2000-04-28

Computation in Optimal Extension Fields

Daniel V. Bailey
Worcester Polytechnic Institute

Follow this and additional works at: https://digitalcommons.wpi.edu/etd-theses

Repository Citation
https://digitalcommons.wpi.edu/etd-theses/418

This thesis is brought to you for free and open access by Digital WPI. It has been accepted for inclusion in Masters Theses (All Theses, All Years) by an authorized administrator of Digital WPI. For more information, please contact wpi-etd@wpi.edu.
Computation in Optimal Extension Fields

by

Daniel V. Bailey

A Thesis
submitted to the Faculty
of the
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree of Master of Science in
Computer Science
by

May, 2000

Approved:

Dr. Christof Paar
Thesis Advisor
ECE and CS Departments

Dr. Gabor Sarkozy
Reader
CS Department

Dr. Micha Hofri
Department Head
CS Department
Abstract

This thesis focuses on a class of Galois field used to achieve fast finite field arithmetic which we call Optimal Extension Fields (OEFs), first introduced in [BP98]. We extend this work by presenting an adaptation of Itoh and Tsujii's algorithm for finite field inversion applied to OEFs. In particular, we use the facts that the action of the Frobenius map in $GF(p^m)$ can be computed with only $m - 1$ subfield multiplications and that inverses in $GF(p)$ may be computed cheaply using known techniques. As a result, we show that one extension field inversion can be computed with a logarithmic number of extension field multiplications. In addition, we provide new variants of the Karatsuba-Ofman algorithm for extension field multiplication which give a performance increase. Further, we provide an OEF construction algorithm together with tables of Type I and Type II OEFs along with statistics on the number of pseudo-Mersenne primes and OEFs. We apply this new work to provide implementation results for elliptic curve cryptosystems on both DEC Alpha workstations and Pentium-class PCs. These results show that OEFs when used with our new inversion and multiplication algorithms provide a substantial performance increase over other reported methods.
Preface

This thesis represents the culmination of a child-like fascination with the world of cryptography. On August 13-14, 1994, I was persuaded by an old friend from high school named Rich Pell to attend a conference called Hackers on Planet Earth. This gathering of hackers, phreakers, Feds, geeks, and other social misfits was held in New York City to mark the tenth anniversary of 2600 Magazine. We were kids fascinated by the vulnerabilities present in the computing and ideological systems which were so quickly changing our world.

At the conference, Bruce Schneier and Matt Blaze gave a panel discussion on cryptography. Years before the explosion of the Internet and electronic commerce, the field of cryptography had not blossomed to its current state of public awareness. They spoke about a new book by Mr. Schneier which had just been published called Applied Cryptography.

It blew me away. It piqued my curiosity to such a degree that I find myself six years later writing my own thesis on the subject. I devoured Applied Cryptography in short order and was inspired to focus my energies on doing research in cryptography. This decision meant a return to full-time study which I’d abandoned in late 1993.

In looking for a university to resume my education, I was persuaded by Amy Bernheisel to cast my gaze toward Massachusetts. Eventually I decided to attend WPI starting in the fall of 1995, where a new professor had just been hired by the name of Christof Paar, whose research interest was cryptography. Since then, Professor Paar has been my advisor through classes, papers, and projects. Thus I got my wish to
explore the fascinating world of cryptography, and I cannot sufficiently thank those who made it possible.

So I dedicate this thesis to Rich Pell, Bruce Schneier, Matt Blaze, Amy Bernheisel, and Christof Paar, without whom none of this would have been necessary.
# Contents

1 Introduction ..... 1

2 Previous Work ..... 4

3 Optimal Extension Fields ..... 6

4 Optimal Extension Field Arithmetic ..... 8
   4.1 Addition and Subtraction ..... 9
   4.2 Multiplication ..... 9
      4.2.1 Extension Field Modular Reduction ..... 10
      4.2.2 Fast Subfield Multiplication ..... 11
      4.2.3 Fast Subfield Modular Reduction ..... 12
   4.3 Inversion Method: The Extended Euclidean Algorithm ..... 16
   4.4 Inversion Method: The Almost Inverse Algorithm ..... 17
   4.5 Inversion Methods: Modified Almost Inverse Algorithm ..... 19

5 Optimal Extension Field Inversion ..... 20
   5.1 Properties of the Frobenius Map on an OEF ..... 21
   5.2 Itoh and Tsujii Inversion for OEFs ..... 23

6 Fast Polynomial Multiplication ..... 27
   6.1 Polynomials of Degree 2 ..... 28
6.2 Polynomials of Degree 5 ................................. 31

7 Fast Scalar Multiplication ................................. 33

8 Implementation Results .................................. 37

9 OEFs in Practice ............................................ 41
  9.1 Key Validation .......................................... 41
  9.2 Conversion from Field Elements to Integers and Octet Strings . . 42

10 OEF Construction and Statistics ......................... 44
  10.1 Type II OEF Construction Algorithm .................. 44
  10.2 Statistics on the Number of OEFs ..................... 47
  10.3 Statistics on the Number of Pseudo-Mersenne Primes .......... 47
  10.4 Tables of Type I and Type II OEFs .................. 48

11 Discussion .................................................. 49
  11.1 Conclusion .............................................. 49

A Tables ....................................................... 51
List of Tables

5.1 Precomputed inversion constants for \( GF((2^{31} - 1)^6) \) with field polynomial \( P(x) = x^6 - 7 \) .......................................................... 23

8.1 OEF arithmetic timings in \( \mu \)sec on DEC Alpha microprocessors for the field \( GF((2^{61} - 1)^3) \) with field polynomial \( P(x) = x^3 - 5 \) ............ 39

8.2 OEF arithmetic timings in \( \mu \)sec on Intel microprocessors for the field \( GF((2^{31} - 1)^6) \) with field polynomial \( P(x) = x^6 - 7 \) ............ 39

A.1 Number of Pseudo-Mersenne Primes, \( 2^n \pm c, \log_2 c \leq \lfloor (n/2) \rfloor \) ............ 52

A.2 Number of Type II OEFs of order between \( 2^{130} \) and \( 2^{256}, 7 \leq n \leq 10 \) . 53

A.3 Number of OEFs of order between \( 2^{130} \) and \( 2^{256}, 11 \leq n \leq 18 \) ............ 53

A.4 Number of Type II OEFs of order between \( 2^{130} \) and \( 2^{256}, 19 \leq n \leq 55 \) 54

A.5 Type I OEFs for \( 7 \leq n \leq 61 \) .......................................................... 55

A.6 Type II OEFs .......................................................... 55
Chapter 1

Introduction

Since their introduction by Victor Miller [Mil86] and Neil Koblitz [Kob87], elliptic curve cryptosystems (ECCs) have been shown to be a secure and computationally efficient method of performing public-key operations. Our focus in the present thesis is the efficient realization of ECCs in software. Our approach focuses on the finite field arithmetic required for ECCs. Finite fields are identified with the notation $\mathbb{GF}(p^m)$, where $p$ is a prime and $m$ is a positive integer. It is well known that finite fields exist for any choice of prime $p$ and integer $m$.

A standard technique in the development of symmetric-key systems has been to design a cipher to be efficient on a particular type of computing platform. For example, the International Data Encryption Algorithm [LM90] and RC5 [Riv95] are designed to use operations that are efficient on desktop-class microprocessors. Similarly, the NIST/ANSI Data Encryption Algorithm has been designed so that hardware realizations are particularly efficient [NIS77] [ANS81].

We propose to take the same approach with public-key system design. ECCs provide the user a great deal of flexibility in the choice of system parameters. Our
Introduction

underlying assumption is that some choices of $p$ and $m$ of a finite field $GF(p^m)$ are a better fit for a particular computer than others. The computer systems we are concerned with in this thesis are the microprocessors found in workstations and desktop PCs.

Most of the previous work in this area focuses on two choices of $p$ and $m$. The case of $p = 2$ is especially attractive for hardware circuit design of finite field multipliers, since the elements of the subfield $GF(2)$ can conveniently be represented by the logical values “0” and “1.” However, $p = 2$ does not offer the same computational advantages in a software implementation, since microprocessors are designed to calculate results in units of data known as words. Traditional software algorithms for multiplication in $GF(2^m)$ have a complexity of $cm^2/w$ steps, where $w$ is the processor’s word length and $c$ is some constant greater than one. For the large values of $m$ required for practical public-key algorithms, multiplication in $GF(2^m)$ can be very slow.

Similarly, prime fields $GF(p)$ also have computational difficulties on standard computers. For example, practical elliptic curve schemes fix $p$ to be greater than $2^{160}$. Multiple machine words are required to represent elements from these fields on general-purpose workstation microprocessors, since typical word sizes are simply not large enough. This representation presents two computational difficulties: carries between words must be accommodated, and reduction modulo $p$ must be performed with operands that span multiple machine words.

Optimal Extension Fields (OEFs) as introduced in [BP98], are finite fields of the form $GF(p^m), p > 2$. OEFs offer considerable computational advantages by selecting $p$ and $m$ specifically to match the underlying hardware used to perform the arithmetic. The previous work in this area has focused on the application of OEFs to RISC workstations, notably the DEC Alpha microprocessor.
This contribution extends the work in [BP98] by providing an efficient inversion algorithm, improved formulas for extension field multiplication, a new algorithm for OEF construction, tables of Type I and Type II OEFs, tables of the number of OEFs for $\lceil \log p \rceil$ up to 57 of the required order for ECCs, as well as statistics on the existence of primes in short intervals. In addition, we review the work on OEFs by others since [BP98] appeared.
Chapter 2

Previous Work

Previous work on optimization of software implementations of finite field arithmetic has often focused on a single cryptographic application, such as designing a fast implementation for one particular finite field. One popular optimization for ECCs involves the use of subfields of characteristic two. A paper due to DeWin et.al. [WBV+96] analyzes the use of $GF((2^n)^m)$, with a focus on $n = 16, m = 11$. This construction yields an extension field with $2^{176}$ elements. The subfield $GF(2^{16})$ has a Cayley table of sufficiently small size to fit in the memory of a workstation. Optimizations for multiplication and inversion in such composite fields of characteristic two are described in [GP97].

Schroeppel et.al. [SOOS95] report an implementation of an elliptic curve analogue of Diffie-Hellman key exchange over $GF(2^{155})$. The arithmetic is based on a polynomial basis representation of the field elements. Another paper by DeWin et.al. [DMPW98] presents a detailed implementation of elliptic curve arithmetic on a desktop PC, with a focus on its application to digital signature schemes using the fields $GF(p)$ with $p$ a 192-bit prime and $GF(2^{191})$. For ECCs over prime fields, their construction uses projective coordinates to eliminate the need for inversion, along with a
balanced ternary representation of the multiplicand. The work in [Bai98] and [BP98] marks a departure from these methods and serves as a starting point for this new research.

A great deal of work has been done in studying aspects of inversion in a finite field especially since inversion is the most costly of the four basic operations. In the case of prime fields, in [Knu81], Knuth demonstrates that the Extended Euclidean Algorithm requires \( 0.843 \log_2(s) + 1.47 \) divisions in the average case, for \( s \) the element we wish to invert. A great number of variants on Euclid’s algorithm have been developed for use in cryptographic applications, as in [WBV96], [LKL98], and [SOOS95].

Itoh and Tsujii present an algorithm in [IT88] for multiplicative inversion in \( GF(q^m) \) based on the idea of reducing extension field inversion to the problem of subfield inversion. Their method is presented in the context of normal bases, where exponentiation to the \( q \)-th power is very efficient.

In [GP97], a version of Itoh and Tsujii’s algorithm for inversion when applied to composite Galois fields \( GF(2^n)^m \) in a polynomial basis is described which serves as the basis for our development of a variant of this method applied to OEFs.

Lee et.al. [LKL98] provide an implementation of OEFs using a choice of \( p \) less than \( 2^{16} \). The authors present a new inversion algorithm they call the Modified Almost Inverse Algorithm (MAIA) which is especially suited for OEFs. Their choice of \( p \) of this size allows for the use of look-up tables for subfield inversion.

Kobayashi et.al. present in [KMKH99] a method of OEF inversion which is based on a direct solution of a set of linear equations. The method is efficient for small values of \( m \).
Chapter 3

Optimal Extension Fields

In the following, we define a class of finite fields, which we call Optimal Extension Fields (OEFs). To simplify matters, we introduce a name for a class of prime numbers:

**Definition 1** Let $c$ be a positive rational integer. A pseudo-Mersenne prime is a prime number of the form $2^n \pm c, \log_2 c \leq [\frac{1}{2}n]$.

We now define an OEF:

**Definition 2** An Optimal Extension Field is a finite field $GF(p^m)$ such that:

1. $p$ is a pseudo-Mersenne prime,

2. An irreducible binomial $P(x) = x^m - \omega$ exists over $GF(p)$.

The following theorem from [LN83] describes the cases when an irreducible binomial exists:

**Theorem 1** Let $m \geq 2$ be an integer and $\omega \in GF(p)^*$. Then the binomial $x^m - \omega$ is irreducible in $GF(p)[x]$ if and only if the following two conditions are satisfied:
(i) each prime factor of $m$ divides the order $e$ of $\omega$ over $GF(p)$, but not $(p - 1)/e$;  
(ii) $p \equiv 1 \mod 4$ if $m \equiv 0 \mod 4$.

An important corollary is given in [Jun93]:

**Corollary 1** Let $\omega$ be a primitive element for $GF(p)$ and let $m$ be a divisor of $p - 1$. Then $x^m - \omega$ is an irreducible polynomial.

We observe that there are two special cases of OEF which yield additional arithmetic advantages, which we call Type I and Type II.

**Definition 3** A Type I OEF has $p = 2^n \pm 1$.

A Type I OEF allows for subfield modular reduction with very low complexity. For ECCs in practice, particularly good choices of $p$ are $2^{31} - 1$ and $2^{61} - 1$.

**Definition 4** A Type II OEF has an irreducible binomial $x^m - 2$.

As will be shown in Section 4.2.1, a Type II OEF allows for a reduction in the complexity of extension field modular reduction since the multiplications by $\omega$ in Theorem 2 can be implemented using shifts instead of explicit multiplications.

The range of possible $m$ for a given $p$ depends on the factorization of $p - 1$ due to Theorem 1 and Corollary 1.
Chapter 4

Optimal Extension Field

Arithmetic

This section describes the previous work on arithmetic in OEFs. Our new method for inversion is treated separately in Chapter 5. In Chapter 6, improved multiplication algorithms are introduced. In Sections 4.2.2 and 4.2.3, the operations of multiplication and modular reduction in the subfield are discussed. Some of the material of this section is described in previous work, and appears here solely for completeness of presentation.

An OEF $GF(p^m)$ is isomorphic to $GF(p)[x]/(P(x))$, where $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i, p_i \in GF(p)$, is a monic irreducible polynomial of degree $m$ over $GF(p)$. In the following, a residue class will be identified with the polynomial of least degree in this class. We consider a standard (or polynomial or canonical) basis representation of a field element $A(x) \in GF(p^m)$:

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0,$$  \hspace{1cm} (4.1)
where $a_i \in GF(p)$. Since we choose $p$ to be less than the processor’s word size, we can represent $A(x)$ with $m$ registers, each containing one $a_i$.

All arithmetic operations are performed modulo the field polynomial. The choice of field polynomial determines the complexity of the modular reduction.

4.1 Addition and Subtraction

Addition and subtraction of two field elements is implemented in a straightforward manner by adding or subtracting the coefficients of their polynomial representation and if necessary, performing a modular reduction by subtracting or adding $p$ once from the intermediate result.

4.2 Multiplication

Field multiplication can be performed in two stages. First, we perform an ordinary polynomial multiplication of two field elements $A(x)$ and $B(x)$, resulting in an intermediate product $C'(x)$ of degree less than or equal to $2m - 2$:

$$C'(x) = A(x) \times B(x) = c_{2m-2}x^{2m-2} + \cdots + c_1'x + c_0'; \quad c'_i \in GF(p). \quad (4.2)$$

The schoolbook method to calculate the coefficients $c'_i, i = 0, 1, \ldots, 2m - 2$, requires $m^2$ multiplications and $(m - 1)^2$ additions in the subfield $GF(p)$.

In Section 4.2.1 we present an efficient method to calculate the residue $C(x) \equiv C'(x) \mod P(x), C(x) \in GF(p^m)$. Section 6 shows ways to reduce the number of coefficient multiplications required.
Squaring can be considered a special case of multiplication. The only difference is that the number of coefficient multiplications can be reduced to \( m(m + 1)/2 \).

In order to perform coefficient multiplications, we must multiply in the subfield. Methods for fast subfield multiplication were noted in [MA85] and [BP98]. For the case of a Type I OEF, we require a single integer multiplication to implement the subfield multiply, whereas with a general OEF we require three.

### 4.2.1 Extension Field Modular Reduction

After performing a multiplication of field elements in a polynomial representation, we obtain the intermediate result \( C'(x) \). In general the degree of \( C'(x) \) will be greater than or equal to \( m \). In this case, we need to perform a modular reduction. The canonical method to carry out this calculation is long polynomial division with remainder by the field polynomial. However, field polynomials of special form allow for computational efficiencies in the modular reduction.

Since monomials \( x^m, m > 1 \) are obviously always reducible, we turn our attention to **irreducible binomials**. An OEF has by definition a field polynomial of the form \( P(x) = x^m - \omega \). The use of an irreducible binomial as a field polynomial yields major computational advantages as will be shown below. Observe that irreducible binomials do not exist over \( GF(2) \). Modular reduction with a binomial can be performed with the following complexity:

**Theorem 2** Given a polynomial \( C'(x) \) over \( GF(p) \) of degree less than or equal to \( 2m - 2 \), \( C'(x) \) can be reduced modulo \( P(x) = x^m - \omega \) requiring at most \( m - 1 \) multiplications by \( \omega \) and \( m - 1 \) additions, where both of these operations are performed in \( GF(p) \).
A general expression for the reduced polynomial is given by:

\[ C(x) \equiv c'_{m-1}x^{m-1} + [\omega c'_{2m-2} + c'_{m-2}]x^{m-2} + \cdots + [\omega c'_m + c'_0] \mod P(x) \quad (4.3) \]

As an optimization, when possible we choose those fields with an irreducible binomial \( x^m - 2 \), allowing us to implement the multiplications as shifts. OEFs that offer this optimization are known as Type II OEFs.

### 4.2.2 Fast Subfield Multiplication

As shown above, fast subfield multiplication is essential for fast multiplication in \( GF(p^m) \). Subfield arithmetic in \( GF(p) \) is implemented with standard modular integer techniques. We recall that multiplication of two elements \( a, b \in GF(p) \) is performed by \( a \times b \equiv c \pmod{p} \). Modern workstation CPUs are optimized to perform integer arithmetic on operands of size up to the width of their registers. An OEF takes advantage of this fact by constructing subfields whose elements may be represented by integers in a single register. For example, on a workstation with 64-bit registers, the largest prime we may represent is \( 2^{64} - 59 \). So we choose a prime \( p \leq 2^{64} - 59 \) as the characteristic of our subfield on this computer. To this end, we recommend the use of Galois fields with subfields as large as possible while still within single-precision limits of our host CPU.

We perform multiplication of two single-word integers and in general obtain a double-word integer result. In order to finish the calculation, we must perform a modular reduction. Obtaining a remainder after division of two integers is a well-studied problem [MA85]. Many methods such as Barrett Reduction exist which offer computational advantages over traditional long division on integers. These methods, however, are still slow when compared to multiplication of single-word integers. Our
choice of $p$ allows a far less complex modular reduction operation.

### 4.2.3 Fast Subfield Modular Reduction

A technique due to Mohan and Adiga shows that fast modular reduction is possible for moduli of the form $2^n \pm c$, where $c$ is a “small” integer [MA85]. Integers of this form allow modular reduction without division. We present a form of such a modular reduction algorithm, adapted from [MA85] and [MvOV97]. This algorithm addresses only the primes of the form $2^n - c$, although trivial change to the allows the use of primes $2^n + c$.

The operators $\ll$ and $\gg$ are taken to mean “left shift” and “right shift” respectively.

**Algorithm 1 Fast Subfield Modular Reduction**

```plaintext
Require: $p = 2^n - c$, $\log_2 c \leq \frac{1}{2}n$, $x < p^2$ is the integer to reduce
Ensure: $r \equiv x \pmod{p}$

$q_0 \leftarrow x \gg n$
$r_0 \leftarrow x - q_02^n$
$r \leftarrow r_0$
$i \leftarrow 0$

while $q_i > 0$ do
  $q_{i+1} \leftarrow q_i c >> n$
  $r_{i+1} \leftarrow q_i c - (q_{i+1} \ll n)$
  $i \leftarrow i + 1$
  $r \leftarrow r + r_i$
end while
while $r \geq p$ do
  $r \leftarrow r - p$
end while
```

To understand the operation of this algorithm, consider the following graphical representation of the situation:
In this example, we begin with two subfield elements $a_i$ and $b_j$, which are of size less than $2^n$, where $2^n - 1$ is the maximum integer we can represent in a single machine register, and $c$ is as in the above algorithm. We form the product $x = a_i b_j$ which is of size less than $2^{2n}$, but in general larger than $2^n$, and by implication larger than $p$. So we use the above algorithm to perform a modular reduction.

We let $r_0$ be the lower $n$ bits of the product $a_i b_j$ and the remaining upper bits $q_0$. We observe that $2^n \pmod{p} \equiv c$ so we may write the following:

\[
q_0, r_0 \leq 2^n - c
\]
\[
x = a_i b_j = 2^n q_0 + r_0
\]
\[
2^n = [1](2^n - c) + [c]
\]
\[
2^n \equiv c \pmod{(p = 2^n - c)}
\]
\[
r = x \equiv cq_0 + r_0 \pmod{p}
\]

This is the situation depicted in the next figure.
In general, this new expression for the residue class is still larger than $2^n$, and by implication larger than $p$. So we repeat this process once more, again rewriting the equations and replacing $2^n$ by $c$:

$$r \equiv cq_1 + r_1 = 2^nq_1 + r_1 \equiv cq_1 + r_1 \pmod{p}$$

Finally we have an expression for the residue of $a_ib_j$ that in general is less than $2^{n+2}$. To complete the reduction, we may need to subtract $p$ from the intermediate result $r$ one or more times. We simply test for this case and we have completed the reduction.

Modular reduction with this algorithm requires only two multiplications by $c$, six shifts by $n$, and six additions and subtractions, when $p$ is a pseudo-Mersenne prime. In practice, this leads to a dramatic performance increase over performing
explicit division with remainder. For example, when \( p = 2^{32} - 5, m = 5, \) and we implement subfield reduction by performing an explicit division with remainder on a 500 MHz DEC Alpha CPU, we require 7.74 \( \mu \text{sec} \) for a multiplication in \( GF(p^m) \). When we perform modular reduction using this algorithm, we require only 1.35 \( \mu \text{sec} \), a fivefold savings.

If \( c = 1 \), this algorithm executes the first while loop only once. To observe this behavior, we simply set \( c = 1 \) in the algorithm and walk through the algorithm:

\[
\begin{align*}
q_0 & \leftarrow x >> n \\
r_0 & \leftarrow x - (q_0 << n) \\
r & \leftarrow r_0 \\
i & \leftarrow 0 \\
q_1 & \leftarrow (q_0 \times 1 >> n) = 0 \\
r_1 & \leftarrow (q_0 - 0) \\
r & \leftarrow r_0 + r_1
\end{align*}
\]

At this point, the algorithm terminates since \( q_1 = 0 \), and we may rewrite this result as:

\[
x \pmod{2^n - 1} \equiv x - ((x >> n) << n) + (x >> n)
\]

In this special case, no multiplications are required for the modular reduction and the entire operation may be performed with two shifts and two adds if the intermediate result is contained in a single word. This is a substantial improvement over the \( c > 1 \) case. An OEF that offers this optimization is known as Type I. In
our implementation as reported in Section 8, we have included \( p = 2^{61} - 1 \) for this reason. Our implementation takes advantage of its special form, making \( p = 2^{61} - 1 \) the best performing choice of \( p \) we consider.

### 4.3 Inversion Method: The Extended Euclidean Algorithm

In [LKL98], the authors propose a method for inversion which we include here for completeness. Our method for inversion is treated in Chapter 5 and is based on an entirely different approach. The material of these sections on Euclidean approaches to extension field inversion is presented in [LKL98].

Traditionally, inversion methods have been based on either Fermat’s Little Theorem or the Extended Euclidean Algorithm (EEA). The Almost Inverse Algorithm (AIA), introduced in [SOOS95], is a variant of the EEA, and is treated in Section 4.4.

The EEA for Polynomials is found in Algorithm 2 [LKL98]. Capital letters denote extension field elements while lowercase letters denote subfield elements and integers. The subscript on a subfield element indicates which coefficient of the polynomial is to be selected.

The algorithm proceeds by adding multiples of the shorter of \( F(x) \) and \( G(x) \) to the longer [LKL98]. This action reduces the degree of the larger polynomial by at least one. With probability \((p - 1)/p\), the degree of the larger polynomial is reduced by two, and so on. Thus for purposes of analysis we may safely assume that each iteration of the algorithm reduces the degree of the larger polynomial by one. The process is repeated until \( F(x) \in GF(p) \) or \( G(x) \in GF(p) \). \( A(x) \) in the worst case will have degree \( m - 1 \), while \( P(x) \) will always have degree \( m \). Thus it is clear that in the
Algorithm 2 Extended Euclidean Algorithm for Polynomials [LKL98]

Require: $A(x) \in GF(p^m)^*$, $P(x)$ is the field polynomial
Ensure: $A(x)B(x) \equiv 1 \in GF(p^m)$

$B(x) \leftarrow 0, C(x) \leftarrow 1, G(x) \leftarrow A(x)$

while $\deg(P(x)) \neq 0$ do
  if $\deg(P(x)) < \deg(G(x))$ then
    exchange $P(x)$ with $G(x)$ and $B(x)$ with $C(x)$
  end if
  $j \leftarrow \deg(P(x)) - \deg(G(x))$
  $\alpha \leftarrow -p_{\deg(P(x))}/g_{\deg(G(x))}$
  $P(x) \leftarrow P(x) + \alpha x^j G(x)$
  $B(x) \leftarrow B(x) + \alpha x^j C(x)$
end while

$B(x) \leftarrow B(x)/p_0$

worst case $2(m - 1)$ iterations are required.

Each iteration of the algorithm requires a subfield inverse. If the subfield $GF(p)$ is small enough, a table of inverses may be precomputed or stored. Otherwise, an algorithm such as the EEA must be run to determine the inverse. Thus this method is useful for those cases where enough storage exists to store a table of inverses. In particular, this is possible for $p \approx 2^{32}$ on a workstation. The remaining methods in this section are also subject to this restriction. Our method in Section 5 overcomes this restriction while maintaining fast performance.

4.4 Inversion Method: The Almost Inverse Algorithm

The Almost Inverse Algorithm [SOOS95] [LKL98] offers a variant on the EEA which is advantageous in particular circumstances. For example, [SOOS95] shows a performance gain when used in the field $GF(2^{155})$. In particular, the EEA finds polynomials
$B(x)$ and $U(x)$ such that $A(x)B(x) + P(x)U(x) \equiv 1 \in GF(p^m)$. In contrast, the AIA modifies the EEA to find $A(x)B(x) + P(x)U(x) \equiv x^k$. The inversion is completed by computing $B(x) \leftarrow B(x)/x^k$. The algorithm is found as Algorithm 3 [LKL98].

<table>
<thead>
<tr>
<th>Algorithm 3 Almost Inverse Algorithm [LKL98]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Require:</strong> $A(x) \in GF(p^m)^*$, $P(x)$ is the field polynomial</td>
</tr>
<tr>
<td><strong>Ensure:</strong> $A(x)B(x) \equiv 1 \in GF(p^m)$</td>
</tr>
<tr>
<td>$k \leftarrow 0$, $B(x) \leftarrow 0$, $C(x) \leftarrow 1$, $G(x) \leftarrow A(x)$</td>
</tr>
<tr>
<td><strong>while</strong> $x</td>
</tr>
<tr>
<td>$P(x) \leftarrow P(x)/x$</td>
</tr>
<tr>
<td>$C(x) \leftarrow C(x)x$</td>
</tr>
<tr>
<td>$k \leftarrow k + 1$</td>
</tr>
<tr>
<td><strong>end while</strong></td>
</tr>
<tr>
<td><strong>while</strong> $\deg(P(x) \neq 0)$ <strong>do</strong></td>
</tr>
<tr>
<td><strong>if</strong> $\deg(P(x)) &lt; \deg(G(x))$ <strong>then</strong></td>
</tr>
<tr>
<td>exchange $P(x)$ with $G(x)$ and $B(x)$ with $C(x)$</td>
</tr>
<tr>
<td><strong>end if</strong></td>
</tr>
<tr>
<td>$\alpha \leftarrow -p_0/g_0$</td>
</tr>
<tr>
<td>$P(x) \leftarrow P(x) + \alpha G(x)$</td>
</tr>
<tr>
<td>$B(x) \leftarrow B(x) + \alpha C(x)$</td>
</tr>
<tr>
<td><strong>end while</strong></td>
</tr>
<tr>
<td>$B(x) \leftarrow B(x)/p_0$</td>
</tr>
<tr>
<td>$B(x) \leftarrow B(x)/x^k$</td>
</tr>
</tbody>
</table>

In $GF(2^m)$ in polynomial basis, the multiplication by $x^j$ is implemented with bitwise shifts. The AIA eliminates the need for these shifts. In addition, the algorithm reduces the degree of $P(x)$ when $\deg(P(x)) = \deg(G(x))$, thus saving iterations. In contrast with the EEA, which saves iterations with probability only $1/p$, this algorithm saves iterations roughly 20% of the time.

However, these advantages are only present in fields of the form $GF(2^m)$. Lee, et.al. present a variant of the AIA which offers comparable advantages to fields of the form $GF(p^m)$ in [LKL98].
4.5 Inversion Methods: Modified Almost Inverse Algorithm

While the EEA works from highest coefficients down to lowest and the AIA works from lowest to highest, the MAIA [LKL98] works on the lowest and highest in the same iteration. However, the total number of operations is almost identical to the EEA. The advantage to this method is that the number of iterations and hence the number of polynomial scalar multiplications are reduced by half.

The algorithm is given as Algorithm 4.

**Algorithm 4** Modified Almost Inverse Algorithm [LKL98]

**Require:** \( A(x) \in GF(p^m)^* \), \( P(x) \) is the field polynomial

**Ensure:** \( A(x)B(x) \equiv 1 \in GF(p^m) \)

\[ k \leftarrow 0, B(x) \leftarrow 0, C(x) \leftarrow 1, G(x) \leftarrow A(x) \]

\[ \text{while } x | P(x) \text{ do} \]

\[ P(x) \leftarrow P(x)/x \]

\[ C(x) \leftarrow C(x)x \]

\[ k \leftarrow k + 1 \]

\[ \text{end while} \]

\[ \text{while } \deg(P(x)) \neq 0 \text{ do} \]

\[ \text{if } \deg(P(x)) < \deg(G(x)) \text{ then} \]

\[ \text{exchange } P(x) \text{ with } G(x) \text{ and } B(x) \text{ with } C(x) \]

\[ \text{end if} \]

\[ j \leftarrow \deg(P(x)) - \deg(G(x)) \]

\[ \beta \leftarrow -p_0/g_0 \]

\[ \text{if } j \neq 0 \text{ then} \]

\[ \alpha \leftarrow -p_{\deg(P(x))}/g_{\deg(G(x))} \]

\[ \text{else} \]

\[ \alpha \leftarrow 0 \]

\[ \text{end if} \]

\[ P(x) \leftarrow P(x) + (\alpha x^j + \beta)G(x) \]

\[ B(x) \leftarrow B(x) + (\alpha x^j + \beta)C(x) \]

\[ \text{end while} \]

\[ B(x) \leftarrow B(x)/p_0 \]

\[ B(x) \leftarrow B(x)/x^k \]
Chapter 5

Optimal Extension Field Inversion

The inversion algorithm for OEFs is based on the observation that the inversion algorithm due to Itoh and Tsujii may be efficiently realized in the context of OEFs. In fact, we show that the inversion method is particularly suited to finite fields in polynomial basis that have a binomial as the field polynomial.

The Itoh and Tsujii Inversion (ITI) [IT88] reduces the problem of extension field inversion to subfield inversion. This reduction relies on a special mapping that is defined for all finite fields. In particular, the norm function maps elements of the extension field to the subfield by raising them to the $(p^m - 1)/(p - 1)$ power [LN83]. In previous reported applications of ITI [GP97], researchers have used look-up tables to perform the subfield inversion. While this approach is efficient, it is also quite limited. For a choice of $p$ less than $2^{16}$, tables easily fit in the storage of modern desktop PCs and workstations. However, a choice of $p$ of approximately $2^{22}$ or $2^{64}$ leads to tables which are simply too large. Our implementation computes the subfield inverse using the Binary Extended Euclidean Algorithm [Nor86]. We show that an efficient implementation of this algorithm is fast enough to make ITI suitable for OEFs.
We outline our version of the ITI here. Our objective is to find an element $A^{-1}(x)$ such that $A(x)A^{-1}(x) \equiv 1 \mod P(x)$.

One method for evaluating the norm of an element is to apply the binary method of exponentiation [Knu81] or one of its improved derivatives [MvOV97]. Such straightforward methods are very costly. Clearly, a faster method would be preferable. Fortunately, we can use the Frobenius map to quickly evaluate the norm function.

5.1 Properties of the Frobenius Map on an OEF

**Definition 5** Let $\alpha \in GF(p^m)$. Then the mapping $\alpha \rightarrow \alpha^p$ is an automorphism known as the Frobenius map.

As noted in [Bas84], the $i$th iterate of the Frobenius map $\alpha \rightarrow \alpha^{pi}$ is also an automorphism. Let us consider the action of an arbitrary iterate $i$ of the Frobenius map on an arbitrary element of $GF(p^m)$: $A(x)^{pi} = \sum a_j x^{jpi}$, for $a_j \in GF(p)$. We know by Fermat’s Little Theorem that $a_j^p \equiv a_j \mod p$. Thus the $a_j$ coefficients are fixed points of Frobenius map iterates and we can write:

$$A^{pi}(x) \equiv a_{m-1}x^{(m-1)pi} + \cdots + a_1 x^{pi} + a_0 \mod P(x) \quad (5.1)$$

Now we need to consider the elements which are not kept fixed by the action of the Frobenius map: $(x^j)^p, 0 < j < m$. We can express these as $x^{jp}$. But this expression is always a polynomial with a single non-zero term due to the following theorem (see also [KMKH99]):

**Theorem 3** Let $P(x)$ be an irreducible polynomial of the form $P(x) = x^m - \omega$ over
Inversion

\[ GF(p), \ e \ \text{an integer, } x \in GF(p)[x]. \ \text{Then:} \]

\[ x^e \equiv \omega^s x^s \mod P(x) \quad (5.2) \]

where \( s \equiv e \mod m \) with \( q = \frac{e-s}{m} \).

**Proof 1** First, we observe that \( x^m \equiv \omega \mod P(x) \). Now,

\[ x^e = x^{qm+s} \quad (5.3) \]

where \( q \) and \( s \) are defined above. Then:

\[ x^e = x^{qm} x^s \equiv \omega^s x^s \mod P(x) \quad (5.4) \]

\( \square \)

We have the following corollary which is of especial interest in our case of applying iterates of the Frobenius map:

**Corollary 2**

\[ (x^i)^{p^i} \equiv \omega^s x^i \mod P(x) \quad (5.5) \]

where \( x^i \in GF(p)[x], \ i \) is an arbitrary positive rational integer, and other variables are defined in Theorem 3.

**Proof 2** Since \( P(x) \) is an irreducible binomial, by Theorem 1, \( m|(p-1) \), which implies \( p = (p-1) + 1 \equiv 1 \mod m \). Thus \( s \equiv fp^i \equiv j \mod m \). \( \square \)

Note that all \( x^{jp^i}, 1 \leq j, i \leq m-1 \) in Equation (5.1) can be precomputed if \( P(x) \) is given. Given the above, to compute \( (a_jx^i)^{p^i} \) we need only a single subfield
Inversion

multiplication. Thus, we can raise $A(x)$ to the $p^j$-th power using only $m - 1$ subfield multiplications if we make use of Corollary 2 and the precomputed values of $x^{jp}, 1 \leq j \leq m - 1$.

Consider $p = 2^{31} - 1$, $P(x) = x^6 - 7$. Using Corollary 2, we can precompute the values needed for the subfield multiplications for both the $p$ and $p^2$ case. These are found in Table 5.1.

Table 5.1: Precomputed inversion constants for $GF((2^{31} - 1)^6)$ with field polynomial $P(x) = x^6 - 7$.

\[
\begin{align*}
    x^p \mod P(x) &\equiv 1513477736 x \\
    x^{2p} \mod P(x) &\equiv 1513477735 x^2 \\
    x^{3p} \mod P(x) &\equiv -1 x^3 \\
    x^{4p} \mod P(x) &\equiv 634005911 x^4 \\
    x^{5p} \mod P(x) &\equiv 634005912 x^5 \\
    x^{p^2} \mod P(x) &\equiv 1513477735 x \\
    x^{2p^2} \mod P(x) &\equiv 634005911 x^2 \\
    x^{3p^2} \mod P(x) &\equiv x^3 \\
    x^{4p^2} \mod P(x) &\equiv 1513477735 x^4 \\
    x^{5p^2} \mod P(x) &\equiv 634005911 x^5
\end{align*}
\]

5.2 Itoh and Tsujii Inversion for OEFs

Returning now to the problem of inverting non-zero elements in an OEF, recall that we observed $\alpha^{(p^m-1)/(p-1)} \in GF(p)$. We begin with a simple algebraic substitution:

$$A^{-1}(x) = (A^r)^{-1}(x)A^{-1}(x), \quad r = \frac{p^m - 1}{p - 1}$$  \hspace{1cm} (5.6)
Algorithm 5 describes the procedure for computing the inverse according to Equation (5.6). In the following, we will address the individual steps of the algorithm. Capital letters denote extension field elements while lowercase letters denote subfield elements.

**Algorithm 5** Optimal Extension Field Inversion

**Require:** $A(x) \in GF(p^m)^*$

**Ensure:** $A(x)B(x) \equiv 1 \mod P(x), B(x) = \sum b_i x^i$

1. $B(x) \leftarrow A(x)$
2. Use an addition chain to compute $B(x) \leftarrow B(x)^{r-1}$
3. $c_0 \leftarrow B(x)A(x)$
4. $c \leftarrow c_0^{-1}$
5. $B(x) \leftarrow B(x)c$

The core of the algorithm is an exponentiation to the $r$-th power. We have the following power series representation for $r$:

$$r = p^{m-1} + p^{m-2} + \cdots + p + 1.$$  \hspace{1cm} (5.7)

Thus, we have the $p$-adic representation $r - 1 = (11\ldots10)_p$. To evaluate our expression in Equation (5.6), we require an efficient method to evaluate $A^{r-1}(x)$. For a given field, $r - 1$ will be fixed. Thus, our problem is to raise a general element to a fixed exponent. One popular method of doing this is an addition chain.

From analogous results in [GP97] and [IT88], we see that using such an addition chain constructed from the $p$-adic representation of $r - 1$ requires:

$$\#\text{general multiplications} = \lceil \log_2(m - 1) \rceil + \text{HW}(m - 1) - 1$$  \hspace{1cm} (5.8)

$$\#\text{Frobenius maps} = \lceil \log_2(m - 1) \rceil + \text{HW}(m - 1)$$  \hspace{1cm} (5.9)

where $\text{HW}$ is the Hamming weight of the operand.
Given the inversion constants in Table 5.1, we can now present an addition chain for this field. We compute $A^{r-1}(x)$ as shown in Algorithm 6. In this algorithm, all exponents are understood to be expressed in base $p$ for clarity. This example requires three exponentiations to the $p$-th power, one exponentiation to the $p^2$-th power and three general multiplications, as predicted by Equation (5.8).

**Algorithm 6** Addition Chain for $A^{r-1}$ in $GF((2^{31} - 1)^6)$

**Require:** $A \in GF(p^m)^*$

**Ensure:** $B \equiv A^{r-1} \mod P(x)$

\[
\begin{align*}
B & \leftarrow A^p = A^{(10)} \\
B_0 & \leftarrow B_A = A^{(11)} \\
B & \leftarrow B_0^p = A^{(1100)} \\
B & \leftarrow B_B0 = A^{(1111)} \\
B & \leftarrow B_B^p = A^{(11110)} \\
B & \leftarrow B_A = A^{(11111)} \\
B & \leftarrow B_B^p = A^{(111110)}
\end{align*}
\]

We observe that $A(x)^r$ is always an element of $GF(p)$ due to the form chosen for $r$. Thus, to compute its inverse according to Equation 5.6, we use a single-precision implementation of the Binary Extended Euclidean Algorithm. At this point in our development of the OEF inversion algorithm, we have computed $A(x)^{r-1}$ and $(A(x)^r)^{-1}$. Multiplying these two elements gives $A(x)^{-1}$ and we are done.

In terms of computational complexity, the critical operations are the computations of $A(x)^{r-1}$ and $c_0^{-1}$. To compute $A(x)^{r-1}$, we require $\left\lceil \log_2(m-1) \right\rceil + H_w(m-1) - 1$ general multiplications and $\left\lceil \log_2(m - 1) \right\rceil + H_w(m - 1)$ exponentiations to a $p^i$-th power. Since the computation of $c_0$ results in a constant polynomial, we only need $m$ subfield multiplications and a multiplication by $\omega$, as given in the following formula, where we take $A(x) = \sum a_i x^i$ and $B(x) = \sum b_i x^i$:

\[
c_0 = \omega(a_1 b_{m-1} + \cdots + a_{m-1} b_1) + (a_0 b_0)
\]
Further, in the last step of Algorithm 5, since $c$ is also a constant polynomial, we only need $m$ subfield multiplications.

Each exponentiation to a $p^j$-th power requires $m - 1$ subfield multiplications. Each general polynomial multiplication requires $m^2 + m - 1$ subfield multiplications including those for modular reduction. Thus a general expression for the complexity of this algorithm in terms of subfield multiplications is:

$$
\#SM = \left[ \lceil \log_2(m - 1) \rceil + H_w(m - 1)(m - 1) \right] (m - 1) \\
+ \left[ \lceil \log_2(m - 1) \rceil + H_w(m - 1) - 1 \right] (m^2 + m - 1) + 2m \quad (5.10)
$$

The subfield inverse may be computed by any method. Since elements of the subfield fit into a single register, any method for single-precision inversion may be used. Our experience indicates that the Binary Extended Euclidean Algorithm is the superior choice for $p \approx 2^{31}$ and $p \approx 2^{61}$. Of course, for smaller choices of $p$, one may use a precomputed table of subfield inverses.

Finally we note that for small values of $m$, in particular $m = 3$, the direct inversion method in [KMKH99] requires somewhat fewer subfield multiplications. However, a subfield inverse is also required.
Chapter 6

Fast Polynomial Multiplication

Polynomial multiplication is required to implement both the elliptic curve group operation and the algorithm for inversion given in Section 5. In this section, we give a method to reduce the complexity of polynomial multiplication. The method is related to Karatsuba’s method [Knu81], but is optimized for multiplication of polynomials with $3i$ coefficients, for $i$ a positive integer. We observe that OEFs with $m = 3$ and $m = 6$ are well suited for 64-bit and 32-bit processors, respectively. For polynomial degrees that are relevant for ECCs, we show that on Intel microprocessors, this method yields a 10% reduction in the time required for the overall scalar multiplication.
6.1 Polynomials of Degree 2

Consider the degree-2 polynomials:

\[ A(x) = a_2x^2 + a_1x + a_0 \]
\[ B(x) = b_2x^2 + b_1x + b_0 \]

The product of \( A(x) \) and \( B(x) \) is given by:

\[
C'(x) = \sum_{i=0}^{4} c'_i x^i = A(x)B(x) = [a_2b_2]x^4 + [a_2b_1 + a_1b_2]x^3 + [a_1b_0 + a_0b_1 + a_0b_2]x^2 + [a_1b_0 + a_0b_1]x + [a_0b_0]
\]

Using the schoolbook method for polynomial multiplication, we require nine inner products. However, we can derive a more efficient method. We define the following auxiliary products:

\[
D_0 = a_0b_0 \\
D_1 = a_1b_1 \\
D_2 = a_2b_2 \\
D_3 = (a_0 + a_1)(b_0 + b_1) \\
D_4 = (a_0 + a_2)(b_0 + b_2) \\
D_5 = (a_1 + a_2)(b_1 + b_2)
\]

We can construct the coefficients of \( C'(x) \) from the \( D_i \) terms using only addi-
Fast Multiplication

Multiplications and subtractions:

\[
\begin{align*}
c'_0 &= D_0 \\
c'_1 &= D_3 - D_1 - D_0 = (a_0b_0 + a_1b_1 + a_1b_0 + a_1b_1) - a_1b_1 - a_0b_0 \\
c'_2 &= D_4 - D_2 - D_0 + D_1 = (a_0b_0 + a_2b_0 + a_0b_2 + a_2b_2) - a_2b_2 - a_0b_0 + a_1b_1 \\
c'_3 &= D_5 - D_1 - D_2 = (a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2) - a_1b_1 - a_2b_2 \\
c'_4 &= D_2
\end{align*}
\]

Thus, the only multiplications that are needed are in the \( D_i \) products. The complexity of this method is:

<table>
<thead>
<tr>
<th>Method</th>
<th>#MUL</th>
<th>#ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>schoolbook</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>new</td>
<td>6</td>
<td>6 + 7 = 13</td>
</tr>
</tbody>
</table>

where we treat subtractions as additions. Thus, with this method, we are able to trade multiplications for additions and subtractions. On most microprocessors, the operation of addition is much faster than multiplication. However, on digital signal processors, for example, the number of cycles required for a multiplication is often the same as those required for an addition. It is useful, then, to develop a simple timing model for both multiplication methods.

Let \( r = T_{MUL} / T_{ADD} \) on a given platform, where \( T_{MUL} \) and \( T_{ADD} \) are the time required for a subfield multiplication and a subfield addition, respectively. We first analyze the schoolbook method of polynomial multiplication. The time complexity
Fast Multiplication

of this algorithm is given by:

\[ T_{SB} = 9T_{MUL} + 4T_{ADD} = (9r + 4)T_{ADD} \quad (6.1) \]

Then the time complexity of the Karatsuba variant is given by:

\[ T_K = 6T_{MUL} + 13T_{ADD} = (6r + 13)T_{ADD} \quad (6.2) \]

Given these relationships, it is useful to consider for which values of \( r \) this method is of advantage. Specifically, we want the values of \( r \) for which \( T_{SB} > T_K \).

\[
T_{SB} > T_K \\
(9r + 4)T_{ADD} = (6r + 13)T_{ADD} \\
\quad r = 3
\]

As a rough guideline we can conclude that this new method is of advantage when the ratio of multiplication time to addition time is greater than or equal to three. Of course, when using a superscalar processor, the value of \( r \) may depend not only on the cycle counts for multiplication and addition, but also on the data flow dependencies in the code. Some processors may have multiple functional units available to compute additions and only one multiplier, for instance. On such a system, if it is possible to fully utilize all functional units, the operation of addition in effect is speeded up by the ability to perform additions in parallel. This is true even if a multiplication and addition each consume the same number of cycles. The possibility of instruction-level parallelism must be taken into account when determining a suitable value for \( r \).
6.2 Polynomials of Degree 5

Given the above algorithm to compute the product of polynomials of degree 2, we can formulate a procedure to compute the product of polynomials of degree 5. This algorithm combines the degree-2 method in Section 6.1 with a single iteration of the Karatsuba method [Knu81]. As above, we consider the general polynomials:

\[ A(x) = \sum_{i=0}^{5} a_i x^i = (a_5 x^2 + a_4 x + a_3) x^3 + (a_2 x^2 + a_1 x + a_0) = A_h(x) x^3 + A_l(x) \]

\[ B(x) = \sum_{i=0}^{5} b_i x^i = (b_5 x^2 + b_4 x + b_3) x^3 + (b_2 x^2 + b_1 x + b_0) = B_h(x) x^3 + B_l(x) \]

In this way, we decompose each degree-5 polynomial into two degree-2 polynomials in the indeterminate \( x^3 \). We define the auxiliary products:

\[ E_0(x) = A_l(x) B_l(x) \]
\[ E_1(x) = (A_h(x) + A_l(x))(B_h(x) + B_l(x)) \]
\[ E_2(x) = A_h B_h \]

Then our product \( C'(x) \) is given by:

\[ C'(x) = E_2(x) x^6 + [E_1(x) - E_0(x) - E_2(x)] x^3 + E_0(x) \quad (6.3) \]

As above, the only multiplications required are in the auxiliary products \( E_i \). The key idea is to compute \( E_0(x) \), \( E_1(x) \), and \( E_2(x) \), with the method for multiplication of degree-2 polynomials described in Section 6.1.

We observe that there is some overlap which must be resolved between \( E_2(x) x^6 \),
Fast Multiplication

\[(E_1(x) - E_0(x) - E_2(x)) \times^3, \text{ and } E_0(x). \] \(E_2(x) \times^6\) is an expression of the form \(\alpha_{10}x^{10} + \alpha_9x^9 + \alpha_8x^8 + \alpha_7x^7 + \alpha_6x^6,\) while \([E_1(x) - E_0(x) - E_2(x)] \times^3\) has the form \(\beta_7x^7 + \beta_6x^6 + \beta_5x^5 + \beta_4x^4 + \beta_3x^3,\) and we have to compute two subfield additions to obtain the result. A similar situation arises with \([E_1(x) - E_0(x) - E_2(x)] \times^3\) and \(E_0(x).\) Thus in total we require 4 subfield additions to construct the result on top of the 10 subfield subtractions needed for \([E_1(x) - E_0(x) - E_2(x)]\).

As above, we consider the complexity of this algorithm:

<table>
<thead>
<tr>
<th></th>
<th>#MUL</th>
<th>#ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>schoolbook</td>
<td>6^2 = 36</td>
<td>(6 - 1)^2 = 25</td>
</tr>
<tr>
<td>new</td>
<td>3 \times 6 = 18</td>
<td>3 \times 13 + (3 + 3) + (5 + 5) + 4 = 59</td>
</tr>
</tbody>
</table>

Similarly, we solve for \(r\) to determine the break even point:

\[
T_{SB} > T_{ADD}
\]

\[
(36r + 25)T_{ADD} = (18r + 59)T_{ADD}
\]

\[
r = \frac{34}{18} \approx 2
\]

Thus we see that the break even point is lower for degree-5 polynomials than for degree-2 polynomials. Our computational experiments indicate that on a 233 MHz Pentium/MMX, use of this polynomial multiplication procedure yields a 20% speedup over the time required for a polynomial multiplication using the schoolbook method. Use of this procedure yields a 10% speedup in the overall scalar multiplication time.
Chapter 7

Fast Scalar Multiplication

In [KMKH99], the authors present an optimization for OEFs which applies to certain elliptic curves. The content of this section is a discussion of their work. An elliptic curve over $GF(p^m), p > 3$, is an equation of the form:

$$E : y^2 \equiv x^3 + ax + b$$

where $a, b \in GF(p^m)$. The optimization in [KMKH99] applies when $a, b \in GF(p)$. In this case, the Frobenius map, as described in Section 5.1 is an endomorphism on the curve and thus if $(x, y) \in E/GF(p^m)$, then $(x^p, y^p) \in E/GF(p^m)$. In Section 5.1, methods are described for efficient evaluation of iterates of the Frobenius map.

Scalar multiplication on an elliptic curve is an operation of the form $kP$ for an integer $k$ and curve point $P$. That is, $kP$ is the addition of $P$ to itself $k$ times. The canonical methods for exponentiation including the binary method [Knu81] may be used to speed this operation. Given our Frobenius endomorphism which we denote by $\phi$, however, we can improve over these methods.
The Frobenius endomorphism on an elliptic curve satisfies the equation

\[ \phi^2 - t\phi + p = 0, \quad -2\sqrt{p} \leq t \leq 2\sqrt{p}. \]  \hspace{1cm} (7.1)

The quantity \( t \) is called the trace of Frobenius and is defined by [BSS99]:

\[ \#E/\text{GF}(p^m) = p^m + 1 - t \]

Thus we can expand our multiplier as

\[ k = \sum_{i=0}^{l} u_i\phi^i \]  \hspace{1cm} (7.2)

where \(-\frac{p}{2} \leq u_i \leq \frac{p}{2}\). In this equation, \( l \) will be roughly \( 2m + 3 \) \[\text{KMKH99}\].

Then, as in Section 5.1 we can exponentiate using this \( \phi \)-adic representation of the multiplier.

However, since \( u_i \) may grow as large as \( \frac{p}{2} \), this observation is mainly helpful only when \( p \) is very small, such as \( p = 2, 3 \). In order to adapt this method to be effective for larger \( p \), \[\text{KMKH99}\] presents a table look-up method, which is found as Algorithm 7. The symbol \( O \) denotes the Point at Infinity on the elliptic curve.

The algorithm proceeds by first finding a \( \phi \)-adic representation for \( k \) as in Equation 7.2. This task is accomplished in the first while loop using Equation 7.1.

Next the \( \phi \)-adic representation for \( k \) is optimized with two operations. The first reduces its length from \( 2m + 3 \) digits to \( m \) digits. This reduction is accomplished due to the fact that the \( m \)-th iteration of the Frobenius map is the Identity map. Thus we can use the rule \( \phi^m \equiv 1 \in End_E \) to perform a modular reduction on the
Algorithm 7 Base-ϕ Scalar Multiplication Procedure

Require: $k$ an integer, $P \in E/\mathbb{GF}(p^m), p, t$

Ensure: $Q = kP$

$i \leftarrow 0, x \leftarrow k, y \leftarrow 0, u_j \leftarrow 0$

while $x \neq 0$ or $y \neq 0$

$u_i \leftarrow x \mod p$

$v \leftarrow (x - u_i)/p$

$x \leftarrow tv + y$

$y \leftarrow -v$

$i \leftarrow i + 1$

end while

for $0 \leq i < m$

$d_i \leftarrow u_i + u_{i+m} + u_{i+2m}$

end for

for $0 \leq i < m$

$c_i \leftarrow d_i - z$, where $z$ is an integer that minimizes $\sum_i HW(c_i)$

end for

for $0 \leq i < m$

$P_i \leftarrow \phi^i P$

end for

$Q \leftarrow O$

$Q \leftarrow 2Q$

$j \leftarrow \lceil \log_2 p \rceil + 1$

while $j \geq 0$

for $0 \leq i < m - 1$

if $c_{ij} = 1$

$Q \leftarrow Q + P_i$

end if

end for

$j \leftarrow j - 1$

end while
Fast Scalar Multiplication

\( \phi \)-adic representation. Thus:

\[
\sum_{i=0}^{[2 \log_2 k]+3} u_i \phi^i = \sum_{i=0}^{m-1} (u_i + u_{i+m} + u_{i+2m}) \phi^i = \sum_{i=0}^{m-1} d_i \phi^i.
\]

(7.3)

In addition, we can reduce the number of 1s in the 2-adic representation of the digits in the \( \phi \)-adic representation of \( k \) since [KMKH99]:

\[
\sum_{i=0}^{m-1} \phi^i = 0.
\]

(7.5)

The algorithm finishes by building a table of the iterates of the Frobenius map applied to the base point \( P \). It then computes the scalar multiplication of \( P \) by the optimized \( \phi \)-adic representation for \( k \). For the case of \( m = 7 \), the authors report an 68% reduction in the number of elliptic curve operations required from approximately \( 10.5 \lceil \log_2 p \rceil \) to \( 3.4 \lceil \log_2 p \rceil \). The net result on a 400 MHz Pentium/II in the field \( GF((2^{31} - 1)^7) \) is a full scalar multiplication time of 1.95 msec.
Chapter 8

Implementation Results

One of the most important applications of our technique is in elliptic curve cryptosystems, where Galois field arithmetic performance is critical to the performance of the entire system. We show that an OEF yields substantially faster software finite field arithmetic than those previously reported in the literature.

We implemented our algorithms on two platforms. One platform is the DEC Alpha 21064 and 21164A workstations. These RISC computers have a 64-bit architecture. Thus a good choice for $p$ would be $2^{61} - 1$ with an extension degree $m = 3$ since an ECC over a field of approximately $2^{183}$ elements appears quite secure. This implementation is written in optimized C. In addition, we found that the performance of the subfield inverse depended heavily on the organization of branches in the code. A reduction in the number of branches at the expense of copying data proved to be effective in reducing run time. For the DEC Alpha implementation, using our polynomial multiplication formulas presented in Section 6.1 yields a 30% speedup on the 21164A and a 25% speedup on the 21064. Thus, the times reported here for the operations that rely on multiplication use the methods from Section 6.
In addition, we implemented our algorithms on a 233 MHz Intel Pentium MMX using Microsoft Visual C++ version 6.0. This computer has a 32-bit architecture. Thus a good choice for $p$ would be $2^{31} - 1$ with an extension degree $m = 6$ yielding a finite field with approximately $2^{186}$ elements. The Pentium implementation is entirely in C. Because of the larger extension degree required on the Pentium, we observe a roughly 20% speedup due to the formulas in Section 6, which is reflected in the timings reported here.

For our implementation of EC scalar point multiplication, we used the sliding window method with a maximum window size of 5. In addition, we used non-adjacent form balanced ternary to represent the multiplicand [KT92]. To represent the coordinates of points on the curve, we used an affine representation since inversion in an OEF can be performed at moderate cost. In contrast, previous work [BP98] has reported performance numbers using projective coordinates to represent points, thereby avoiding the need to perform inversion.

In order to obtain accurate timings, we executed full scalar multiplication with random multiplicand one thousand times, observed the execution time, and computed the average.

The other arithmetic operations for which we report timings were executed one million times. Tables 8.1 and 8.2 show the result of our timing measurements.

We observe that the ratio of multiplication time to inversion time is highly platform-dependent. On the Alpha 21064, we see a ratio of approximately 5.3. On the Alpha 21164A, we have a ratio of approximately 7.9. On the Intel Pentium, we have a ratio of 5.5. In each of these cases, the ratio is low enough to provide improved performance when compared with a projective space representation of the curve points.

As a final remark, we observe that for some processors, it may be still be
Table 8.1: OEF arithmetic timings in $\mu$sec on DEC Alpha microprocessors for the field $GF((2^{61} - 1)^3)$ with field polynomial $P(x) = x^3 - 5$

<table>
<thead>
<tr>
<th></th>
<th>Alpha 21064, 150 MHz</th>
<th>Alpha 21164A, 600 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schoolbook Multiplication</td>
<td>3.67</td>
<td>0.48</td>
</tr>
<tr>
<td>Karatsuba-variant Multi-</td>
<td>2.77</td>
<td>0.34</td>
</tr>
<tr>
<td>Plication</td>
<td>8.13</td>
<td>1.81</td>
</tr>
<tr>
<td>$GF(p)$ inverse</td>
<td>14.6</td>
<td>2.68</td>
</tr>
<tr>
<td>$GF(p^m)$ inverse</td>
<td>26.1</td>
<td>4.45</td>
</tr>
<tr>
<td>Affine EC doubling</td>
<td>30.5</td>
<td>4.79</td>
</tr>
<tr>
<td>Affine point multiplication</td>
<td>6.57 msec</td>
<td>1.06 msec</td>
</tr>
</tbody>
</table>

Table 8.2: OEF arithmetic timings in $\mu$sec on Intel microprocessors for the field $GF((2^{31} - 1)^6)$ with field polynomial $P(x) = x^6 - 7$

<table>
<thead>
<tr>
<th></th>
<th>Pentium/MMX, 233 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schoolbook Multiplication</td>
<td>5.82</td>
</tr>
<tr>
<td>Karatsuba-variant Multi-</td>
<td>4.60</td>
</tr>
<tr>
<td>Plication</td>
<td>4.15</td>
</tr>
<tr>
<td>$GF(p)$ inverse</td>
<td>25.3</td>
</tr>
<tr>
<td>$GF(p^m)$ inverse</td>
<td>44.8</td>
</tr>
<tr>
<td>Affine EC addition</td>
<td>52.4</td>
</tr>
<tr>
<td>Affine EC doubling</td>
<td>11.4 msec</td>
</tr>
<tr>
<td>Affine point multiplication</td>
<td></td>
</tr>
</tbody>
</table>
Implementation Results

advantageous to use projective coordinates to represent elliptic curve points and thus postpone field inversions in the elliptic curve group operation until the end of the computation. Consider the 500 MHz Alpha 21264, which has a fully-pipelined integer multiplier [Com99]. This hardware improvement dramatically improves the time for an extension field multiplication from 0.34 $\mu$sec to 0.18 $\mu$sec, despite the fact that our 21164A test system is clocked at 600 MHz while our 21264 test system runs at only 500 MHz. This architectural improvement does not speed the Binary Extended Euclidean Algorithm however, so the time for an extension field inversion is only slightly improved from 2.68 $\mu$sec to 2.44 $\mu$sec. In this case, the ratio of multiplication to inversion time grows to 13.5. Thus, our best result on the 500 MHz Alpha 21264 of 0.75 msec for a full scalar multiplication is achieved using projective coordinates. This result once again confirms our thesis that to achieve optimal performance for an elliptic curve cryptosystem, one must tailor the choice of algorithms and finite fields to match the underlying hardware.
Chapter 9

OEFs in Practice

When implementing cryptosystems in the real world, several concerns arise in addition to high performance and hardness of the underlying problem. In this section, we address two issues which must be resolved to use OEFs in a secure real-world system.

9.1 Key Validation

In practical usage of a public-key cryptosystem, two parties perform computations in some mathematical structure such as a ring, field, or elliptic curve. The two parties can generally be assumed to be mutually distrustful. This presents a problem: one party must unilaterally choose a mathematical structure in which computations may be performed. In fact, an attacking party could choose a structure in which the assumed hard problem isn’t very hard at all. For example, there are special cases for which the elliptic curve discrete logarithm problem can be transformed into an easy problem, such as in the case where the number of points on the curve equals the number of elements in the field over which the curve is defined. Further, an attacker could provide bogus parameters which do not define the claimed mathematical struc-
ture. For example, an attacker may be able to solve a discrete logarithm problem more easily if she selects certain parameters [Gor93].

It is useful then to ask: what reasonable steps may be taken to ensure that alleged parameters specifying an elliptic curve cryptosystem over an OEF are plausible? In the following, let the field be $GF(p^m)$, the field polynomial be $P(x)$, the elliptic curve be $y^2 \equiv x^3 + ax + b$ and an elliptic curve point be $W = (v, w)$.

1. Check that $p$ is an odd prime rational integer.
2. Check that $m$ is a positive integer and that $P(x)$ is of degree $m$.
3. Check that $P(x)$ is irreducible.
4. Check that $v^2 \equiv w^3 + aw + b \in GF(p^m)$.

These simple tests allow a user to verify that parameters for an elliptic curve cryptosystem defined over an OEF are valid.

9.2 Conversion from Field Elements to Integers and Octet Strings

Many practical cryptosystems require a user to convert between field elements and integers and/or octet strings. For example, in real-world usage of the Diffie-Hellman key exchange [DH76], users often perform some finite field computations, then find a corresponding octet string to represent the result. This octet string can then be used as input to a hash function, or as a key for a symmetric cipher.

In the case of $GF(p)$ and $GF(2^m)$, the conversion is straightforward: simply take the target computer’s binary representation of a field element and treat it as a
string of octets. In the OEF case, however, things are slightly more difficult.

Suppose a user chooses some \( p = 2^n - c, m \) for her OEF. Then each element of the field can be represented in \( m \) computer words. However, for each word, there will be only \( 2^n - c \) possible values instead of \( 2^n \). Thus the number of possible octet strings formed from the concatenation of the \( m \) computer words is reduced by \( cm \) due to the representation.

To address this problem, a user may simply perform radix conversion arithmetic to find a “densely packed” octet string representation. Thus the field element:

\[
A(x) = a_{m-1}x^{m-1} + \cdots + a_2x^2 + a_1x + a_0
\]

may be represented by the integer \( I \) defined by:

\[
I = a_{m-1}p^{m-1} + \cdots + a_2p^2 + a_1p + a_0
\]

The integer \( I \) will thus have a bit length of \( \lceil m \log_2 p \rceil \). Assuming values for \( p^i \) are precomputed and stored, the effort required to compute \( I \) is essentially \( m - 1 \) subfield multiplications and \( m \) subfield additions. Thus the time to compute \( I \) is negligible when compared to the time required to perform an elliptic curve point multiplication.

To find the element \( A(x) \), some simple radix conversion operations are required. Starting from \( r = m - 1 \) down to \( r = 0 \), simply divide \( I \) by \( p^r \), where \( r \) is the corresponding coefficient of \( A(x) \) desired. The quotient at each step will be the \( r \)th coefficient of \( A(x) \). After each step, set the new value of \( I \) to the remainder.
Chapter 10

OEF Construction and Statistics

In the above sections we have shown that OEFs can offer particular advantages in arithmetic performance when compared with other approaches. It is useful, then, to ask how to construct an OEF and how many OEFs exist of various types. It turns out that OEF construction may be done in an efficient manner using a relatively simple algorithm. We provide statistics on the number of OEFs that exist for various choices of \( n \), and tables of OEFs which may be used in applications.

10.1 Type II OEF Construction Algorithm

Constructing an OEF for a particular application is an essentially straightforward process. Let \( n, c, m, \) and \( \omega \) be positive rational integers. Then we require a prime \( p = 2^n \pm c \), an extension degree \( m \), and a constant \( \omega \) such that these parameters form an irreducible binomial \( x^m - \omega \) over \( GF(p) \).

Theorem 1 gives us the necessary and sufficient conditions on these parameters. For simplicity of presentation, we present an algorithm to construct a Type II OEF,
fixing $\omega = 2$. Even with this restriction, OEFs are plentiful. This algorithm is an improvement over that found in [Bai98] since Algorithm 3 can be used to exhaustively find all Type II OEFs.

The algorithm proceeds by finding pseudo-Mersenne primes and then checking possible extension degrees $m$ for the existence of a binomial. For our application, word size $n$ will be chosen based on the attributes of the target microprocessor. Typical microprocessor word sizes lie between 8 and 64 bits, while a commonly used upper bound for field orders used in elliptic curve cryptography is $2^{256}$. It suffices for this application, then, to search for $m$ up to 32, allowing for the largest possible field order with the smallest typical word size.

We present results from the use of this algorithm to construct tables in the Appendix. Let $c$ and $n$ be positive rational integers. Algorithm 3 finds OEFs with primes of the form $2^n - c$; a trivial change finds OEFs with primes of the form $2^n + c$, if such a field is required. In addition, minor changes to this algorithm will produce Type I OEFs or general OEFs.

A practical implementation of this algorithm would be greatly improved by using sieve methods rather than simply testing consecutive integers for primality. The algorithm is presented in this form for clarity.

The most time consuming part of this algorithm is the factorization of $p - 1$. For our implementation which produced the results in the Appendix, we used trial division with small integers of the form $\pm 1 \pmod{6}$ to extract small factors and Pollard’s Rho Method to recover the remaining factors. This factorization is needed only to compute the order of 2. To our knowledge, it is an open problem to devise a method to compute this order without the full factorization of $p - 1$. 


Algorithm 8 Type II Optimal Extension Field Construction Procedure

**Require:** $n$ given, low, high bounds on bit length of field order

**Ensure:** $p, m$ define a Type II Optimal Extension Field with field order between $2^{\text{low}}$ and $2^{\text{high}}$.

$c \leftarrow 1$

for $\log_2 c \leq \lfloor \frac{1}{2} n \rfloor$ do

$p \leftarrow 2^n - c$

if $p$ is prime then

factor $p - 1$

$\text{ord}_2 \leftarrow$ the order of $2 \in GF(p)^*$

for $m \leftarrow 2$ to $32$ do

if $m \times n \geq \text{low}$ and $m \times n \leq \text{high}$ then

$\text{BadMValue} \leftarrow 0$

for each prime divisor $d$ of $m$ do

if $d \nmid \text{ord}_2$ then

$\text{BadMValue} \leftarrow 1$

Break

end if

end for

if $\text{BadMValue} = 0$ then

if $m \equiv 0 \pmod{4}$ then

if $p \equiv 1 \pmod{4}$ then

return $p, m$

end if

else

return $p, m$

end if

end if

end if

end for

$c \leftarrow c + 2$

end for
10.2 Statistics on the Number of OEFs

We implemented Algorithm 3 on a variety of high-end RISC workstations including DEC Alphas and Sun Sparc Ultras, with an aim toward counting the number of Type II OEFs of approximate order between $2^{130}$ and $2^{256}$. The results from this computation are found in Tables A.2, A.3, and A.4. Each table lists subfield bitlengths going down the column and extension degrees across the rows.

10.3 Statistics on the Number of Pseudo-Mersenne Primes

Many interesting open questions exist in analytic number theory concerning the existence of primes in short intervals. We denote the number of primes not exceeding $x$ as $\pi(x)$. One result in [IP84] shows that

$$\pi(x) - \pi(x - x^{23/42}) > (x^{23/42})/(100 \log x). \quad (10.1)$$

A more recent result due to R. Baker and G. Harman analyzes the interval $\pi(x) - \pi(x - x^{535})$ [Rib96]. Cramer shows that the Extended Riemann Hypothesis implies the difference between a particular prime $p_n$ and the next consecutive prime number is $O(p_n^{1/2} \log p_n)$ [Rib96]. Of course, these results are only asymptotically true.

To exactly determine the number of pseudo-Mersenne primes, we need a result concerning the intervals $\pi(2^n) - \pi(2^n - 2^{(1/2)n})$ and $\pi(2^n + 2^{(1/2)n}) - \pi(2^n)$, about which nothing appears to be known as of this writing [Kob98]. It is important to note that this question concerning the number of primes in a short interval also arises in choosing an elliptic curve over any finite field for cryptographic use.
Since there are no known results of this type which apply to our case of pseudo-Mersenne primes, we explicitly computed the number of primes for $2^n \pm c$, where $7 \leq n \leq 58$ and $\log_2 c \leq \lfloor \frac{1}{7}n \rfloor$. The results are found in Table A.1.

10.4 Tables of Type I and Type II OEFs

The appendix contains tables of OEFs for use in practical applications. Table A.5 provides all Type I OEFs for $7 \leq n \leq 61$. For each choice of $n$ and a sign for $c$, where possible we provide three Type II OEFs, preferably with $nm \approx 160, 200, 240$, respectively, in Table A.6. We observe that due to the fast subfield multiplication available with Type I OEFs, these offer computational advantages on many platforms when compared to Type II OEFs. This is true since although a Type II OEF has $\omega = 2$ and thus implements the multiplications required for extension field modular reduction with shifts, a Type I OEF requires only one multiplication for each subfield multiply. Since subfield multiplication is by far the most often used operation, speedups here are most dramatic.
Chapter 11

Discussion

11.1 Conclusion

In this paper we have extended the work on Optimal Extension Fields by introducing an efficient algorithm for inversion. The use of this algorithm allows for an affine representation of the elliptic curve points which is more efficient than the previously reported projective space representation. In addition, we have provided formulas for fast polynomial multiplication which are particularly suited to extension degrees of the form $3i$. Finally, we have included tables of OEFs for reference and use in implementation.

Acknowledgments

Gabriel Kostolny provided data management and report generation scripts which were invaluable for generating the tables in this paper.

We would like to thank Hans-Georg Rück for an early idea regarding the
Discussion

Karatsuba variant for degree-2 polynomials.
Appendix A

Tables
### Table A.1: Number of Pseudo-Mersenne Primes, $2^n \pm c$, $\log_2 c \leq \lceil(n/2)\rceil$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$2^n - c$</th>
<th>$2^n + c$</th>
<th>$n$</th>
<th>$2^n - c$</th>
<th>$2^n + c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>33</td>
<td>2886</td>
<td>2852</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>4</td>
<td>34</td>
<td>5667</td>
<td>5477</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>2</td>
<td>35</td>
<td>5379</td>
<td>5263</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
<td>36</td>
<td>10413</td>
<td>10503</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>3</td>
<td>37</td>
<td>10197</td>
<td>10254</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>9</td>
<td>38</td>
<td>19799</td>
<td>19812</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>7</td>
<td>39</td>
<td>19461</td>
<td>19502</td>
</tr>
<tr>
<td>14</td>
<td>11</td>
<td>12</td>
<td>40</td>
<td>37798</td>
<td>37871</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>13</td>
<td>41</td>
<td>36743</td>
<td>36902</td>
</tr>
<tr>
<td>16</td>
<td>21</td>
<td>30</td>
<td>42</td>
<td>71805</td>
<td>72138</td>
</tr>
<tr>
<td>17</td>
<td>19</td>
<td>20</td>
<td>43</td>
<td>70257</td>
<td>70325</td>
</tr>
<tr>
<td>18</td>
<td>38</td>
<td>42</td>
<td>44</td>
<td>137313</td>
<td>137285</td>
</tr>
<tr>
<td>19</td>
<td>40</td>
<td>29</td>
<td>45</td>
<td>134641</td>
<td>134452</td>
</tr>
<tr>
<td>20</td>
<td>70</td>
<td>77</td>
<td>46</td>
<td>263004</td>
<td>263544</td>
</tr>
<tr>
<td>21</td>
<td>65</td>
<td>70</td>
<td>47</td>
<td>257295</td>
<td>258091</td>
</tr>
<tr>
<td>22</td>
<td>129</td>
<td>137</td>
<td>48</td>
<td>504634</td>
<td>504016</td>
</tr>
<tr>
<td>23</td>
<td>117</td>
<td>131</td>
<td>49</td>
<td>493785</td>
<td>494248</td>
</tr>
<tr>
<td>24</td>
<td>251</td>
<td>249</td>
<td>50</td>
<td>969072</td>
<td>967704</td>
</tr>
<tr>
<td>25</td>
<td>240</td>
<td>258</td>
<td>51</td>
<td>947752</td>
<td>948011</td>
</tr>
<tr>
<td>26</td>
<td>477</td>
<td>455</td>
<td>52</td>
<td>1863100</td>
<td>1860984</td>
</tr>
<tr>
<td>27</td>
<td>434</td>
<td>452</td>
<td>53</td>
<td>1826661</td>
<td>1826485</td>
</tr>
<tr>
<td>28</td>
<td>871</td>
<td>840</td>
<td>54</td>
<td>3586713</td>
<td>3585449</td>
</tr>
<tr>
<td>29</td>
<td>839</td>
<td>811</td>
<td>55</td>
<td>3521537</td>
<td>3520704</td>
</tr>
<tr>
<td>30</td>
<td>1578</td>
<td>1565</td>
<td>56</td>
<td>6920100</td>
<td>7131669</td>
</tr>
<tr>
<td>31</td>
<td>1527</td>
<td>1542</td>
<td>57</td>
<td>6794704</td>
<td>6792475</td>
</tr>
<tr>
<td>32</td>
<td>2931</td>
<td>2958</td>
<td>58</td>
<td>13351601</td>
<td>13351850</td>
</tr>
</tbody>
</table>
Table A.2: Number of Type II OEFs of order between $2^{130}$ and $2^{256}$, $7 \leq n \leq 10$

<table>
<thead>
<tr>
<th>$m$=</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.3: Number of OEFs of order between $2^{130}$ and $2^{256}$, $11 \leq n \leq 18$

<table>
<thead>
<tr>
<th>$m$=</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>19</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>10</td>
<td>14</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>17</td>
<td>25</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table A.4: Number of Type II OEFs of order between $2^{130}$ and $2^{256}$, $19 \leq n \leq 55$

<table>
<thead>
<tr>
<th>$m$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>9</td>
<td>21</td>
<td>25</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>22</td>
<td>39</td>
<td>48</td>
<td>12</td>
<td>14</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>18</td>
<td>35</td>
<td>50</td>
<td>15</td>
<td>11</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>40</td>
<td>41</td>
<td>66</td>
<td>89</td>
<td>33</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>43</td>
<td>35</td>
<td>56</td>
<td>83</td>
<td>31</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>77</td>
<td>72</td>
<td>126</td>
<td>160</td>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>76</td>
<td>68</td>
<td>124</td>
<td>156</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>183</td>
<td>179</td>
<td>133</td>
<td>219</td>
<td>342</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>177</td>
<td>139</td>
<td>125</td>
<td>218</td>
<td>286</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>333</td>
<td>287</td>
<td>259</td>
<td>422</td>
<td>559</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>329</td>
<td>279</td>
<td>240</td>
<td>404</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>617</td>
<td>512</td>
<td>479</td>
<td>790</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>615</td>
<td>529</td>
<td>432</td>
<td>755</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1180</td>
<td>946</td>
<td>824</td>
<td>1442</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>1424</td>
<td>1136</td>
<td>977</td>
<td>766</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>2813</td>
<td>2180</td>
<td>1857</td>
<td>1561</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>2636</td>
<td>2126</td>
<td>1755</td>
<td>1483</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>5154</td>
<td>4149</td>
<td>3359</td>
<td>2967</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>5095</td>
<td>4139</td>
<td>3429</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>9871</td>
<td>7911</td>
<td>6599</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>9749</td>
<td>7771</td>
<td>6380</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>18864</td>
<td>15179</td>
<td>12499</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>18533</td>
<td>14656</td>
<td>12286</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>36074</td>
<td>28817</td>
<td>23951</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>35215</td>
<td>27905</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>91499</td>
<td>68735</td>
<td>55042</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>89336</td>
<td>67300</td>
<td>53918</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>175514</td>
<td>131656</td>
<td>105347</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>172251</td>
<td>128937</td>
<td>102966</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>336066</td>
<td>252095</td>
<td>201375</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>329827</td>
<td>247247</td>
<td>197553</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>645703</td>
<td>483609</td>
<td>387502</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>315731</td>
<td>236628</td>
<td>189774</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>1241533</td>
<td>931675</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>1218801</td>
<td>913858</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>2391808</td>
<td>1792593</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>2347560</td>
<td>1760993</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table A.5: Type I OEFs for $7 \leq n \leq 61$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$c$</th>
<th>$m$</th>
<th>$mn$</th>
<th>$\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>-1</td>
<td>21</td>
<td>147</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>-1</td>
<td>27</td>
<td>189</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>32</td>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>-1</td>
<td>13</td>
<td>169</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>-1</td>
<td>10</td>
<td>130</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>-1</td>
<td>14</td>
<td>182</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>-1</td>
<td>15</td>
<td>195</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>-1</td>
<td>18</td>
<td>234</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>-1</td>
<td>16</td>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>-1</td>
<td>10</td>
<td>170</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>-1</td>
<td>15</td>
<td>255</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>-1</td>
<td>7</td>
<td>133</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>-1</td>
<td>9</td>
<td>171</td>
<td>3</td>
</tr>
<tr>
<td>31</td>
<td>-1</td>
<td>6</td>
<td>186</td>
<td>7</td>
</tr>
<tr>
<td>31</td>
<td>-1</td>
<td>7</td>
<td>217</td>
<td>7</td>
</tr>
<tr>
<td>61</td>
<td>-1</td>
<td>3</td>
<td>183</td>
<td>37</td>
</tr>
</tbody>
</table>

### Table A.6: Type II OEFs

<table>
<thead>
<tr>
<th>$n$</th>
<th>$c$</th>
<th>$p$</th>
<th>$m$</th>
<th>$nm$</th>
<th>$n$</th>
<th>$c$</th>
<th>$p$</th>
<th>$m$</th>
<th>$nm$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>+3</td>
<td>131</td>
<td>25</td>
<td>175</td>
<td>33</td>
<td>-49</td>
<td>8589934543</td>
<td>7</td>
<td>231</td>
</tr>
<tr>
<td>7</td>
<td>+3</td>
<td>131</td>
<td>26</td>
<td>182</td>
<td>33</td>
<td>-301</td>
<td>8589934291</td>
<td>5</td>
<td>165</td>
</tr>
<tr>
<td>8</td>
<td>-5</td>
<td>251</td>
<td>25</td>
<td>200</td>
<td>33</td>
<td>-301</td>
<td>8589934291</td>
<td>6</td>
<td>198</td>
</tr>
<tr>
<td>8</td>
<td>-15</td>
<td>241</td>
<td>25</td>
<td>200</td>
<td>33</td>
<td>+29</td>
<td>8589934621</td>
<td>5</td>
<td>165</td>
</tr>
<tr>
<td>8</td>
<td>+1</td>
<td>257</td>
<td>32</td>
<td>256</td>
<td>33</td>
<td>+35</td>
<td>8589934627</td>
<td>7</td>
<td>231</td>
</tr>
<tr>
<td>8</td>
<td>+15</td>
<td>271</td>
<td>25</td>
<td>200</td>
<td>34</td>
<td>-113</td>
<td>1717869071</td>
<td>5</td>
<td>170</td>
</tr>
<tr>
<td>8</td>
<td>+15</td>
<td>271</td>
<td>27</td>
<td>216</td>
<td>34</td>
<td>-113</td>
<td>1717869071</td>
<td>7</td>
<td>238</td>
</tr>
<tr>
<td>9</td>
<td>-3</td>
<td>509</td>
<td>16</td>
<td>144</td>
<td>34</td>
<td>-165</td>
<td>1717869019</td>
<td>6</td>
<td>204</td>
</tr>
<tr>
<td>9</td>
<td>+9</td>
<td>521</td>
<td>25</td>
<td>225</td>
<td>34</td>
<td>+153</td>
<td>1717869337</td>
<td>7</td>
<td>238</td>
</tr>
<tr>
<td>9</td>
<td>+11</td>
<td>523</td>
<td>18</td>
<td>162</td>
<td>34</td>
<td>+339</td>
<td>1717869523</td>
<td>6</td>
<td>204</td>
</tr>
<tr>
<td>9</td>
<td>+11</td>
<td>523</td>
<td>27</td>
<td>243</td>
<td>34</td>
<td>+417</td>
<td>1717869601</td>
<td>5</td>
<td>170</td>
</tr>
<tr>
<td>10</td>
<td>-3</td>
<td>1021</td>
<td>16</td>
<td>160</td>
<td>35</td>
<td>-31</td>
<td>34359738337</td>
<td>7</td>
<td>245</td>
</tr>
<tr>
<td>10</td>
<td>-3</td>
<td>1021</td>
<td>20</td>
<td>200</td>
<td>35</td>
<td>-61</td>
<td>34359738307</td>
<td>6</td>
<td>210</td>
</tr>
<tr>
<td>10</td>
<td>-11</td>
<td>1013</td>
<td>23</td>
<td>230</td>
<td>35</td>
<td>-499</td>
<td>34359737869</td>
<td>4</td>
<td>140</td>
</tr>
<tr>
<td>10</td>
<td>+7</td>
<td>1031</td>
<td>25</td>
<td>250</td>
<td>35</td>
<td>+53</td>
<td>34359738421</td>
<td>5</td>
<td>175</td>
</tr>
<tr>
<td>10</td>
<td>+27</td>
<td>1051</td>
<td>14</td>
<td>140</td>
<td>35</td>
<td>+53</td>
<td>34359738421</td>
<td>6</td>
<td>210</td>
</tr>
<tr>
<td>10</td>
<td>+27</td>
<td>1051</td>
<td>25</td>
<td>250</td>
<td>35</td>
<td>+53</td>
<td>34359738421</td>
<td>7</td>
<td>245</td>
</tr>
<tr>
<td>11</td>
<td>-19</td>
<td>2029</td>
<td>13</td>
<td>143</td>
<td>36</td>
<td>-117</td>
<td>68719476619</td>
<td>6</td>
<td>216</td>
</tr>
<tr>
<td>11</td>
<td>-19</td>
<td>2029</td>
<td>16</td>
<td>176</td>
<td>36</td>
<td>-189</td>
<td>68719476547</td>
<td>7</td>
<td>252</td>
</tr>
<tr>
<td>11</td>
<td>-19</td>
<td>2029</td>
<td>18</td>
<td>198</td>
<td>36</td>
<td>-243</td>
<td>68719476493</td>
<td>4</td>
<td>144</td>
</tr>
<tr>
<td>11</td>
<td>+5</td>
<td>2053</td>
<td>16</td>
<td>176</td>
<td>36</td>
<td>+117</td>
<td>68719476853</td>
<td>4</td>
<td>144</td>
</tr>
<tr>
<td>11</td>
<td>+5</td>
<td>2053</td>
<td>18</td>
<td>198</td>
<td>36</td>
<td>+117</td>
<td>68719476853</td>
<td>6</td>
<td>216</td>
</tr>
<tr>
<td>11</td>
<td>+21</td>
<td>2069</td>
<td>22</td>
<td>242</td>
<td>36</td>
<td>+175</td>
<td>68719476911</td>
<td>7</td>
<td>252</td>
</tr>
<tr>
<td>12</td>
<td>-3</td>
<td>4093</td>
<td>16</td>
<td>192</td>
<td>37</td>
<td>-123</td>
<td>137438953349</td>
<td>4</td>
<td>148</td>
</tr>
<tr>
<td>12</td>
<td>-3</td>
<td>4093</td>
<td>18</td>
<td>216</td>
<td>37</td>
<td>-141</td>
<td>137438953331</td>
<td>5</td>
<td>185</td>
</tr>
<tr>
<td>12</td>
<td>-39</td>
<td>4057</td>
<td>13</td>
<td>156</td>
<td>37</td>
<td>-201</td>
<td>137438953271</td>
<td>5</td>
<td>185</td>
</tr>
<tr>
<td>12</td>
<td>+15</td>
<td>4111</td>
<td>15</td>
<td>180</td>
<td>37</td>
<td>+9</td>
<td>137438953481</td>
<td>5</td>
<td>185</td>
</tr>
<tr>
<td>12</td>
<td>+37</td>
<td>4133</td>
<td>16</td>
<td>192</td>
<td>37</td>
<td>+29</td>
<td>137438953501</td>
<td>4</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>+63</td>
<td>4159</td>
<td>21</td>
<td>252</td>
<td>37 +29</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>-1</td>
<td>819</td>
<td>13</td>
<td>169</td>
<td>38 -45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>-13</td>
<td>8179</td>
<td>18</td>
<td>234</td>
<td>38 -107</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>-21</td>
<td>8171</td>
<td>19</td>
<td>247</td>
<td>38 -153</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>+17</td>
<td>8209</td>
<td>19</td>
<td>247</td>
<td>38 +7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>+27</td>
<td>8219</td>
<td>14</td>
<td>182</td>
<td>38 +13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>+29</td>
<td>8221</td>
<td>12</td>
<td>156</td>
<td>38 +117</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>-3</td>
<td>16381</td>
<td>12</td>
<td>168</td>
<td>39 -19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>-3</td>
<td>16381</td>
<td>14</td>
<td>196</td>
<td>39 -67</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>-3</td>
<td>16381</td>
<td>18</td>
<td>252</td>
<td>39 -91</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>+67</td>
<td>16451</td>
<td>14</td>
<td>196</td>
<td>39 +23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>+69</td>
<td>16453</td>
<td>12</td>
<td>168</td>
<td>39 +45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>+69</td>
<td>16453</td>
<td>18</td>
<td>252</td>
<td>39 +149</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>-19</td>
<td>32749</td>
<td>12</td>
<td>180</td>
<td>40 -195</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>-75</td>
<td>32693</td>
<td>11</td>
<td>165</td>
<td>40 -195</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>+3</td>
<td>32771</td>
<td>10</td>
<td>150</td>
<td>40 +15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>+21</td>
<td>32789</td>
<td>14</td>
<td>210</td>
<td>40 +141</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>-15</td>
<td>65521</td>
<td>9</td>
<td>144</td>
<td>41 -21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>-15</td>
<td>65521</td>
<td>15</td>
<td>240</td>
<td>41 -75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>+45</td>
<td>65581</td>
<td>10</td>
<td>160</td>
<td>41 +125</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>+45</td>
<td>65581</td>
<td>12</td>
<td>192</td>
<td>41 +197</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>-13</td>
<td>131059</td>
<td>9</td>
<td>153</td>
<td>42 -11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>-31</td>
<td>131041</td>
<td>15</td>
<td>255</td>
<td>42 -333</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>+29</td>
<td>131101</td>
<td>9</td>
<td>153</td>
<td>42 +75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>+99</td>
<td>131171</td>
<td>13</td>
<td>221</td>
<td>42 +165</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>-11</td>
<td>262133</td>
<td>13</td>
<td>234</td>
<td>43 -67</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>-35</td>
<td>262109</td>
<td>11</td>
<td>198</td>
<td>43 -117</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>-93</td>
<td>262051</td>
<td>9</td>
<td>162</td>
<td>43 +29</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>-19</td>
<td>524269</td>
<td>8</td>
<td>152</td>
<td>44 -21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>+53</td>
<td>524341</td>
<td>10</td>
<td>190</td>
<td>44 +21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>+53</td>
<td>524341</td>
<td>12</td>
<td>228</td>
<td>44 +55</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>+81</td>
<td>524369</td>
<td>13</td>
<td>247</td>
<td>45 -55</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>+81</td>
<td>524369</td>
<td>19</td>
<td>205</td>
<td>45 -81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>-9</td>
<td>104857</td>
<td>8</td>
<td>160</td>
<td>45 -81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>-9</td>
<td>104857</td>
<td>12</td>
<td>240</td>
<td>45 -139</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>-5</td>
<td>104857</td>
<td>10</td>
<td>200</td>
<td>45 +59</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>+13</td>
<td>104857</td>
<td>8</td>
<td>160</td>
<td>45 +165</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>-3</td>
<td>4194301</td>
<td>9</td>
<td>198</td>
<td>47 -127</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>Value</td>
<td>Type</td>
<td>Length</td>
<td>Value</td>
<td>Type</td>
<td>Length</td>
<td>Value</td>
<td>Type</td>
<td>Length</td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
<td>------</td>
<td>--------</td>
<td>-------</td>
<td>------</td>
<td>--------</td>
<td>-------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>22</td>
<td>4194277</td>
<td>8</td>
<td>176</td>
<td>-541</td>
<td>14073488354787</td>
<td>3</td>
<td>141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>4194247</td>
<td>7</td>
<td>154</td>
<td>+5</td>
<td>14073488355333</td>
<td>3</td>
<td>141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>4194319</td>
<td>9</td>
<td>198</td>
<td>+5</td>
<td>14073488355333</td>
<td>4</td>
<td>188</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>4194389</td>
<td>8</td>
<td>176</td>
<td>+273</td>
<td>2814796710597</td>
<td>4</td>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>8388581</td>
<td>10</td>
<td>230</td>
<td>-93</td>
<td>28147967105363</td>
<td>3</td>
<td>147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>8388547</td>
<td>9</td>
<td>207</td>
<td>+75</td>
<td>2814796710491</td>
<td>5</td>
<td>245</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>8388451</td>
<td>11</td>
<td>242</td>
<td>+235</td>
<td>2814796710491</td>
<td>4</td>
<td>192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>8388619</td>
<td>7</td>
<td>161</td>
<td>+75</td>
<td>2814796710491</td>
<td>3</td>
<td>144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>8388623</td>
<td>11</td>
<td>253</td>
<td>+235</td>
<td>2814796710491</td>
<td>5</td>
<td>245</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>16777213</td>
<td>8</td>
<td>192</td>
<td>-123</td>
<td>2251799906842597</td>
<td>4</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>16777153</td>
<td>7</td>
<td>168</td>
<td>-139</td>
<td>2251799906842511</td>
<td>3</td>
<td>147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>16777141</td>
<td>10</td>
<td>240</td>
<td>+69</td>
<td>2251799906842829</td>
<td>4</td>
<td>204</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>16777291</td>
<td>6</td>
<td>144</td>
<td>+69</td>
<td>2251799906842829</td>
<td>3</td>
<td>144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>16777333</td>
<td>8</td>
<td>192</td>
<td>-27</td>
<td>1125899906842597</td>
<td>4</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>33554371</td>
<td>6</td>
<td>150</td>
<td>-51</td>
<td>1125899906842573</td>
<td>3</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>33554371</td>
<td>10</td>
<td>250</td>
<td>-113</td>
<td>1125899906842511</td>
<td>5</td>
<td>250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>33554341</td>
<td>8</td>
<td>200</td>
<td>+159</td>
<td>1125899906842783</td>
<td>3</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>33554467</td>
<td>6</td>
<td>150</td>
<td>+205</td>
<td>1125899906842829</td>
<td>4</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>67108837</td>
<td>8</td>
<td>208</td>
<td>-397</td>
<td>2251799813685269</td>
<td>4</td>
<td>204</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>67108819</td>
<td>9</td>
<td>234</td>
<td>+21</td>
<td>2251799813685269</td>
<td>3</td>
<td>153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>67108879</td>
<td>9</td>
<td>234</td>
<td>+65</td>
<td>2251799813685313</td>
<td>3</td>
<td>153</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>67108933</td>
<td>6</td>
<td>156</td>
<td>+165</td>
<td>2251799813685413</td>
<td>4</td>
<td>204</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>67108933</td>
<td>8</td>
<td>208</td>
<td>-183</td>
<td>450359962736101</td>
<td>3</td>
<td>156</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>536870909</td>
<td>7</td>
<td>210</td>
<td>-315</td>
<td>9007199254740847</td>
<td>4</td>
<td>216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>536870909</td>
<td>8</td>
<td>232</td>
<td>-195</td>
<td>9007199254740847</td>
<td>3</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>536870869</td>
<td>6</td>
<td>174</td>
<td>+5</td>
<td>1801439850948195</td>
<td>3</td>
<td>162</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>536870951</td>
<td>5</td>
<td>145</td>
<td>+159</td>
<td>18014398509482143</td>
<td>3</td>
<td>162</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>536870951</td>
<td>7</td>
<td>203</td>
<td>+341</td>
<td>9007199254740133</td>
<td>3</td>
<td>159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>536870951</td>
<td>8</td>
<td>248</td>
<td>+37</td>
<td>9007199254740333</td>
<td>4</td>
<td>212</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1073741789</td>
<td>7</td>
<td>210</td>
<td>-67</td>
<td>36028797018963913</td>
<td>3</td>
<td>165</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1073741789</td>
<td>8</td>
<td>240</td>
<td>-99</td>
<td>36028797018963869</td>
<td>4</td>
<td>216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1073741789</td>
<td>5</td>
<td>150</td>
<td>+11</td>
<td>36028797018963979</td>
<td>3</td>
<td>165</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1073741831</td>
<td>7</td>
<td>210</td>
<td>+629</td>
<td>36028797018964597</td>
<td>4</td>
<td>220</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1073741909</td>
<td>8</td>
<td>240</td>
<td>-27</td>
<td>7205754037927909</td>
<td>4</td>
<td>224</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>2147483629</td>
<td>6</td>
<td>186</td>
<td>-57</td>
<td>7205754037927879</td>
<td>3</td>
<td>168</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>2147483629</td>
<td>8</td>
<td>248</td>
<td>-147</td>
<td>7205754037927789</td>
<td>4</td>
<td>224</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>2147483629</td>
<td>7</td>
<td>217</td>
<td>+81</td>
<td>7205754037928017</td>
<td>3</td>
<td>168</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------------</td>
<td>---</td>
<td>------</td>
<td>------</td>
<td>------------</td>
<td>---</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>31</td>
<td>+45</td>
<td>2147483693</td>
<td>8</td>
<td>248</td>
<td>56</td>
<td>+177</td>
<td>3</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>+209</td>
<td>2147483857</td>
<td>7</td>
<td>217</td>
<td>56</td>
<td>+201</td>
<td>3</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>+245</td>
<td>2147483893</td>
<td>6</td>
<td>186</td>
<td>57</td>
<td>-13</td>
<td>3</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>-5</td>
<td>4294967291</td>
<td>5</td>
<td>160</td>
<td>57</td>
<td>-195</td>
<td>4</td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>-17</td>
<td>4294967279</td>
<td>7</td>
<td>224</td>
<td>57</td>
<td>-363</td>
<td>4</td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>-99</td>
<td>4294967197</td>
<td>8</td>
<td>256</td>
<td>57</td>
<td>+35</td>
<td>3</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>+15</td>
<td>4294967311</td>
<td>5</td>
<td>160</td>
<td>57</td>
<td>+141</td>
<td>4</td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>+61</td>
<td>4294967357</td>
<td>8</td>
<td>256</td>
<td>57</td>
<td>+189</td>
<td>4</td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>+75</td>
<td>4294967371</td>
<td>6</td>
<td>192</td>
<td>57</td>
<td>+701</td>
<td>4</td>
<td>228</td>
<td></td>
</tr>
</tbody>
</table>
Bibliography


[DMPW98] Erik DeWin, Serge Mister, Bart Preneel, and Michael Wiener. On the Performance of Signature Schemes Based on Elliptic Curves. In *Algo-


