$; $c_8 = suv_{18} + suv_{28}$; $c_9 = suv_{19} + suv_{29}$; $c_{10} = suv_{1a} + suv_{2a}$; $t_0 = c_{10}$; $t_2 = t_0 \cdot u'_7$; $t_1 = c_9 - t_2$; $v'_8 = c_8 - (t_0 + t_1) \cdot (u'_6 + u'_7) + t_2 + t_1 \cdot u'_6$; $v'_7 = c_7 - t_0 \cdot u'_5 - t_1 \cdot u'_6 - v'_8 \cdot u'_7$; $v'_6 = c_6 - (t_0 + t_1) \cdot (u'_4 + u'_5) + t_0 \cdot u'_5 + t_1 \cdot u'_4 - v'_8 \cdot u'_6$; $v'_5 = c_5 - t_0 \cdot u'_3 - t_1 \cdot u'_4 - v'_8 \cdot u'_5$; $v'_4 = c_4 - (t_0 + t_1) \cdot (u'_2 + u'_3) + t_1 \cdot u'_2 + t_0 \cdot u'_3 - v'_8 \cdot u'_4$; $v'_3 = c_3 - t_0 \cdot u'_1 - u'_1 \cdot u'_2 - v'_8 \cdot u'_3$; $v'_2 = c_2 - (t_0 + t_1) \cdot (u'_0 + u'_1) + t_0 \cdot u'_1 + t_1 \cdot u'_0 - v'_8 \cdot u'_2$; $v'_1 = c_1 - t_1 \cdot u'_0 - v'_8 \cdot u'_1$; $v'_0 = c_0 - v'_8 \cdot u'_0$;	
4	Comp. $u_3 = (f - v'h - v'^2)/u'$ $= x^6 + u_{35}x^5 + u_{34}x^4 + u_{33}x^3 + u_{32}x^2 + u_{31}x + u_{30}$: $u_{36} = -v'^2_7$; $u_{35} = -2 \cdot v'_7 \cdot v'_6 - u_{36} \cdot u'_7$; $u_{34} = -v'^2_6 - 2 \cdot v'_7 \cdot v'_5 - u_{35} \cdot u'_7 - u_{36} \cdot u'_6$; $u_{33} = -v'_7 \cdot h_4 - 2 \cdot v'_7 \cdot v'_4 - 2 \cdot v'_6 \cdot v'_5 - u_{36} \cdot u'_5 - u_{35} \cdot u'_6 - u_{34} \cdot u'_7$; $u_{32} = -v'^2_5 - v'_6 \cdot h_4 - 2 \cdot v'_7 \cdot v'_3 - v'_7 \cdot h_3 - 2 \cdot v'_6 \cdot v'_4 - u_{35} \cdot u'_5 - u_{36} \cdot u'_4 - u_{33} \cdot u'_7 - u_{34} \cdot u'_6$; $u_{31} = -v'_5 \cdot h_4 - 2 \cdot v'_5 \cdot v'_4 - 2 \cdot v'_7 \cdot v'_2 - v'_7 \cdot h_2 + 1 - 2 \cdot v'_6 \cdot v'_3 - v'_6 \cdot h_3 - u_{35} \cdot u'_4 - u_{36} \cdot u'_3 - u_{33} \cdot u'_6 - u_{34} \cdot u'_5 - u_{32} \cdot u'_7$; $u_{30} = -v'_5 \cdot h_3 - v'_7 \cdot h_1 - v'^2_4 - 2 \cdot v'_5 \cdot v'_3 - v'_4 \cdot h_4 - 2 \cdot v'_6 \cdot v'_2 - 2 \cdot v'_7 \cdot v'_1 + f_8 - v'_6 \cdot h_2 - u_{35} \cdot u'_3 - u_{36} \cdot u'_2 - u_{32} \cdot u'_6 - u_{33} \cdot u'_5 - u_{34} \cdot u'_4 - u_{31} \cdot u'_7$; $t0 = u_{36}^{-1}$; $u_{35} = u_{35} \cdot t_0$; $u_{34} = u_{34} \cdot t_0$; $u_{33} = u_{33} \cdot t_0$; $u_{32} = u_{32} \cdot t_0$; $u_{31} = u_{31} \cdot t_0$; $u_{30} = u_{30} \cdot t_0$;	$I + 38M + 3S$
5	Compute $v_3 = -(v' + h) \bmod u_3$ $= v_{35}x^5 + v_{34}x^4 + v_{33}x^3 + v_{32}x^2 + v_{31}x + v_{30}$: $t_0 = -v'_7$; $t_2 = t_0 \cdot u_{35}$; $t_1 = -v'_6 - t_2$; $t_3 = t_1 \cdot u_{34}$; $t_4 = t_1 \cdot u_{32}$; $t_5 = t_1 \cdot u_{30}$; $t_6 = t_0 \cdot u_{33}$; $t_7 = t_0 \cdot u_{31}$;	$9M$

Table A.13: (continued)

Step	Procedure	Cost
	$v_{35} = -v_5' - (t_0+t_1)\cdot(u_{34}+u_{35}) + t_3 + t_2$; $v_{34} = -(v_4'+h_4) - t_6 - t_3$; $v_{33} = -(v_3'+h_3) - (t_0+t_1)\cdot(u_{32}+u_{33}) + t_4 + t_6$; $v_{32} = -(v_2'+h_2) - t_7 - t_4$; $v_{31} = -(v_1'+h_1) - (t_0+t_1)\cdot(u_{30}+u_{31}) + t_5 + t_7$; $v_{30} = -(v_0'+h_0) - t_5$;	
6	Compute $u_4 = (f - v_3 h - v_3^2)/u_3$ $= x^4 + a_3x^3 + b_3x^2 + c_3x + d_3$: $u_{44} = -v_{35}^2$; $a_3 = 1 - 2\cdot v_{35}\cdot v_{34} - v_{35}\cdot h_4 - u_{44}\cdot u_{35}$; $b_3 = f_8 - v_{34}^2 - 2\cdot v_{35}\cdot v_{33} - v_{34}\cdot h_4 - v_{35}\cdot h_3 - a_3\cdot u_{35} - u_{44}\cdot u_{34}$; $c_3 = f_7 - 2\cdot v_{34}\cdot v_{33} - 2\cdot v_{35}\cdot v_{32} - v_{35}\cdot h_2 - v_{34}\cdot h_3 - v_{33}\cdot h_4 - a_3\cdot u_{34} - b_3\cdot u_{35} - u_{44}\cdot u_{33}$; $d_3 = f_6 - 2\cdot v_{35}\cdot v_{31} - 2\cdot v_{34}\cdot v_{32} - v_{33}^2 - v_{32}\cdot h_4 - v_{35}\cdot h_1 - v_{34}\cdot h_2 - v_{33}\cdot h_3 - u_{44}\cdot u_{32} - c3\cdot u_{35} - b_3\cdot u_{34} - a_3\cdot u_{33}$; $t_0 = u_{44}^{-1}$; $a_3 = a_3\cdot t_0$; $b_3 = b_3\cdot t_0$; $c_3 = c_3\cdot t_0$; $d_3 = d_3\cdot t_0$;	$I + 20M + 2S$
7	Compute $v_4 = -(v_3 + h) \bmod u_4 = i_3x^3 + j_3x^2 + k_3x + l_3$: $t_0 = -v_{35}$; $t_2 = t_0\cdot a_3$; $t_1 = -(v_{34}+h_4) - t_2$; $t_3 = t_1\cdot b_3$; $t_4 = t_0\cdot c_3$; $t_5 = t_1\cdot d_3$; $i_3 = -(v_{33}+h_3) - (t_0+t_1)\cdot(b_3+a_3) + t_3 + t_2$; $j_3 = -(v_{32}+h_2) - t_4 - t_3$; $k_3 = -(v_{31}+h_1) - (t_0+t_1)\cdot(d_3+c_3) + t_5 + t_4$; $l_3 = -(v_{30}+h_0) - t_5$;	$6M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_8 = 0$	$3I + 222M + 6S$
	fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_8 = 0$	$3I + 204M + 6S$

Table A.14: Explicit formulae for doubling on a HEC of genus four (Cantor).

Input	A weight four reduced divisor $D_1 = (u, v)$ $u = x^4 + ax^3 + bx^2 + cx + d$; $v = ix^3 + jx^2 + kx + l$ $h = h_4x^4 + h_3x^3 + h_2x^2 + h_1x + h_0$ where $h_i \in \{0,1\}$; $f = x^9 + f_8x^8 + f_7x^7 + f_6x^6 + f_5x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$ where $f_8 \in \{0,1\}$;	
Output	A weight four reduced divisor $D_2 = (u_2, v_2) = [2]D_1$ $u_2 = x^4 + a_2x^3 + b_2x^2 + c_2x + d_2$; $v_2 = i_2x^3 + j_2x^2 + k_2x + l_2$	
Step	Procedure	Cost
1	$gcd(u_1, h + 2v_1) = d = s_1\cdot u_1 + s_3\cdot(h + 2v_1)$ (EEA): $r_{13} = h_3 + 2\cdot i - h_4\cdot a$; $r_{12} = h_2 + 2\cdot j - h_4\cdot b$; $r_{11} = h_1 + 2\cdot k - h_4\cdot c$; $r_{10} = h_0 + 2\cdot l - h_4\cdot d$; $r_{23} = r_{13}\cdot a - r_{12}$; $r_{22} = r_{13}\cdot b - r_{11}$; $r_{21} = r_{13}\cdot c - r_{10}$; $r_{20} = r_{13}\cdot d$; $t_{21} = -1$; $t_{20} = 0$; $r_{32} = r_{23}\cdot r_{12} - r_{13}\cdot r_{22}$; $r_{31} = r_{23}\cdot r_{11} - r_{13}\cdot r_{21}$;	$I + 60M$

Table A.14: (continued)

Step	Procedure	Cost
	$r_{30} = r_{23} \cdot r_{10} - r_{13} \cdot r_{20}$; $t_{31} = r_{13}$; $t_{30} = r_{23}$; $r_{42} = r_{32} \cdot r_{22} - r_{23} \cdot r_{31}$; $r_{41} = r_{32} \cdot r_{21} - r_{23} \cdot r_{30}$; $r_{40} = r_{32} \cdot r_{20}$; $t_{42} = -r_{23} \cdot t_{31}$; $t_{41} = r_{32} \cdot t_{21} - r_{23} \cdot t_{30}$; $t_{40} = 0$; $r_{51} = r_{42} \cdot r_{31} - r_{32} \cdot r_{41}$; $r_{50} = r_{42} \cdot r_{30} - r_{32} \cdot r_{40}$; $t_{52} = -r_{32} \cdot t_{42}$; $t_{51} = r_{42} \cdot t_{31} - r_{32} \cdot t_{41}$; $t_{50} = r_{42} \cdot t_{30}$; $r_{61} = r_{51} \cdot r_{41} - r_{42} \cdot r_{50}$; $r_{60} = r_{51} \cdot r_{40}$; $t_{63} = -r_{42} \cdot t_{52}$; $t_{62} = r_{51} \cdot t_{42} - r_{42} \cdot t_{51}$; $t_{61} = r_{51} \cdot t_{41} - r_{42} \cdot t_{50}$; $t_{60} = r_{51} \cdot t_{40}$; $r_{80} = r_{61} \cdot r_{50} - r_{51} \cdot r_{60}$; $s_{33} = -r_{51} \cdot t_{63}$; $s_{32} = r_{61} \cdot t_{52} - r_{51} \cdot t_{62}$; $s_{31} = r_{61} \cdot t_{51} - r_{51} \cdot t_{61}$; $s_{30} = r_{61} \cdot t_{50} - r_{51} \cdot t_{60}$; If $r_{80} = 0$ call Cantor's Algorithm; $t_0 = r_{80}^{-1}$; $s_{33} = s_{33} \cdot t_0$; $s_{32} = s_{32} \cdot t_0$; $s_{31} = s_{31} \cdot t_0$; $s_{30} = s_{30} \cdot t_0$; $s_{13} = -s_{33} \cdot h_4$; $s_{12} = -2 \cdot s_{33} \cdot i - s_{33} \cdot h_3 - s_{32} \cdot h_4 - a \cdot s_{13}$; $s_{11} = -s_{31} \cdot h_4 - 2 \cdot s_{33} \cdot j - s_{33} \cdot h_2 - 2 \cdot s_{32} \cdot i - s_{32} \cdot h_3 - a \cdot s_{12} - b \cdot s_{13}$; $s_{10} = -s_{32} \cdot h_2 - 2 \cdot s_{31} \cdot i - s_{31} \cdot h_3 - 2 \cdot s_{33} \cdot k - s_{33} \cdot h_1 - 2 \cdot s_{32} \cdot j - s_{30} \cdot h_4 - a \cdot s_{11} - b \cdot s_{12} - c \cdot s_{13}$;	
2	$\underline{u_1 = u^2 = x^8 + u_{17}x^7 + u_{16}x^6 + u_{15}x^5 + u_{14}x^4}$ $\underline{+u_{13}x^3 + u_{12}x^2 + u_{11}x + u_{10}:}$ $t_1 = a^2$; $t_2 = b^2$; $t_3 = c^2$; $t_4 = d^2$; $t_5 = (c+d)^2$; $t_6 = (a+b)^2$; $t_7 = (a+c)^2$; $t_8 = (b+d)^2$; $t_9 = (a+b+c+d)^2$; $u_{10} = t_4$; $u_{11} = t_5 - t_3 - t_4$; $u_{12} = t_3 + t_8 - t_2 - t_4$; $u_{13} = t_9 - t_7 - t_8 - t_6 + t_1 + t_2 - t_5 + t_3 + t_4$; $u_{14} = 2 \cdot d + t_7 - t_1 - t_3 + t_2$; $u_{15} = 2 \cdot c + t_6 - t_1 - t_2$; $u_{16} = 2 \cdot b + t_1$; $u_{17} = 2 \cdot a$;	$9S$
3	$\underline{v_1 = s_1 \cdot u \cdot v + s_3 \cdot (v^2 + f) \bmod u_1 = v_{17}x^7 + v_{16}x^6}$ $\underline{+v_{15}x^5 + v_{14}x^4 + v_{13}x^3 + v_{12}x^2 + v_{11}x + v_{10}:}$ $t_{00} = l \cdot s_{10}$; $t_{01} = k \cdot s_{11}$; $t_{02} = (s_{11} + s_{10}) \cdot (k+l)$; $t_{10} = j \cdot s_{12}$; $t_{11} = i \cdot s_{13}$; $t_{12} = (s_{13} + s_{12}) \cdot (i+j)$; $t_{20} = (s_{10} + s_{12}) \cdot (j+l)$; $t_{21} = (s_{11} + s_{13}) \cdot (i+k)$; $t_{22} = (s_{13} + s_{12} + s_{11} + s_{10}) \cdot (i+j+k+l)$; $sv_0 = t_{00}$; $sv_1 = t_{02} - t_{00} - t_{01}$; $sv_2 = t_{01} + t_{20} - t_{10} - t_{00}$; $sv_3 = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $sv_2 = t_{21} - t_{11} - t_{01} + t_{10}$; $sv_5 = t_{12} - t_{10} - t_{11}$; $sv_6 = t_{11}$;	$73M + 10S$

Table A.14: (continued)

Step	Procedure	Cost
	$t_{00} = d \cdot sv_0$; $t_{01} = c \cdot sv_1$; $t_{02} = (sv_1 + sv_0) \cdot (c + d)$; $t_{10} = b \cdot sv_2$; $t_{11} = a \cdot sv_3$; $t_{12} = (sv_3 + sv_2) \cdot (a + b)$; $t_{20} = (sv_2 + sv_0) \cdot (b + d)$; $t_{21} = (sv_3 + sv_1) \cdot (a + c)$; $t_{22} = (sv_3 + sv_1 + sv_2 + sv_0) \cdot (a + b + c + d)$; $d_{100} = t_{00}$; $d_{101} = t_{02} - t_{00} - t_{01}$; $d_{102} = t_{01} + t_{20} - t_{10} - t_{00}$; $d_{103} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $d_{104} = t_{21} - t_{11} - t_{01} + t_{10}$; $d_{105} = t_{12} - t_{10} - t_{11}$; $d_{106} = t_{11}$; $t_{00} = d \cdot (sv_0 + sv_4)$; $t_{01} = c \cdot (sv_1 + sv_5)$; $t_{02} = (sv_1 + sv_5 + sv_0 + sv_4) \cdot (c + d)$; $t_{10} = b \cdot (sv_2 + sv_6)$; $t_{11} = a \cdot sv_3$; $t_{12} = (sv_3 + sv_2 + sv_6) \cdot (a + b)$; $t_{20} = (sv_2 + sv_6 + sv_0 + sv_4) \cdot (b + d)$; $t_{21} = (sv_3 + sv_1 + sv_5) \cdot (a + c)$; $t_{22} = (sv_3 + sv_2 + sv_6 + sv_1 + sv_5 + sv_0 + sv_4) \cdot (a + b + c + d)$; $d_{120} = t_{00}$; $d_{121} = t_{02} - t_{00} - t_{01}$; $d_{122} = t_{01} + t_{20} - t_{10} - t_{00}$; $d_{123} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $d_{124} = t_{21} - t_{11} - t_{01} + t_{10}$; $d_{125} = t_{12} - t_{10} - t_{11}$; $d_{126} = t_{11}$; $suv_0 = d_{100}$; $suv_1 = d_{101}$; $suv_2 = d_{102}$; $suv_3 = d_{103}$; $suv_4 = d_{104} + d_{120} - d_{100} + sv_0$; $suv_5 = d_{105} + d_{121} - d_{101} + sv_1$; $suv_6 = d_{106} + d_{122} - d_{102} + sv_2$; $suv_7 = d_{123} - d_{103} + sv_3$; $suv_8 = d_{124} - d_{104} + sv_4$; $suv_9 = d_{125} - d_{105} + sv_5$; $suv_{10} = d_{126} - d_{106} + sv_6$; $t_0 = i^2$; $t_1 = j^2$; $t_2 = k^2$; $t_3 = l^2$; $t_4 = (i+k)^2$; $t_5 = (j+l)^2$; $t_{00} = t_3$; $t_{01} = (k + l)^2 - t_2 - t_3$; $t_{02} = t_2$; $t_{10} = t_1$; $t_{11} = (i+j)^2 - t_0 - t_1$; $t_{12} = t_0$; $t_{20} = t_5$; $t_{21} = (i+j+k+l)^2 - t_4 - t_5$; $t_{22} = t_4$; $vsq_0 = t_{00}$; $vsq_1 = t_{01}$; $vsq_2 = t_{02} + t_{20} - t_{10} - t_{00}$; $vsq_3 = t_{21} - t_{11} - t_{01}$; $vsq_4 = t_{10} + t_{22} - t_{12} - t_{02}$; $vsq_5 = t_{11}$; $vsq_6 = t_{12}$; $t_0 = f_0 + vsq_0$; $t_1 = f_1 + vsq_1$; $t_2 = f_2 + vsq_2$; $t_3 = f_3 + vsq_3$; $t_4 = f_4 + vsq_4$; $t_5 = f_5 + vsq_5$; $t_6 = f_6 + vsq_6$; $vsf_0 = t_0 - f_8 \cdot u_{10} + u_{17} \cdot u_{10}$; $vsf_1 = t_1 - u_{10} - f_8 \cdot u_{11} + u_{17} \cdot u_{11}$; $vsf_2 = t_2 - u_{11} - f_8 \cdot u_{12} + u_{17} \cdot u_{12}$; $vsf_3 = t_3 - u_{12} - f_8 \cdot u_{13} + u_{17} \cdot u_{13}$; $vsf_4 = t_4 - u_{13} - f_8 \cdot u_{14} + u_{17} \cdot u_{14}$; $vsf_5 = t_5 - u_{14} - f_8 \cdot u_{15} + u_{17} \cdot u_{15}$; $vsf_6 = t_6 - u_{15} - f_8 \cdot u_{16} + u_{17} \cdot u_{16}$; $vsf_7 = f_7 - u_{16} - f_8 \cdot u_{17} + u_{17}^2$; $t_{00} = s_{30} \cdot vsf_0$; $t_{01} = s_{31} \cdot vsf_1$; $t_{02} = (vsf_0 + vsf_1) \cdot (s_{30} + s_{31})$; $t_{10} = s_{32} \cdot vsf_2$; $t_{11} = s_{33} \cdot vsf_3$; $t_{12} = (vsf_2 + vsf_3) \cdot (s_{32} + s_{33})$; $t_{20} = (vsf_0 + vsf_2) \cdot (s_{30} + s_{32})$; $t_{21} = (vsf_1 + vsf_3) \cdot (s_{33} + s_{31})$; $t_{22} = (vsf_0 + vsf_1 + vsf_2 + vsf_3) \cdot (s_{30} + s_{31} + s_{32} + s_{33})$; $d_{100} = t_{00}$; $d_{101} = t_{02} - t_{00} - t_{01}$; $d_{102} = t_{01} + t_{20} - t_{10} - t_{00}$; $d_{103} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $d_{104} = t_{21} - t_{11} - t_{01} + t_{10}$; $d_{105} = t_{12} - t_{10} - t_{11}$; $d_{106} = t_{11}$;	

Table A.14: (continued)

Step	Procedure	Cost
	$t_{00} = s_{30} \cdot (vsf_0 + vsf_4)$; $t_{01} = s_{31} \cdot (vsf_1 + vsf_5)$; $t_{02} = (vsf_1+vsf_5+vsf_0+vsf_4)\cdot(s_{31}+s_{30})$; $t_{10} = s_{32}\cdot(vsf_2+vsf_6)$; $t_{11} = s_{33}\cdot(vsf_3+vsf_7)$; $t_{12} = (s_{32}+s_{33})\cdot(vsf_3+vsf_7+vsf_2+vsf_6)$; $t_{20} = (vsf_0 + vsf_2 + vsf_4 + vsf_6) \cdot (s_{30} + s_{32})$; $t_{21} = (vsf_1+vsf_3+vsf_5+vsf_7)\cdot(s_{33}+s_{31})$; $t_{22} = (vsf_0+vsf_1+vsf_2+vsf_3+vsf_4+vsf_5+vsf_6+vsf_7)\cdot(s_{30}+s_{31}+s_{32}+s_{33})$; $d_{120} = t_{00}$; $d_{121} = t_{02} - t_{00} - t_{01}$; $d_{122} = t_{01} + t_{20} - t_{10} - t_{00}$; $d_{123} = t_{22} - t_{21} - t_{20} - t_{12} + t_{11} + t_{10} - t_{02} + t_{01} + t_{00}$; $d_{124} = t_{21} - t_{11} - t_{01} + t_{10}$; $d_{125} = t_{12} - t_{10} - t_{11}$; $d_{126} = t_{11}$; $svsf_0 = d_{100}$; $svsf_1 = d_{101}$; $svsf_2 = d_{102}$; $svsf_3 = d_{103}$; $svsf_4 = d_{104} + d_{120} - d_{100}$; $svsf_5 = d_{105} + d_{121} - d_{101}$; $svsf_6 = d_{106} + d_{122} - d_{102}$; $svsf_7 = d_{123} - d_{103}$; $svsf_8 = d_{124} - d_{104}$; $svsf_9 = d_{125} - d_{105}$; $svsf_{10} = d_{126} - d_{106}$; $c_0 = suv_0 + svsf_0$; $c_1 = suv_1 + svsf_1$; $c_2 = suv_2 + svsf_2$; $c_3 = suv_3 + svsf_3$; $c_4 = suv_4 + svsf_4$; $c_5 = suv_5 + svsf_5$; $c_6 = suv_6 + svsf_6$; $c_7 = suv_7 + svsf_7$; $c_8 = suv_8 + svsf_8$; $c_9 = suv_9 + svsf_9$; $t_0 = suv_{10} + svsf_{10}$; $t_1 = c_9 - t_0 \cdot u_{17}$; $t_3 = t_1\cdot u_{16}$; $t_4 = t_0\cdot u_{15}$; $t_5 = t_1\cdot u_{14}$; $t_6 = t_0\cdot u_{13}$; $t_7 = t_1\cdot u_{12}$; $t_8 = t_0 \cdot u_{11}$; $t_9 = t_1 \cdot u_{10}$; $t_2 = c_8 - (t_0 + t_1) \cdot (u_{16} + u_{17}) + t_0 \cdot u_{17} + t_3$; $v_{17} = c_7 - t_4 - t_3 - t_2 \cdot u_{17}$; $v_{16} = c_6 - (t_0 + t_1) \cdot (u_{14} + u_{15}) + t_4 + t_5 - t_2 \cdot u_{16}$; $v_{15} = c_5 - t_6 - t_5 - t_2 \cdot u_{15}$; $v_{14} = c_4-(t_0+t_1)\cdot(u_{12}+u_{13})+t_7+t_6-t_2\cdot u_{14}$; $v_{13} = c_3-t_8-t_7-t_2\cdot u_{13}$; $v_{12} = c_2-(t_0+t_1)\cdot(u_{10}+u_{11})+t_8+t_9-t_2\cdot u_{12}$; $v_{11} = c_1 - t_9 - t_2 \cdot u_{11}$; $v_{10} = c_0 - t_2 \cdot u_{10}$;	
4	$\underline{u_3 = (f - v_1h - v_1^2)/u_1 \text{ made monic}}$ $\underline{= x^6 + u_{35}x^5 + u_{34}x^4 + u_{33}x^3 + u_{32}x^2 + u_{31}x + u_{30}:}$ $u_{36} = -v_{17}^2$; $u_{35} = -2 \cdot v_{17} \cdot v_{16} - u_{36} \cdot u_{17}$; $u_{34} = -v_{16}^2 - 2 \cdot v_{17} \cdot v_{15} - u_{35} \cdot u_{17} - u_{36} \cdot u_{16}$; $u_{33} = -v_{17} \cdot h_4 - 2 \cdot v_{17} \cdot v_{14} - 2 \cdot v_{16} \cdot v_{15} - u_{36} \cdot u_{15} - u_{35} \cdot u_{16} - u_{34} \cdot u_{17}$; $u_{32} = -v_{15}^2 - v_{16} \cdot h_4 - 2 \cdot v_{17} \cdot v_{13} - v_{17} \cdot h_3 - 2 \cdot v_{16} \cdot v_{14} - u_{35} \cdot u_{15} - u_{36} \cdot u_{14} - u_{33} \cdot u_{17} - u_{34} \cdot u_{16}$; $u_{31} = -v_{15}\cdot h_4-2\cdot v_{15}\cdot v_{14}-2\cdot v_{17}\cdot v_{12}-v_{17}\cdot h_2+1-2\cdot v_{16}\cdot v_{13}-v_{16}\cdot h_3-u_{35}\cdot u_{14}-u_{36}\cdot u_{13}-u_{33}\cdot u_{16}-u_{34}\cdot u_{15}-u_{32}\cdot u_{17}$; $u_{30} = -v_{15} \cdot h_3 - v_{17} \cdot h_1 - v_{14}^2 - 2 \cdot v_{15} \cdot v_{13} - v_{14} \cdot h_4 - 2 \cdot v_{16} \cdot v_{12} - 2 \cdot v_{17} \cdot v_{11} + f_8 - v_{16} \cdot h_2 - u_{35} \cdot u_{13} - u_{36} \cdot u_{12} - u_{32} \cdot u_{16} - u_{33} \cdot u_{15} - u_{34} \cdot u_{14} - u_{31} \cdot u_{17}$; $t_0 = u_{36}^{-1}$; $u_{36} = 1$; $u_{35} = u_{35} \cdot t_0$; $u_{34} = u_{34} \cdot t_0$; $u_{33} = u_{33} \cdot t_0$; $u_{32} = u_{32} \cdot t_0$; $u_{31} = u_{31} \cdot t_0$; $u_{30} = u_{30} \cdot t_0$;	$39M + 4S + I$

Table A.14: (continued)

Step	Procedure	Cost
5	$\underline{v_3 = -(v_1 + h) \bmod u_3}$ $\underline{= v_{35}x^5 + v_{34}x^4 + v_{33}x^3 + v_{32}x^2 + v_{31}x + v_{30}:}$ $t_0 = -v_{17}$; $t_2 = t_0 \cdot u_{35}$; $t_1 = -v_{16} - t_2$; $t_3 = t_1 \cdot u_{34}$; $t_4 = t_1 \cdot u_{32}$; $t_5 = t_1 \cdot u_{30}$; $t_6 = t_0 \cdot u_{33}$; $t_7 = t_0 \cdot u_{31}$; $v_{35} = -v_{15} - (t_0 + t_1) \cdot (u_{34} + u_{35}) + t_3 + t_2$; $v_{34} = -(v_{14} + h_4) - t_6 - t_3$; $v_{33} = -(v_{13} + h_3) - (t_0 + t_1) \cdot (u_{32} + u_{33}) + t_4 + t_6$; $v_{32} = -(v_{12} + h_2) - t_7 - t_4$; $v_{31} = -(v_{11} + h_1) - (t_0 + t_1) \cdot (u_{30} + u_{31}) + t_5 + t_7$; $v_{30} = -(v_{10} + h_0) - t_5$;	$9M$
6	$\underline{u_4 = (f - v_3h - v_3^2)/u_3 \text{ made monic}}$ $\underline{= x^4 + a_2x^3 + b_2x^2 + c_2x + d_2}$ $u_{44} = -v_{35}^2$; $a_2 = 1 - 2 \cdot v_{35} \cdot v_{34} - v_{35} \cdot h_4 - u_{44} \cdot u_{35}$; $b_2 = f_8 - v_{34}^2 - 2 \cdot v_{35} \cdot v_{33} - v_{34} \cdot h_4 - v_{35} \cdot h_3 - a_2 \cdot u_{35} - u_{44} \cdot u_{34}$; $c_2 = f_7 - 2 \cdot v_{34} \cdot v_{33} - 2 \cdot v_{35} \cdot v_{32} - v_{35} \cdot h_2 - v_{34} \cdot h_3 - v_{33} \cdot h_4 - a_2 \cdot u_{34} - b_2 \cdot u_{35} - u_{44} \cdot u_{33}$; $d_2 = f_6 - 2 \cdot v_{35} \cdot v_{31} - 2 \cdot v_{34} \cdot v_{32} - v_{33}^2 - v_{32} \cdot h_4 - v_{35} \cdot h_1 - v_{34} \cdot h_2 - v_{33} \cdot h_3 - u_{44} \cdot u_{32} - c_2 \cdot u_{35} - b_2 \cdot u_{34} - a_2 \cdot u_{33}$; $t_0 = u_{44}^{-1}$; $a_2 = a_2 \cdot t_0$; $b_2 = b_2 \cdot t_0$; $c_2 = c_2 \cdot t_0$; $d_2 = d_2 \cdot t_0$;	$20M + 3S + I$
7	$\underline{v_4 = -(v_3 + h) \bmod u_4 = i_2x^3 + j_2x^2 + k_2x + l_2:}$ $t_0 = -v_{35}$; $t_2 = t_0 \cdot a_2$; $t_1 = -(v_{34} + h_4) - t_2$; $t_3 = t_1 \cdot b_2$; $t_4 = t_0 \cdot c_2$; $t_5 = t_1 \cdot d_2$; $i_2 = -(v_{33} + h_3) - (t_0 + t_1) \cdot (b_2 + a_2) + t_3 + t_2$; $j_2 = -(v_{32} + h_2) - t_4 - t_3$; $k_2 = -(v_{31} + h_1) - (t_0 + t_1) \cdot (d_2 + c_2) + t_5 + t_4$; $l_2 = -(v_{30} + h_0) - t_5$;	$6M$
Total	fields of arbitrary characteristic, $h_i \in \mathbb{F}_2$, $f_8 = 0$	$3I + 206M + 17S$
	fields of characteristic two, $h_i \in \mathbb{F}_2$, $f_8 = 0$	$3I + 181M + 14S$
	fields of characteristic two, $h(x) = x$, $f_8 = 0$	$2I + 76M + 13S$

Table A.15: Optimized inversionfree explicit formulae for adding a divisor on a HEC of genus two over $\mathbb{F}_{2^n}$.

Input	$[U_{11}, U_{10}, V_{11}, V_{10}, Z_1]$; $[U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$; $h = x$; and $f = x^5 + f_1 x + f_0$;	
Output	$[U_1', U_0', V_1', V_0', Z']$ $= [U_{11}, U_{10}, V_{11}, V_{10}, Z_1] + [U_{21}, U_{20}, V_{21}, V_{20}, Z_2]$	
Step	Procedure	Cost
1	precomputation: $Z = Z_1 Z_2; \tilde{U}_{21} = Z_1 U_{21}; \tilde{U}_{20} = Z_1 U_{20}; \tilde{V}_{21} = Z_1 V_{21}; \tilde{V}_{20} = Z_1 V_{20};$	5M
2	compute resultant r of U_1, U_2: $z_1 = U_{11} Z_2 - \tilde{U}_{21}; z_2 = \tilde{U}_{20} - U_{10} Z_2; z_3 = U_{11} z_1 + z_2 Z_1;$ $r = z_2 z_3 + z_1^2 U_{10};$	6M + 1S
3	compute almost inverse of U_2 modulo U_1: $inv_1 = z_1; inv_0 = z_3;$	
4	compute s: $w_0 = V_{10} Z_2 - \tilde{V}_{20}; w_1 = V_{11} Z_2 - \tilde{V}_{21}; w_2 = inv_0 w_0;$ $w_3 = inv_1 w_1; s_1 = (inv_0 + Z_1 inv_1)(w_0 + w_1) - w_2 - w_3(Z_1 + U_{11}); s_0 = w_2 - U_{10} w_3;$	8M
5	precomnputation: $R = Zr; s_0 = s_0 Z; s_3 = s_1 Z; \tilde{R} = R s_3; S_3 = s_3^2; S = s_0 s_1;$ $\tilde{S} = s_3 s_1; \tilde{\tilde{S}} = s_0 s_3; \tilde{\tilde{R}} = \tilde{R}\tilde{S};$	8M + 1S
6	compute l: $l_2 = \tilde{S}\tilde{U}_{21}; l_0 = S\tilde{U}_{20}; l_1 = (\tilde{S} + S)(\tilde{U}_{21} + \tilde{U}_{20}) - l_2 - l_0; l_2 = l_2 + \tilde{\tilde{S}};$	3M
7	compute U': $U_0' = s_0^2 + s_1^2 z_1 (z_1 + \tilde{U}_{21}) + z_2 \tilde{S} + R(s_3 + r z_1); U_1' = \tilde{S} z_1 + R^2;$	6M + 3S
8	precomputations: $l_2 = l_2 - U_1'; w_0 = U_0' l_2 - S_3 l_0; w_1 = U_1' l_2 + S_3(U_0' - l_1);$	4M
9	adjust $Z' = \tilde{R} S_3; U_1' = \tilde{R} U_1'; U_0' = \tilde{R} U_0';$	3M
10	compute V' $V_0' = w_0 + \tilde{\tilde{R}} \tilde{V}_{20}; V_1' = w_1 + \tilde{\tilde{R}}(\tilde{V}_{21} + Z);$	2M
Total		$45M + 5S$

Table A.16: Optimized inversionfree explicit formulae for doubling a divisor on a HEC of genus two over $\mathbb{F}_{2^n}$.

Input	$[U_1, U_0, V_1, V_0, Z]$; $h = x$; and $f = x^5 + f_1 x + f_0$;	
Output	$[U_1', U_0', V_1', V_0', Z'] = 2[U_1, U_0, V_1, V_0, Z]$	
Step	Procedure	Cost
1	compute resultant and precomutations: $Z_2 = Z^2; w_0 = V_1^2; w_1 = U_1^2; w_3 = U_1 Z;$	1M + 2S
2	compute k: $k_0 = U_1 w_1 + Z(ZV_1 + w_0);$	3M
3	compute $s = kinv \mod u$: $w_4 = k_0 w_3; w_5 = w_1 Z; s_3 = (w_3 + Z)(k_0 + w_1) + w_4 + (1 + U_1)w_5; s_1 = s_3 Z; s_0 = w_4 - Z U_0 w_5;$	7M
4	precomputations: $R = Z_2^2 U_0; \tilde{R} = R s_1; S_1 = s_1^2; S_0 = s_0^2; s_4 = s_3 s_1; s_5 = s_0 s_3; S = s_5 Z; \tilde{\tilde{R}} = \tilde{R} s_4;$	6M + 3S
5	compute l: $l_2 = U_1 s_4; l_0 = U_0 s_5; l_1 = (s_4 + s_5)(U_1 + U_0) + l_2 + l_0;$	3M
6	compute U': $U_0'' = S_0 + R s_3 Z; U_1'' = R^2;$	2M + 1S
7	precomputations: $l_3 = l_2 + S + U_1''; w_6 = U_0'' l_3 + S_1 l_0; w_7 = U_1'' l_3 + S_1(U_0'' + l_1);$	4M
8	adjust $Z' = S_1 \tilde{R}; U_1' = \tilde{R} U_1''; U_0' = \tilde{R} U_0'';$	3M
9	compute V' $V_0' = w_6 + \tilde{\tilde{R}} V_0; V_1' = w_7 + \tilde{\tilde{R}} V_1 + Z';$	2M
Total		$31M + 6S$

Table A.17: Optimized inversionfree explicit formulae for mixed adding a divisor on a HEC of genus two over $\mathbb{F}_{2^n}$.

Input	$[U_{11}, U_{10}, V_{11}, V_{10}, 1]$; $[U_{21}, U_{20}, V_{21}, V_{20}, Z_{02}]$; $h = x$; and $f = x^5 + f_1 x + f_0$;	
Output	$[U_1', U_0', V_1', V_0', Z'] =$ $[U_{11}, U_{10}, V_{11}, V_{10}, 1] + [U_{21}, U_{20}, V_{21}, V_{20}, Z_{02}]$	
Step	Procedure	Cost
1	precomputation: $\tilde{U}_{21} = U_{21}; \tilde{U}_{20} = U_{20}; \tilde{V}_{21} = V_{21}; \tilde{V}_{20} = V_{20};$	-
2	compute resultant r of U_1, U_2: $z_1 = U_{11} Z_2 - \tilde{U}_{21}; z_2 = \tilde{U}_{20} - U_{10} Z_2; z_3 = U_{11} z_1 + z_2; r = z_2 z_3 + z_1^2 U_{10};$	5M + 1S

Table A.17: (continued)

Step	Procedure	Cost
3	compute almost inverse of U_2 modulo U_1: $inv_1 = z_1; inv_0 = z_3;$	
4	compute s: $w_0 = V_{10}Z_2 - \tilde{V}_{20}; w_1 = V_{11}Z_2 - \tilde{V}_{21}; w_2 = inv_0 w_0; w_3 = inv_1 w_1; s_1 = (inv_0 + inv_1)(w_0 + w_1) - w_2 - w_3(1 + U_{11}); s_0 = w_2 - U_{10} w_3;$	7M
5	precomnputation: $R = Z_2 r; s_0 = s_0 Z_2; s_3 = s_1 Z_2; \tilde{R} = R s_3; S_3 = s_3^2; S = s_0 s_1; \tilde{S} = s_3 s_1; \tilde{\tilde{S}} = s_0 s_3; \tilde{\tilde{R}} = \tilde{R}\tilde{S};$	8M + 1S
6	compute l: $l_2 = \tilde{S}\tilde{U}_{21}; l_0 = S\tilde{U}_{20}; l_1 = (\tilde{S} + S)(\tilde{U}_{21} + \tilde{U}_{20}) - l_2 - l_0; l_2 = l_2 + \tilde{\tilde{S}};$	3M
7	compute U': $U'_0 = s_0^2 + s_1^2 z_1 (z_1 + \tilde{U}_{21}) + z_2 \tilde{S} + R(s_3 + r z_1); U'_1 = \tilde{S} z_1 + R^2;$	6M + 3S
8	precomputations: $l_2 = l_2 - U'_1; w_0 = U'_0 l_2 - S_3 l_0; w_1 = U'_1 l_2 + S_3(U'_0 - l_1);$	4M
9	adjust $Z' = \tilde{R} S_3; U'_1 = \tilde{R} U'_1; U'_0 = \tilde{R} U'_0;$	3M
10	compute V' $V'_0 = w_0 + \tilde{\tilde{R}}\tilde{V}_{20}; V'_1 = w_1 + \tilde{\tilde{R}}(\tilde{V}_{21} + Z_2);$	2M
Total		$38M + 5S$

Table A.18: Optimized affine inversionfree explicit formulae for doubling a divisor on a HEC of genus two over $\mathbb{F}_{2^n}$.

Input	$[U_1, U_0, V_1, V_0, 1]$; $h = x$; and $f = x^5 + f_1 x + f_0$;	
Output	$[U'_1, U'_0, V'_1, V'_0, Z'] = 2[U_1, U_0, V_1, V_0, 1]$	
Step	Procedure	Cost
1	compute k: $k_1 = U_1^2; u_3 = U_1 k_1; k_0 = u_3 + V_1 + V_1^2;$	1M + 2S
2	compute $s = kinv \mod u$: $w_4 = k_0 U_1; s_0 = w_4 - U_0 k_1;$	2M
3	precomputations: $R = U_0 k_0; S_1 = k_0^2; S_0 = s_0^2; s_5 = s_0 k_0; \tilde{R} = R S_1;$	3M + 2S
4	compute l: $l_2 = U_1 S_1; l_0 = U_0 s_5; l_1 = (S_1 + s_5)(U_1 + U_0) + l_2 + l_0;$	3M

Table A.18: (continued)

Step	Procedure	Cost
5	compute U': $U'_0 = S_0 + R; U'_1 = U_0^2;$	1S
6	precomputations: $l_3 = l_2 + s_5 + U'_1; w_6 = U'_0 l_3 + S_1 l_0; w_7 = U'_1 l_3 + S_1(U'_0 + l_1);$	4M
7	adjust $Z' = S_1 R; U'_1 = RU'_1; U'_0 = RU'_0;$	3M
8	compute V' $V'_0 = w_6 + \tilde{R}V_0; V'_1 = w_7 + \tilde{R}V_1 + Z';$	2M
Total		$18M + 5S$

Bibliography

[ADH94] L.M. Adlemann, J. DeMarrais, and M.-D. Huang. A Subexponential Algorithm for Discrete Logarithms over the Rational Subgroup of the Jacobians of Large Genus Hyperelliptic Curves over Finite Fields. In L. Adleman and M.-D.Huang, editors, *Algorithmic Number Theory, First International Symposium, ANTS-I*, LNCS 877, pages 28 – 40, Berlin, May 1994. Springer-Verlag.

[ANS99] ANSI X9.62-1999. The Elliptic Curve Digital Signature Algorithm. Technical report, ANSI, 1999.

[ARM00] ARM. ARM Evaluator-7T Board User Guide, 2000. `http://www.arm.com/support/`.

[Ava03] R. M. Avanzi. Countermeasures against Differential Power Analysis for Hyperelliptic Curve Cryptosystems. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003*, LNCS 2779, pages 366 – 381. Springer-Verlag, 2003.

[Ava04] R. M. Avanzi. Aspects of Hyperelliptic Curves over Large Prime Fields in

Software Implementations. In *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2004*, LNCS. Springer-Verlag, 2004.

[BBWP04a] G. Bertoni, L. Breveglieri, T. Wollinger, and C Paar. *Embedded Cryptographic Hardware: Design and Security*, chapter Hyperelliptic Curve Cryptosystem: What is the Best Parallel Hardware Architecture? Nova Science Publishers, NY, USA, 2004. editor Nadia Nedjah.

[BBWP04b] G. Bertoni, L. Breveglieri, T. Wollinger, and C. Paar. Finding Optimum Parallel Coprocessor Design for Genus 2 Hyperelliptic Curve Cryptosystems. In *International Conference on Information Technology: Coding and Computing - ITCC 2004*. IEEE Computer Society, April 2004.

[BCH93] H. Brunner, A. Curiger, and M. Hofstetter. On Computing Multiplicative Inverses in GF(2^m). *IEEE Transactions on Computers*, 42:1010–1015, August 1993.

[BCLW02] N. Boston, T. Clancy, Y. Liow, and J. Webster. Genus Two Hyperelliptic Curve Coprocessor. In B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2002*, LNCS 2523, pages 529–539. Springer-Verlag, 2002. Updated version available at `http://www.cs.umd.edu/~clancy/docs/hec-ches2002.pdf`.

[Ber01] D. J. Bernstein. Multidigit Multiplication for Mathematicians. *Advances in Applied Mathematics*, 2001. `http://cr.yp.to/papers.html`.

[BGL93] I. F. Blake, S. Gao, and R. L. Lambert. Constructive Problems for Irreducible Polynomials Over Finite Fields. In A. Gulliver and N. Secord, editors, *Information Theory and Applications*, LNCS 793, pages 1–23. Springer-Verlag, 1993.

[BSS99] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, London Mathematical Society Lecture Notes Series 265, 1999.

[BW00] G. Borriello and R. Want. Embedded Computation Meets the World Wide Web. *Communications of the ACM*, 43(5):59–66, May 2000.

[Can87] D.G. Cantor. Computing in Jacobian of a Hyperelliptic Curve. In *Mathematics of Computation*, volume 48(177), pages 95 – 101, January 1987.

[CC87] D.V. Chudnovsky and G.V. Chudnovsky. Sequences of Numbers Generated by Addition in Formal Groups and New Primality and Factorization Tests. *Advances in Applied Mathematics*, 7:385–434, 1987.

[CKK02] Y. J. Choi, H. W. Kim, and M. S. Kim. A Design and Implementation of the ECC Crypto Processor. Technical Report, ETRI, DaeJeon, Korea, September 2002.

[Cla02] T. Clancy. Analysis of FPGA-based Hyperelliptic Curve Cryptosystems. Master's thesis, University of Illinois Urbana-Champaign, December 2002.

[Cla03] T. Clancy. FPGA-based Hyperelliptic Curve Cryptosystems. invited paper presented at AMS Central Section Meeting, April 2003.

[CMO98] H. Cohen, A. Miyaji, and T. Ono. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology — ASIACRYPT'98*, LNCS 1514, pages 51–65, Berlin, 1998. Springer-Verlag.

[Coh93] H. Cohen. *A Course in Computational Algebraic Number Theory.* Graduate Texts in Math. 138. Springer-Verlag, Berlin, Germany, 1993. Third corrected printing 1996.

[Cor02] Certicom Corperation. Certicom Announces Elliptic Curve Cryptosystem (ECC) Challenge Winner, 2002. `http://www.certicom.com/about/pr/02/021106_ecc_winner.html`.

[DA99] T. Dierks and C. Allen. *RFC 2246: The TLS Protocol Version 1.0*. Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston, Virginia, USA, January 1999.

[DH76] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[EG02] A. Enge and P. Gaudry. A General Framework for Subexponential Discrete Logarithm Algorithms. *Acta Arith.*, 102:83 – 103, 2002.

[EGCS03] H. Eberle, N. Gura, and S. Chang-Shantz. A Cryptographic Processor for Arbitrary Elliptic Curves over GF(2^m). In *IEEE International Conference on Application Specific Systems Architectures and Processors — ASAP 2003*, 2003.

[EGH00] D. Estrin, R. Govindan, and J. Heidemann. Embedding the Internet. *Communications of the ACM*, 43(5):39–41, May 2000.

[EMY04] G. Elias, A. Miri, and T. H. Yeap. High-Performance, FPGA-Based Hyperelliptic Curve Cryptosystems. In *The Proceeding of the 22nd Biennial Symposium on Communications*, May 2004.

[Eng99a] A. Enge. Computing Discrete Logarithms in High-Genus Hyperelliptic Jacobians in Provably Subexponential Time. `http://www.math.waterloo.ca/Cond0_Dept/CORR/corr99.html`, 1999. Preprint.

[Eng99b] A. Enge. The Extended Euclidean Algorithm on Polynomials, and the

Computational Efficiency of Hyperelliptic Cryptosystems, November 1999. Preprint.

[FKK96] A. O. Freier, P. Karlton, and P. C. Kocher. *The SSL Protocol Version 3.0*. Transport Layer Security Working Group INTERNET-DRAFT, November 1996.

[FR94] G. Frey and H.-G. Rück. A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of Computation*, 62(206):865–874, April 1994.

[Fre98] G. Frey. How to disguise an elliptic curve. Talk at ECC 1998, 1998. `http://cacr.math.uwaterloo.ca/conferences/1998/ecc98/slides.html`.

[FS97] R. Flassenberg and S.Paulus. Sieving in Fuction Fields. `ftp://ftp.informatik.tu-darmstadt.de/pub/TI/TR/TI-97-13.rafla.ps.gz`, 1997. Preprint.

[Ful69] W. Fulton. *Algebraic Curves - An Introduction to Algebraic Geometry*. W. A. Benjamin, Inc., Reading, Massachusetts, 1969.

[Gal01] S.D. Galbraith. Supersingular Curves in Cryptography. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT '03*, LNCS 2248, pages 495 – 517, Berlin, 2001. Springer Verlag.

[Gau00a] P. Gaudry. *Algorithmique des Courbes Hyperelliptiques et Applications à la Cryptologie, PhD Thesis*. PhD thesis, France, 2000.

[Gau00b] P. Gaudry. An Algorithm for Solving the Discrete Log Problem on Hyperelliptic Curves. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, LNCS 1807, pages 19–34, Berlin, Germany, 2000. Springer-Verlag.

[GH00] P. Gaudry and R. Harley. Counting Points on Hyperelliptic Curves over Finite Fields. In W. Bosma, editor, *ANTS IV*, LNCS 1838, pages 297 – 312, Berlin, 2000. Springer Verlag.

[GHS02] P. Gaudry, F. Hess, and N. P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *Journal of Cryptology*, 15(1):19–46, 2002.

[GLV98] R. Gallant, R. Lambert, and S. Vanstone. Improving the Parallelized Pollard Lambda Search on Binary Anomalous Curves. `http://www.certicom.com/chal/download/paper.ps`, 1998.

[GMA+04] M. Goda, K. Matsuo, K. Aoki, J. Chao, and S. Tsujii. Improvements of Addition Algorithm on Gemus 3 Hyperelliptic Curves and their Implementations. In *The 2004 Symposium on Cryptography and Information Security, Japan — SCIS 2004*, January 2004.

[Gol67] S.W. Golomb. *Shift Register Sequences.* Holden-Day, San Francisco, California, USA, 1967.

[Gor98] D. M. Gordon. A Survey of Fast Exponentiation Methods. *Journal of Algorithms*, 27:129–146, 1998.

[Gov03] R. Govindaraian. *Instruction Scheduling.* CRC Press, The Compiler Design Handbook edition, 2003. editor Y. N. Srikant and P. Shankar.

[GP97] J. Guajardo and C. Paar. Efficient Algorithms for Elliptic Curve Cryptosystems. In B. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, LNCS 1294, pages 342–356, Berlin, Germany, August 1997. Springer-Verlag.

[Har00] R. Harley. Fast Arithmetic on Genus Two Curves. `http://cristal.inria.fr/harley/hyper/`, 2000. adding.txt and doubling.c.

[HHM00] D. Hankerson, J. López Hernandez, and A. Menezes. Software Implementation of Elliptic Curve Cryptography Over Binary Fields. In Ç. Koç and C. Paar, editors, *Second International Workshop on Cryptographic Hardware and Embedded Systems — CHES 2000*, LNCS 1965, Berlin, 2000. Springer-Verlag.

[KA98] S. Kent and R. Atkinson. *RFC 2401: Security Architecture for the Internet Protocol.* Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group, Reston, Virginia, USA, November 1998.

[KGM+02] J. Kuroki, M. Gonda, K. Matsuo, J. Chao, and S. Tsujii. Fast Genus Three Hyperelliptic Curve Cryptosystems. In *The 2002 Symposium on Cryptography and Information Security, Japan — SCIS 2002*, January 2002.

[Knu81] D. E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms.* Addison-Wesley, Reading, Massachusetts, USA, 2nd edition, 1981.

[KO63] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Sov. Phys. Dokl. (English translation)*, 7(7):595–596, 1963.

[Kob87] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Kob88] N. Koblitz. A Family of Jacobians Suitable for Discrete Log Cryptosystems. In Shafi Goldwasser, editor, *Advances in Cryptology - Crypto '88*, LNCS 403, pages 94 – 99, Berlin, 1988. Springer-Verlag.

[Kob89] N. Koblitz. Hyperelliptic Cryptosystems. *Journal of Cryptology*, 1(3):129–150, 1989.

[Kob91] N. Koblitz. CM - Curves With Good Cryptographic Properties. In J. Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, LNCS 576, pages 279–287, Berlin, Germany, August 1991. Springer-Verlag.

[Kob98] N. Koblitz. *Algebraic Aspects of Cryptography.* Springer-Verlag, Berlin, Germany, first edition, 1998.

[Kri97] U. Krieger. signature.c. Master's thesis, Mathematik und Informatik, Universität Essen, Fachbereich 6, Essen, Germany, February 1997.

[Lan01] T. Lange. *Efficient Arithmetic on Hyperelliptic Curves.* PhD thesis, Institute for Experimental Mathematics, University of Essen, Essen, Germany, 2001.

[Lan02a] T. Lange. Efficient Arithmetic on Genus 2 Hyperelliptic Curves over Finite Fields via Explicit Formulae. Cryptology ePrint Archive, Report 2002/121, 2002. `http://eprint.iacr.org/`.

[Lan02b] T. Lange. Inversion-Free Arithmetic on Genus 2 Hyperelliptic Curves. Cryptology ePrint Archive, Report 2002/147, 2002. `http://eprint.iacr.org`.

[Lan02c] T. Lange. Weighted Coordinates on Genus 2 Hyperelliptic Curves. Cryptology ePrint Archive, Report 2002/153, 2002. `http://eprint.iacr.org`.

[Lan03] T. Lange. Formulae for Arithmetic on Genus 2 Hyperelliptic Curves. to appear in J. AAECC, September 2003. `http://www.ruhr-uni-bochum.de/itsc/tanja/preprints/expl_sub.pdf`.

[LD98] J. López and R. Dahab. Improved algorithms for Elliptic Curve Arithmetic in $GF(2^n)$. In S. Tavares and H. Meijer, editors, *Selected Areas in Cryptog-*

raphy — SAC 1998, LNCS 1556, pages 201 – 212. Springer-Verlag, August 1998.

[LD00] J. López and R. Dahab. High-Speed Software Multiplication in F_{2^m}. In B. Roy and E. Okamoto, editors, *Progress in Cryptology — Indocrypt 2000*, LNCS 1977, pages 203 – 212, Berlin, 2000. Springer-Verlag.

[LSW83] A. Lempel, G. Seroussi, and S. Winograd. On the Complexity of Multiplication in Finite Fields. *Theoretical Computer Science*, 22:285–296, 1983.

[LV01] A. K. Lenstra and E. R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[MCT01] K. Matsuo, J. Chao, and S. Tsujii. Fast Genus Two Hyperelliptic Curve Cryptosystems. In *ISEC2001-31, IEICE*, 2001.

[MDM+02] Y. Miyamoto, H. Doi, K. Matsuo, J. Chao, and S. Tsuji. A Fast Addition Algorithm of Genus Two Hyperelliptic Curve. In *The 2002 Symposium on Cryptography and Information Security — SCIS 2002, IEICE Japan*, pages 497 – 502, 2002. in Japanese.

[Mil86] V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology — CRYPTO '85*, LNCS 218, pages 417–426, Berlin, Germany, 1986. Springer-Verlag.

[Mot00a] Motorola. MFC5307 User's Manual, August 2000. `http://e-www.motorola.com/collateral/MCF5307BUM.pdf`.

[Mot00b] Motorola. MPC823 User's Manual, October 2000. `http://e-www.motorola.com/brdata/PDFDB/docs/MPC823UM.pdf`.

[MS03] P. K. Mishra and P. Sarkar. Parallelizing Explicit Formula for Arithmetic in the Jacobian of Hyperelliptic Curves. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Advances in Cryptology — Asiacrypt 2003*, LNCS 2894, pages 93 –110. Springer-Verlag, 2003.

[Mum84] D. Mumford. Tata Lectures on Theta II. In *Prog. Math.*, volume 43. Birkhäuser, 1984.

[MvOV97] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, USA, 1997.

[MWZ98] A. Menezes, Y. Wu, and R. Zuccherato. *An Elementary Introduction to Hyperelliptic Curves*. Springer-Verlag, Berlin, Germany, first edition, 1998. In: N. Koblitz, Algebraic Aspects of Cryptography.

[Nag00] K. Nagao. Improving Group Law Algorithms for Jacobians of Hyperelliptic Curves. In W. Bosma, editor, *ANTS IV*, LNCS 1838, pages 439 – 448, Berlin, 2000. Springer Verlag.

[Ngu02] K. Nguyen. Curve based cryptography - the state of the art in smart card environments. In *6rd Workshop on Elliptic Curve Cryptosystems (ECC '02)*, Essen, Germany, September 23–25 2002. Invited Contribution.

[OP00] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, LNCS 1965, pages 41 –56. Springer-Verlag, 2000.

[P1399] IEEE. *IEEE P1363 Standard Specifications for Public Key Cryptography*, November 1999. Last Preliminary Draft.

[Pel02] J. Pelzl. Hyperelliptic Cryptosystems on Embedded Microprocessor. Master's thesis, Department of Electrical Engineering and Information Sciences, Ruhr-Universitaet Bochum, Bochum, Germany, September 2002.

[Pol78] J. M. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32(143):918–924, July 1978.

[PWGP03] J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar. Hyperelliptic Curve Cryptosystems: Closing the Performance Gap to Elliptic Curves. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003*, LNCS 2779, pages 349 – 365. Springer-Verlag, September 2003.

[PWP03] J. Pelzl, T. Wollinger, and C. Paar. Low Cost Security: Explicit Formulae for Genus-4 Hyperelliptic Curves. In M. Matsui and R. Zuccherato, editors, *Tenth Annual Workshop on Selected Areas in Cryptography — SAC 2003*, LNCS 3006, pages 1 – 16. Springer-Verlag, 2003.

[PWP04a] J. Pelzl, T. Wollinger, and C Paar. *Embedded Cryptographic Hardware: Design and Security*, chapter Special Hyperelliptic Curve Cryptosystems of Genus Two: Efficient Arithmetic and Fast Implementation. Nova Science Publishers, NY, USA, 2004. editor Nadia Nedjah.

[PWP04b] J. Pelzl, T. Wollinger, and C. Paar. High Performance Arithmetic for Special Hyperelliptic Curve Cryptosystems of Genus Two. In *International Conference on Information Technology: Coding and Computing - ITCC 2004*. IEEE Computer Society, April 2004.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital

Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Rüc99] H.-G. Rück. On the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of Computation*, 68(226):805–806, 1999.

[Sho01] V. Shoup. NTL: A Libary for Doing Number Theory (version 5.0c), 2001. `http://www.shoup.net/ntl/index.html`.

[SK98] S. Sair and D. Kaeli. A study of loop unrolling for vliwbased dsp processor. In *IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE Computer Society, 1998.

[Sma99] N. Smart. On the Performance of Hyperelliptic Cryptosystems. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, LNCS 1592, pages 165–175. Springer-Verlag, 1999.

[SMCT02] H. Sugizaki, K. Matsuo, J. Chao, and S. Tsujii. An Extension of Harley Addition Algorithm for Hyperelliptic Curves over Finite Fields of Characteristic Two. Technical report, The institue of electronics, information and communication engineers, May 2002.

[SOOS95] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck. Fast key exchange with elliptic curve systems. In D. Coppersmith, editor, *Advances in Cryptology — CRYPTO '95*, LNCS 963, pages 43–56, Berlin, Germany, 1995. Springer-Verlag.

[SP97] L. Song and K. K. Parhi. Low-Energy Digit-Serial/Parallel Finite Field Multipliers. *Journal of VLSI Signal Processing Systems*, 2(22):1–17, 1997.

[Spa94] A. M. Spallek. *Kurven vom Geschlecht 2 und ihre Anwendung in Public-Key-Kryptosystemen*. PhD thesis, Institute for Experimental Mathematics, University of Essen, Essen, Germany, July 1994.

[SS98] Y. Sakai and K. Sakurai. Design of Hyperelliptic Cryptosystems in small Characteristic and a Software Implementation over $\mathbb{F}_{2^n}$. In K. Ohta and D. Pei, editors, *Advances in Cryptology - ASIACRYPT '98*, LNCS 1514, pages 80 – 94, Berlin, 1998. Springer Verlag.

[SS00] Y. Sakai and K. Sakurai. On the Practical Performance of Hyperelliptic Curve Cryptosystems in Software Implementation. In *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, volume E83-A NO.4, pages 692 – 703, April 2000.

[SSI98] Y. Sakai, K. Sakurai, and H. Ishizuka. Secure Hyperelliptic Cryptosystems and their Performance. In H. Imai and Y. Zheng, editors, *Public Key Cryptography: First International Workshop on Practice and Theory in Public Key Cryptography — PKC'98*, LNCS 1431, pages 164 – 181, Berlin, 1998. Springer-Verlag.

[Ste01] A. Stein. Sharp upper bounds for arithmetics in hyperelliptic function fields. *Journal of the Ramanujan Mathematical Society*, 16(2):1 – 86, 2001.

[SZ02] J. Scholten and J. Zhu. Hyperelliptic Curves in Characteristic 2. *International Mathematics Research Notices*, 2002(17):905 – 917, 2002.

[Tak02] M. Takahashi. Improving Harley Algorithms for Jacobians of Genus 2 Hyperelliptic Curves. In *SCIS, IEICE Japan*, 2002. in Japanese.

[Thé03] N. Thériault. Index Calculus Attack for Hyperelliptic Curves of Small Genus. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Advances*

in Cryptology - ASIACRYPT '03, LNCS 2894, pages 79 – 92, Berlin, 2003. Springer Verlag.

[vzG01] J. von zur Gathen. Irreducible Trinomials over Finite Fields. In B. Mourrain, editor, *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation — ISSAC2001*, pages 332–336. ACM Press, 2001.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Campridge University Press, The Edingburgh Building, Cambridge CB2 2RU, UK, 1999.

[vzGN00] J. von zur Gathen and M. Nöcker. Exponentiation in Finite Fields: Theory and Practice. In T. Mora and H. Mattson, editors, *Applied Algebra, Agebraic Algorithms and Error Correcting Codes — AAECC-12*, LNCS 1255, pages 88–113, Berlin, 2000. Springer-Verlag.

[WBV+96] E. De Win, A. Bosselaers, S. Vandenberghe, P. De Gersem, and J. Vandewalle. A fast software implementation for arithmetic operations in $GF(2^n)$. In *Asiacrypt '96*, LNCS 1233, pages 65–76. Springer Lecture Notes in Computer Science, 1996.

[WGP04] T. Wollinger, J. Guajardo, and C Paar. Security on FPGAs: State of the Art Implementations and Attacks. *ACM Transactions in Embedded Computing Systems (TECS)*, 2004. Special Issue on Embedded Systems and Security, to appear.

[Wie86] D. H. Wiedemann. Solving Sparse Linear Equations Over Finite Fields. *IEEE Transactions on Information Theory*, IT-32(1):54–62, January 1986.

[Win77] S. Winograd. Some Bilinear Forms Whose Multiplicative Complexity Depends on the Field of Constants. *Mathematical Systems Theory*, 10:169–180, 1977.

[Wol01] T. Wollinger. Computer Architectures for Cryptosystems Based on Hyperelliptic Curves. Master's thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, May 2001.

[WP02] T. Wollinger and C. Paar. Hardware Architectures Proposed for Cryptosystems Based on Hyperelliptic Curves. In *Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems - ICECS 2002*, volume III, pages 1159 – 1163, September 2002.

[WP03a] A. Weimerskirch and C. Paar. Generalizations of the Karatsuba Algorithm for Polynomail Multiplication. Technical report, Ruhr-University Bochum, Germany, 2003. Available at `http://www.crypto.rub.de/Publikationen/texte/kaweb.pdf`.

[WP03b] T. Wollinger and C. Paar. How Secure Are FPGAs in Cryptographic Applications? In P.Y.K. Cheung, G.A. Constantinides, and J.T. de Sousa, editors, *13th International Conference on Field Programmable Logic and Applications — FPL 2003*, LNCS 2778, pages 91 – 100. Springer-Verlag, September 2003.

[WP04] T. Wollinger and C Paar. *New Algorithms, Architectures, and Applications for Reconfigurable Computing*, chapter Security Aspects of FPGAs in Cryptographic Applications. Kluwer, 2004. editor W. Rosenstiel and P. Lysaght.

[WPW+04] T. Wollinger, J. Pelzl, V. Wittelsberger, C Paar, G. Saldamli, and Ç.

K. Koç. Elliptic & Hyperelliptic Curves on Embedded μP. *ACM Transactions in Embedded Computing Systems (TECS)*, 2004. Special Issue on Embedded Systems and Security, to appear.

[ZB68] N. Zierler and J. Brillhart. On Primitive Trinomials (mod2). *Information and Control*, 13:541–554, 1968.

[ZB69] N. Zierler and J. Brillhart. On Primitive Trinomials (mod2), II. *Information and Control*, 14:566–569, 1969.

[Zie70] N. Zierler. On $x^n + x + 1$ over $GF(2)$. *Information and Control*, 16:67–69, 1970.

Curriculum Vitae

Personal Data

Born on December 5th, 1971 in Wasserlos/Alzenau, Germany.

Secondary Education

1999 – 2001	Worcester Polytechnic Institute, Worcester, MA, USA. Degree: M. S. in Electrical and Computer Engineering.
1995 – 1999	Fachhochschule Dieburg, Dieburg, Germany. (University of Applied Technology) Degree: Diplom Ingenieur (FH) in Electrical Engineering (B.S. in Electrical Engineering).
1992 – 1993	Ludwig-Geisler Schule, Hanau, Germany. (Grammer School with technical branch) Degree: Fachabitur (a-level).
1988 – 1991	Deutsche Telekom, Frankfurt, Germany. Apprenticeship, Vocational College.

Profesional Experience

08.2001 – present	Researcher, Ruhr-University Bochum, Bochum, Germany.
08.1999 – 05.2001	Research Assistant, Worcester Polytechnic Institute, Worcester, MA, USA.
09.1998 – 06.1999	Security Engineer, Security Networks AG, Frankfurt, Germany.
10.1997 – 02.1998	Electronic Engineer, National Avionics Ltd, Dublin, Ireland.
08.1991 – 07.1992	Telecommunications Technician, Deutsche Telekom, Aschaffenburg, Germany.

Publications

Journals

- T. Wollinger, J. Guajardo, C. Paar, "Security on FPGAs: State of the Art Implementations and Attacks", *ACM Transactions on Embedded Computing Systems — Special Issue on Embedded Systems Security*, 2004.

- T. Wollinger, J. Pelzl, V. Wittelsberger, C. Paar, G. Saldamli, and C. Koc, "Elliptic & Hyperelliptic Curves on Embedded uP", *ACM Transactions on Embedded Computing Systems — Special Issue on Embedded Systems Security*, 2004.

Book Chapters

- T. Wollinger and C. Paar, "Security aspects of FPGAs in cryptographic applications", *New Algorithms, Architectures, and Applications for Reconfigurable Computing*, editors W. Rosenstiel and P. Lysaght, Kluwer, 2004.

- J. Pelzl, T. Wollinger, and C. Paar, "Special Hyperelliptic Curve Cryptosystems of Genus Two: Efficient Arithmetic and Fast Implementation", *Embedded Cryptographic Hardware: Design and Security*, editor Nadia Nedjah, Nova Science Publishers, 2004.

- G. Bertoni, L Breveglieri, T. Wollinger, and C. Paar, "Hyperelliptic Curve Cryptosystem: What is the Best Parallel Hardware Architecture?", *Embedded Cryptographic Hardware: Design and Security*, editor Nadia Nedjah, Nova Science Publishers, 2004.

Conferences

- E. Barteska, J. Pelzl, C. Paar, V. Wittelsberger, and T. Wollinger, "Case Study: Compiler Comparison for an Embedded Cryptographical Application", *The 2004 International Conference on Embedded Systems and Applications - ESA*, June 21-24, 2004, Las Vegas, USA

- G. Bertoni, L. Breveglieri, T. Wollinger, and C. Paar, "Finding Optimum Parallel Coprocessor Design for Genus 2 Hyperelliptic Curve Cryptosystems", *International Conference on Information Technology: Coding and Computing - ITCC*, April 5 - 7, 2004, Las Vegas, USA.

- J. Pelzl, T. Wollinger, and C. Paar, "High Performance Arithmetic for Hyperelliptic Curve Cryptosystems of Genus Two", *International Conference on Information Technology: Coding and Computing - ITCC*, April 5 - 7, 2004, Las Vegas, USA.

- C. Paar, J. Pelzl, K. Schramm, A. Weimerskirch, and T. Wollinger, "Eingebettete Sicherheit: State-of-the-art und zukünftige Entwicklungen" ("Embedded Security: State-of-the-art and future developments"), *DACH Security*, March 30 - 31, 2004, Basel, Swiss.

- T. Wollinger, "Publik-Key Kryptographie in eingebetteten Systemen: Eine Einführung" ("Public-key Cryptography in Embedded Systems: An overview") (Invited Paper), *SEI Herbsttagung*, September 24, 2003, Bochum, Germany.

- C. Paar and T. Wollinger, "Eingebettet Sicherheit und Kryptographie im Automobil: Eine Einführung" ("Embedded Security and Cryptography in Automobiles: An Introdution"), *Informatik 2003, Workshop: Automotive SW Engineering & Concepts, 33. Jahrestagung der GI*, September/October 29 - 2, 2003, Frankfurt, Germany.

- J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar, "Hyperelliptic Curve Cryptosystems: Closing the Performance Gap to Elliptic Curves", *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2003*, September 7 - 10, 2003, Colon, Germany.

- T. Wollinger and C. Paar, "How Secure Are FPGAs in Cryptographic Applications?", *13th International Conference on Field Programmable Logic and Applications – FPL 2003*, September 1 - 3, 2003, Lisbon, Portugal.

- J. Pelzl, T. Wollinger, and C. Paar, "Low Cost Security: Explicit Formulae for Genus-4 Hyperelliptic Curves", *Tenth Annual Workshop on Selected Areas in Cryptography - SAC 2003*, August 14 - 15, 2003, Ottawa, Canada.

- K. Schramm, T. Wollinger, and C. Paar, "A New Class of Collision Attacks and its Application to DES", *Fast Software Encryption*, February 24 - 26, Lund, Sweden.

- T. Wollinger, J Guajardo, and C. Paar, "Cryptography in Embedded Systems: An Overview" (Invited Paper), *Proceedings of the Embedded World 2003 Conference*, February 18 - 20, 2003, Nürnberg, Germany.

- G. Bertoni, J. Guajardo, S. Kumar, G. Orlando, C. Paar, and T. Wollinger, "Efficient GF(p^m) Arithmetic Architectures for Cryptographic Applications", *Cryptographer's Track of the RSA Conference 2003*, April 13 - 17, 2003, San Francisco, USA.

- J. Guajardo, T. Wollinger, and C. Paar, "Area Efficient GF(p) Architectures for GF(pm) Multipliers", *45th IEEE International Midwest Symposium on Circuits and Systems - MWSCAS 2002*, August 4 - 7, 2002, Tulsa, USA.

- T. Wollinger and C. Paar, "Hardware Architectures proposed for Cryptosystems Based on Hyperelliptic Curves", *9th IEEE International Conference on Electronics, Circuits and Systems - ICECS 2002*, September 15 - 18, 2002, Dubrovnik, Croatia.

- T. Wollinger, M. Wang, J. Guajardo, and C. Paar, "How Well Are High-End DSPs Suited for the AES Algorithms? AES Algorithms on the TMS320C6x DSP", *The Third Advance Encryption Standard (AES3) Candidate Conference*, April 13 - 14, 2000, New York, USA.