Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography

Daniel V. Bailey[1] and Christof Paar[2]

[1] CS Department, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609 USA
Email: bailey@cs.wpi.edu
[2] ECE and CS Departments, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609 USA
Email: christof@ece.wpi.edu

To appear in Journal of Cryptology

Abstract. This contribution focuses on a class of Galois field used to achieve fast finite field arithmetic which we call an Optimal Extension Field (OEF), first introduced in [3]. 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 extension field multiplication formulas 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 using these methods to construct 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.

Keywords

finite fields, fast arithmetic, binomials, modular reduction, elliptic curves, inversion

1 Introduction

Since their introduction by Victor Miller [19] and Neil Koblitz [13], 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 paper 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 $GF(p^m)$, 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 platform. For example, the International Data Encryption Algorithm [15] and RC5 [23] are designed to use operations that are efficient on desktop-class microprocessors. In addition, the NIST/ANSI Data Encryption Algorithm has been designed so that hardware realizations are particularly efficient [20] [1].

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. The 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 contribution 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 [3], 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. A similar construction is described in independent work by Preda Mihailescu, which describes an exponentiation algorithm for use with OEFs [22]. Much of 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 [3] 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 $\lfloor \log p \rfloor$ up to 57 of the required order for ECCs, as well as statistics on the existence of primes in short intervals.

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. [25] 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 [7].

Schroeppel et.al. [24] 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. [6] presents a detailed implementation of elliptic curve arithmetic on a desktop PC, with a focus on its application to digital signature schemes. 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 authors' previous work in [2] and [3] 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 [11], Knuth demonstrates that the Extended Euclidean Algorithm requires $.843 \log_2(s) + 1.47$ 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 [25], [16], and [24].

Itoh and Tsujii present an algorithm in [8] 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 [7], a version of Itoh and Tsujii's algorithm for inversion when applied to composite Galois fields of characteristic 2 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. [16] 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 [12] 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.

3 Optimal Extension Fields

In the following, we define a class of finite field, which we call an Optimal Extension Field (OEF). 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 $2^n \pm c, \log_2 c \leq \lfloor \frac{1}{2} n \rfloor$.*

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 [17] 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 ω over $GF(p)$, but not $(p-1)/e$; (ii) $p \equiv 1 \bmod 4$ if $m \equiv 0 \bmod 4$.*

An important corollary is given in [10]:

Corollary 1. *Let ω 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 $x^m - 2$.*

A Type II OEF allows for a reduction in the complexity of extension field modular reduction since the multiplications by ω 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.

4 Basic Optimal Extension Field Arithmetic

This section describes the basic method for arithmetic in fields $GF(p^m)$, of which an OEF is a special case. The operation of inversion is the most costly of the four basic operations, and is thus treated separately in Section 5. In Section 6, improved multiplication algorithms are introduced. The material of this section is described in [2] and [3], 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 $A(x) \in GF(p^m)$:

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0, \qquad (1)$$

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 $2m-2$:

$$C'(x) = A(x) \times B(x) = c'_{2m-2}x^{2m-2} + \cdots + c'_1 x + c'_0; \;\; c'_i \in GF(p). \qquad (2)$$

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 4.3 we present an efficient method to calculate the residue $C(x) \equiv C'(x) \bmod P(x), C(x) \in GF(p^m)$. 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 [3] and [18]. 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.

4.3 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 ω and $m-1$ additions, where both of these operations are performed in $GF(p)$.*

A general expression for the reduced polynomial is given by:

$$C(x) \equiv c'_{m-1}x^{m-1} + [\omega c'_{2m-2} + c'_{m-2}]x^{m-2} + \cdots + [\omega c'_m + c'_0] \bmod P(x) \quad (3)$$

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.

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) [8] reduces the problem of extension field inversion to subfield inversion. This reduction relies on the definition of the norm function [17], which states that for any element $\alpha \in GF(p^m), \alpha^{(p^m-1)/(p-1)} \in GF(p)$. In previous reported applications of ITI [7], 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^{32} or 2^{64} leads to tables which are simply too large. Our implementation computes the subfield inverse using the Binary Extended Euclidean Algorithm [21]. 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 $A(x)A^{-1}(x) \equiv 1 \bmod P(x)$. A high-level algorithmic description is given as Algorithm 1. Capital letters denote extension field elements, while lower case letters denote subfield elements.

One method for evaluating the norm of an element is to apply the binary method of exponentiation [11] or one of its improved derivatives [18]. 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 [4], the ith iterate of the Frobenius map $\alpha \rightarrow \alpha^{p^i}$ is also an automorphism. Let us consider the action of an arbitrary iterate i of the Frobenius map on an arbitrary element of $GF(p^m) : A(x) = \sum a_j x^j$, for $a_j \in GF(p)$. We know by Fermat's Little Theorem that $a_j^p \equiv a_j \bmod p$. Thus the a_j coefficients are fixed points of Frobenius map iterates and we can write:

$$A^{p^i}(x) \equiv a_{m-1}x^{(m-1)p^i} + \cdots + a_1 x^{p^i} + a_0 \bmod P(x) \tag{4}$$

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 [12]):

Theorem 3. *Let $P(x)$ be an irreducible polynomial of the form $P(x) = x^m - \omega$ over $GF(p)$, e an integer, $x \in GF(p)[x]$. Then:*

$$x^e \equiv \omega^q x^s \bmod P(x) \tag{5}$$

where $s \equiv e \bmod m$ with $q = \frac{e-s}{m}$.

Proof. First, we observe that $x^m \equiv \omega \bmod P(x)$. Now,

$$x^e = x^{qm+s} \tag{6}$$

where q and s are defined above. Then:

$$x^e = x^{qm} x^s \equiv \omega^q x^s \bmod P(x) \tag{7}$$

□

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

Corollary 2.

$$(x^j)^{p^i} \equiv \omega^q x^j \bmod P(x) \tag{8}$$

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

Proof. Since $P(x)$ is an irreducible binomial, by Theorem 1, $m|(p-1)$, which implies $p = (p-1)+1 \equiv 1 \bmod m$. Thus $s \equiv jp^i \equiv j \bmod m$. □

Note that all $x^{jp^i}, 1 \leq j, i \leq m-1$ in Equation (4) can be precomputed if $P(x)$ is given. Given the above, to compute $(a_j x^j)^{p^i}$ we need only a single subfield multiplication. Thus, we can raise $A(x)$ to the p^i-th power using only $m-1$ subfield multiplications if we make use of Corollary 2 and the precomputed values of $x^{jp}, 1 \leq j \leq m-1$.

For example, 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 1.

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

$x^p \bmod P(x) \equiv 1513477736\ x$	$x^{p^2} \bmod P(x) \equiv 1513477735\ x$
$x^{2p} \bmod P(x) \equiv 1513477735\ x^2$	$x^{2p^2} \bmod P(x) \equiv 634005911\ x^2$
$x^{3p} \bmod P(x) \equiv -1\ x^3$	$x^{3p^2} \bmod P(x) \equiv\ x^3$
$x^{4p} \bmod P(x) \equiv 634005911\ x^4$	$x^{4p^2} \bmod P(x) \equiv 1513477735\ x^4$
$x^{5p} \bmod P(x) \equiv 634005912\ x^5$	$x^{5p^2} \bmod P(x) \equiv 634005911\ x^5$

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:

$$A^{-1}(x) = (A^r)^{-1}(x)A^{r-1}(x), \quad r = \frac{p^m - 1}{p - 1} \tag{9}$$

Algorithm 1 describes the procedure for computing the inverse according to Equation 9. In the following, we will address the individual steps of the algorithm.

Algorithm 1 Optimal Extension Field Inversion

Require: $A(x) \in GF(p^m)^*$
Ensure: $A(x)B(x) \equiv 1 \bmod P(x), B(x) = \sum b_i x^i$
$B(x) \leftarrow A(x)$
Use an addition chain to compute $B(x) \leftarrow B(x)^{r-1}$
$c_0 \leftarrow B(x)A(x)$
$c \leftarrow c_0^{-1}$
$B(x) \leftarrow B(x)c$

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

$$r = p^{m-1} + p^{m-2} + \cdots + p + 1. \tag{10}$$

Thus, we have the p-adic representation $r - 1 = (11\ldots10)_p$. To evaluate our expression in Equation (9), 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 [7] and [8], we see that using such an addition chain constructed from the p-adic representation of $r-1$ requires:

$$\lfloor \log_2(m-1) \rfloor + H_w(m-1) - 1 \text{ general multiplications} + \lfloor \log_2(m-1) \rfloor + H_w(m-1) \text{ Frobenius maps} \quad (11)$$

where H_w is the Hamming weight of the operand.

Given the inversion constants in Table 1, we can now present an addition chain for this field. We compute $A^{r-1}(x)$ as shown in Algorithm 2. In this algorithm, all exponents are understood to be expressed in base p for clarity. This example requires 3 exponentiations to the p-th power, 1 exponentiation to the p^2-th power and 3 general multiplications, as predicted by Equation (11).

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

Require: $A \in GF(p^m)^*$
Ensure: $B \equiv A^{r-1} \bmod P(x)$
$B \leftarrow A^p = A^{(10)}$
$B_0 \leftarrow BA = A^{(11)}$
$B \leftarrow B_0^{p^2} = A^{(1100)}$
$B \leftarrow BB_0 = A^{(1111)}$
$B \leftarrow B^p = A^{(11110)}$
$B \leftarrow BA = A^{(11111)}$
$B \leftarrow B^p = A^{(111110)}$

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 9, 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 $\lfloor \log_2(m-1) \rfloor + H_w(m-1) - 1$ general multiplications and $\lfloor \log_2(m-1) \rfloor + H_w(m-1)$ 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 ω, 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 1, since c is also a constant polynomial, we only need m subfield multiplications.

Each exponentiation to a p^i-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:

$$\begin{aligned} \#\mathrm{SM} = {} & [\lfloor \log_2(m-1) \rfloor + H_w(m-1)](m-1) \\ & + [\lfloor \log_2(m-1) \rfloor + H_w(m-1) - 1](m^2 + m - 1) + 2m \end{aligned} \quad (12)$$

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 $p \approx 2^{61}$ or greater. 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 [12] requires somewhat fewer subfield multiplications. However, a subfield inverse is also required.

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 [11], 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:

$$C'(x) = \sum_{i=0}^{4} c'_i x^i = A(x)B(x) = [a_2b_2]x^4 + [a_2b_1 + a_1b_2]x^3 + [a_2b_0 + a_1b_1 + a_0b_2]x^2 + [a_1b_0 + a_0b_1]x + [a_0b_0]$$

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 additions and subtractions:

$$c'_0 = D_0$$
$$c'_1 = D_3 - D_1 - D_0 = (a_0b_0 + a_0b_1 + a_1b_0 + a_1b_1) - a_1b_1 - a_0b_0$$
$$c'_2 = D_4 - D_2 - D_0 + D_1 = (a_0b_0 + a_2b_0 + a_0b_2 + a_2b_2) - a_2b_2 - a_0b_0 + a_1b_1$$
$$c'_3 = D_5 - D_1 - D_2 = (a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2) - a_1b_1 - a_2b_2$$
$$c'_4 = D_2$$

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

	#MUL	#ADD
schoolbook	9	4
new	6	6 + 7 = 13

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 of this algorithm is given by:

$$T_{SB} = 9T_{MUL} + 4T_{ADD} = (9r + 4)T_{ADD} \tag{13}$$

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

$$T_K = 6T_{MUL} + 13T_{ADD} = (6r + 13)T_{ADD} \tag{14}$$

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 $T_{SB} > T_K$.

$$\begin{aligned} T_{SB} &> T_K \\ (9r + 4)T_{ADD} &= (6r + 13)T_{ADD} \\ r &= 3 \end{aligned}$$

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 3. 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 [11]. 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:

$$C'(x) = E_2(x)\ x^6 + [E_1(x) - E_0(x) - E_2(x)]\ x^3 + E_0(x) \tag{15}$$

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 $E_2(x)\ x^6$, $[E_1(x) - E_0(x) - E_2(x)]\ x^3$, and $E_0(x)$. $E_2(x)\ x^6$ is an expression of the form $\alpha_{10} x^{10} + \alpha_9 x^9 + \alpha_8 x^8 + \alpha_7 x^7 + \alpha_6 x^6$, while $[E_1(x) - E_0(x) - E_2(x)]\ x^3$ has the form $\beta_7 x^7 + \beta_6 x^6 + \beta_5 x^5 + \beta_4 x^4 + \beta_3 x^3$, and we have to compute two subfield additions to obtain the result. A similar situation arises with $[E_1(x) - E_0(x) - E_2(x)]\ x^3$ 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:

	#MUL	#ADD
schoolbook	$6^2 = 36$	$(6-1)^2 = 25$
new	$3 \times 6 = 18$	$3 \times 13 + (3+3) + (5+5) + 4 = 59$

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.

7 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 $m = 3$. 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 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 $m = 6$. 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 scalar 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. 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 [3] 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 2 and 3 shows 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.

Table 2. OEF arithmetic timings in μsec on DEC Alpha microprocessors for the field $GF((2^{61}-1)^3)$ with field polynomial $P(x) = x^3 - 5$

	Alpha 21064, 150 MHz	**Alpha 21164A, 600 MHz**
Schoolbook Multiplication	3.67	0.48
Karatsuba-variant Multiplication	2.77	0.34
$GF(p)$ inverse	8.13	1.81
$GF(p^m)$ inverse	14.6	2.68
Affine EC addition	26.1	4.45
Affine EC doubling	30.5	4.79
Affine point multiplication	6.57 msec	1.06 msec

As a final remark, we observe that for some processors, it may be still be 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 [5]. This hardware improvement dramatically improves the time for an extension field multiplication from 0.34 μsec to 0.18 μsec, 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 μsec to 2.44 μsec. In this case, the ratio of multiplication to

Table 3. OEF arithmetic timings in μsec on Intel microprocessors for the field $GF((2^{31}-1)^6)$ with field polynomial $P(x)=x^6-7$

	Pentium/MMX, 233 MHz
Schoolbook Multiplication	5.82
Karatsuba-variant Multiplication	4.60
$GF(p)$ inverse	4.15
$GF(p^m)$ inverse	25.3
Affine EC addition	44.8
Affine EC doubling	52.4
Affine point multiplication	11.4 msec

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.

8 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.

8.1 Type II OEF Construction Algorithm

Constructing an OEF for a particular application is an essentially straightforward process. Let n, c, m, and ω be positive rational integers. Then we require a prime $p = 2^n \pm c$, an extension degree m, and a constant ω 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 [2] 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.

Algorithm 3 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^{low} and 2^{high}.

```
c ← 1
for log_2 c ≤ ⌊(1/2)n⌋ do
  p ← 2^n − c
  if p is prime then
    factor p − 1
    ord2 ← the order of 2 ∈ GF(p)
    for m ← 2 to 32 do
      if m ∗ n ≥ low and m ∗ n ≤ high then
        BadMValue ← 0
        for each prime divisor d of m do
          if d ∤ ord2 then
            BadMValue ← 1
            Break
          end if
        end for
        if BadMValue = 0 then
          if m ≡ 0 (mod 4) then
            if p ≡ 1 (mod 4) then
              return p, m
            end if
          else
            return p, m
          end if
        end if
      end if
    end for
  end if
  c ← c + 2
end for
```

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 $p-1$.

8.2 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 5, 6, and 7.

8.3 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 [9] shows that

$$\pi(x) - \pi(x - x^{23/42}) > (x^{23/42})/(100 \log x) \tag{16}$$

However, to 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 [14]. 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 $\log_2 c \leq \lfloor \frac{1}{2} n \rfloor$. The results are found in Table 4.

8.4 Tables of Type I and Type II OEFs

The appendix contains tables of OEFs for use in practical applications. Table 8 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 9. 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.

9 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 Karatsuba variant for degree-2 polynomials.

The comments of the anonymous reviewers were greatly appreciated.

References

[1] ANSI X3.92-1981. Data Encryption Algorithm. Technical report, 1981.

[2] D. V. Bailey. Optimal Extension Fields. Major Qualifying Project (Senior Thesis), 1998. Computer Science Department, Worcester Polytechnic Institute, Worcester, MA, USA.

[3] D. V. Bailey and C. Paar. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. In *Crypto '98*, Berlin, 1998. Springer Lecture Notes in Computer Science.

[4] J. R. Bastida. *Field Extensions and Galois Theory*, volume 22 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, Reading, Massachusetts, 1984.

[5] Compaq Computer Corporation, Maynard, Massachusetts. *Compaq 21264 Alpha Microprocessor Hardware Reference Manual*, July 1999.

[6] E. DeWin, S. Mister, B. Preneel, and M. Wiener. On the Performance of Signature Schemes Based on Elliptic Curves. In *Algorithmic Number Theory: Third International Symposium*, pages 252–266, Berlin, 1998. Springer Lecture Notes in Computer Science.

[7] J. Guajardo and C. Paar. Efficient Algorithms for Elliptic Curve Cryptosystems. In *Advances in Cryptology — Crypto '97*, pages 342–356. Springer Lecture Notes in Computer Science, August 1997.

[8] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation*, 78:171–177, 1988.

[9] H. Iwaniec and J. Pintz. Primes in short intervals. *Monatshefte für Mathematik*, 98:115–143, 1984.

[10] D. Jungnickel. *Finite Fields.* B.I.-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1993.

[11] D. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms.* Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981.

[12] T. Kobayashi, H. Morita, K. Kobayashi, and F. Hoshino. Fast Elliptic Curve Algorithm Combining Frobenius Map and Table Reference to Adapt to Higher Characteristic. In *Advances in Cryptography — EUROCRYPT '99.* Springer-Verlag, 1999.

[13] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[14] N. Koblitz. *Algebraic Aspects of Cryptography.* Springer-Verlag, Berlin, first edition, 1998.

[15] X. Lai and J. Massey. A proposal for a new block encryption standard. In *Advances in Cryptography — EUROCRYPT '90*, pages 389–404. Springer-Verlag, 1990.

[16] E. J. Lee, D. S. Kim, and P. J. Lee. Speed-up of F_{p^m} Arithmetic for Elliptic Curve Cryptosystems. In *Proceedings of ICICS '98*, Berlin, 1998. Springer Lecture Notes in Computer Science.

[17] R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications.* Addison-Wesley, Reading, Massachusetts, 1983.

[18] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[19] V. Miller. Uses of elliptic curves in cryptography. In *Lecture Notes in Computer Science 218: Advances in Cryptology — CRYPTO '85*, pages 417–426. Springer-Verlag, Berlin, 1986.

[20] NIST FIPS PUB 46-3. Data Encryption Standard. Technical report, 1977.

[21] G. H. Norton. Extending the Binary GCD Algorithm. In *Algebraic Algorithms and Error-Correcting Codes*, pages 363–372, Berlin, 1986. Springer-Verlag.

[22] Preda Mihailescu. Optimal Galois Field Bases which are not Normal. Recent Results Session, Fast Software Encryption, 1997.

[23] R. L. Rivest. The RC5 Encryption Algorithm. In *Second Fast Software Encryption Workshop*, pages 86–96, Berlin, 1995. Springer Lecture Notes in Computer Science.

[24] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck. Fast key exchange with elliptic curve systems. *Advances in Cryptology — CRYPTO '95*, pages 43–56, 1995.

[25] E. D. Win, A. Bosselaers, S. Vandenberghe, P. D. Gersem, and J. Vandewalle. A fast software implementation for arithmetic operations in $GF(2^n)$. In *Asiacrypt '96.* Springer Lecture Notes in Computer Science, 1996.

A Tables

Table 4. Number of Pseudo-Mersenne Primes, $2^n \pm c, \log_2 c \leq \lfloor (n/2) \rfloor$

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

Table 5. Number of Type II OEFs of order between 2^{130} and $2^{256}, 7 \leq n \leq 10$

m=	14	15	16	17	18	19	20	21	22	23	24	25	26	27	32
n															
7												1	1		
8												3	1	2	3
9			1		1							1		1	
10	1		3	1			1	1		1		3			

Table 6. Number of OEFs of order between 2^{130} and $2^{256}, 11 \leq n \leq 18$

m=	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
n															
11					2	1			3		2	1			1
12				3	1	1		1	3		2			1	
13			2		1	2	1	1	2	1	2	2			
14			4	1	4	2	4	1	8		6				
15		8	1	3	3	1	1		7						
16		19	5	1	4	6	4	4	14						
17	10	14	3	4	4	4	4	3							
18	17	25	7	7	3	5	5								

Table 7. Number of OEFs of order between 2^{130} and $2^{256}, 19 \leq n \leq 55$

m=	3	4	5	6	7	8	9	10	11	12	13
n											
19					9	21	25	4	3	6	4
20					22	39	48	12	14	13	
21					18	35	50	15	11	13	
22				40	41	66	89	33	29		
23				43	35	56	83	31	20		
24				77	72	126	160	48			
25				76	68	124	156	47			
26			183	179	133	219	342				
27			177	139	125	218	286				
28			333	287	259	422	559				
29			329	279	240	404					
30			617	512	479	790					
31			615	529	432	755					
32			1180	946	824	1442					
33		1424	1136	977	766						
34		2813	2180	1857	1561						
35		2636	2126	1755	1483						
36		5154	4149	3359	2967						
37		5095	4139	3429							
38		9871	7911	6599							
39		9749	7771	6380							
40		18864	15179	12499							
41		18533	14656	12286							
42		36074	28817	23951							
43		35215	27905								
44	91499	68735	55042								
45	89336	67300	53918								
46	175514	131656	105347								
47	172251	128937	102966								
48	336066	252095	201375								
49	329827	247247	197553								
50	645703	483609	387502								
51	315731	236628	189774								
52	1241533	931675									
53	1218801	913858									
54	2391808	1792593									
55	2347560	1760093									

Table 8. Type I OEFs for $7 \leq n \leq 61$

n	c	m	mn	ω
7	-1	21	147	3
7	-1	27	189	3
8	1	32	256	2
13	-1	13	169	2
13	-1	10	130	17
13	-1	14	182	17
13	-1	15	195	17
13	-1	18	234	17
16	1	16	256	2
17	-1	9	153	3
17	-1	10	170	3
17	-1	15	255	3
19	-1	7	133	3
19	-1	9	171	3
31	-1	6	186	7
31	-1	7	217	7
61	-1	3	183	37

Table 9: Type II OEFs

n	c	p	m	nm	n	c	p	m	nm
7	+3	131	25	175	33	-49	8589934543	7	231
7	+3	131	26	182	33	-301	8589934291	5	165
8	-5	251	25	200	33	-301	8589934291	6	198
8	-15	241	25	200	33	+29	8589934621	5	165
8	-15	241	27	216	33	+29	8589934621	6	198
8	+1	257	32	256	33	+35	8589934627	7	231
8	+15	271	25	200	34	-113	17179869071	5	170
8	+15	271	27	216	34	-113	17179869071	7	238
9	-3	509	16	144	34	-165	17179869019	6	204
9	+9	521	25	225	34	+153	17179869337	7	238
9	+11	523	18	162	34	+339	17179869523	6	204
9	+11	523	27	243	34	+417	17179869601	5	170
10	-3	1021	16	160	35	-31	34359738337	7	245
10	-3	1021	20	200	35	-61	34359738307	6	210
10	-11	1013	23	230	35	-499	34359737869	4	140
10	+7	1031	25	250	35	+53	34359738421	5	175
10	+27	1051	14	140	35	+53	34359738421	6	210
10	+27	1051	25	250	35	+53	34359738421	7	245
11	-19	2029	13	143	36	-117	68719476619	6	216
11	-19	2029	16	176	36	-189	68719476547	7	252
11	-19	2029	18	198	36	-243	68719476493	4	144
11	+5	2053	16	176	36	+117	68719476853	4	144
11	+5	2053	18	198	36	+117	68719476853	6	216
11	+21	2069	22	242	36	+175	68719476911	7	252
12	-3	4093	16	192	37	-123	137438953349	4	148
12	-3	4093	18	216	37	-141	137438953331	5	185
12	-39	4057	13	156	37	-201	137438953271	5	185
12	+15	4111	15	180	37	+9	137438953481	5	185
12	+37	4133	16	192	37	+29	137438953501	4	148
12	+63	4159	21	252	37	+29	137438953501	5	185
13	-1	8191	13	169	38	-45	274877906899	6	228

13	-13	8179	18	234	38	-107	274877906837	4	152
13	-21	8171	19	247	38	-153	274877906791	5	190
13	+17	8209	19	247	38	+7	274877906951	5	190
13	+27	8219	14	182	38	+13	274877906957	4	152
13	+29	8221	12	156	38	+117	274877907061	6	228
14	-3	16381	12	168	39	-19	549755813869	4	156
14	-3	16381	14	196	39	-67	549755813821	5	195
14	-3	16381	18	252	39	-91	549755813797	6	234
14	+67	16451	14	196	39	+23	549755813911	5	195
14	+69	16453	12	168	39	+45	549755813933	4	156
14	+69	16453	18	252	39	+149	549755814037	6	234
15	-19	32749	12	180	40	-195	1099511627581	4	160
15	-19	32749	16	240	40	-195	1099511627581	5	200
15	-75	32693	11	165	40	-195	1099511627581	6	240
15	+3	32771	10	150	40	+15	1099511627791	5	200
15	+21	32789	14	210	40	+141	1099511627917	4	160
15	+21	32789	16	240	40	+141	1099511627917	6	240
16	-15	65521	9	144	41	-21	2199023255531	5	205
16	-15	65521	13	208	41	-75	2199023255477	4	164
16	-15	65521	15	240	41	-133	2199023255419	6	246
16	+45	65581	10	160	41	+125	2199023255677	4	164
16	+45	65581	12	192	41	+197	2199023255749	6	246
16	+45	65581	15	240	41	+299	2199023255851	5	205
17	-13	131059	9	153	42	-11	4398046511093	4	168
17	-31	131041	13	221	42	-53	4398046511051	5	210
17	-61	131011	15	255	42	-333	4398046510771	5	210
17	+29	131101	9	153	42	+75	4398046511179	6	252
17	+29	131101	12	204	42	+87	4398046511191	5	210
17	+99	131171	13	221	42	+165	4398046511269	4	168
18	-11	262133	13	234	43	-67	8796093022141	4	172
18	-35	262109	11	198	43	-117	8796093022091	5	215
18	-93	262051	9	162	43	+29	8796093022237	4	172
18	+3	262147	9	162	43	+293	8796093022501	5	215
18	+9	262153	11	198	43	+603	8796093022811	5	215
18	+93	262237	13	234	44	-495	17592186043921	5	220
19	-19	524269	8	152	44	-539	17592186043877	4	176
19	-19	524269	12	228	44	-597	17592186043819	3	132
19	-27	524261	10	190	44	+21	17592186044437	3	132
19	+21	524309	8	152	44	+21	17592186044437	4	176
19	+53	524341	12	228	44	+55	17592186044471	5	220
19	+81	524369	13	247	45	-55	35184372088777	3	135
20	-3	1048573	8	160	45	-81	35184372088751	5	225
20	-3	1048573	12	240	45	-139	35184372088693	4	180
20	-5	1048571	10	200	45	+59	35184372088891	5	225
20	+13	1048589	8	160	45	+165	35184372088997	4	180
20	+33	1048609	11	220	45	+179	35184372089011	3	135
20	+57	1048633	9	180	46	-21	70368744177643	3	138
21	-19	2097133	8	168	46	-333	70368744177331	5	230
21	-19	2097133	12	252	46	-635	70368744177029	4	184
21	-61	2097091	10	210	46	+127	70368744177791	5	230
21	+59	2097211	10	210	46	+165	70368744177829	3	138
21	+77	2097229	8	168	46	+165	70368744177829	4	184
21	+77	2097229	12	252	47	-115	140737488355213	4	188
22	-3	4194301	9	198	47	-127	140737488355201	5	235
22	-27	4194277	8	176	47	-541	140737488354787	3	141
22	-57	4194247	7	154	47	+5	140737488355333	3	141
22	+15	4194319	9	198	47	+5	140737488355333	4	188
22	+85	4194389	8	176	47	+273	140737488355601	5	235
22	+85	4194389	11	242	48	-59	281474976710597	4	192
23	-27	8388581	10	230	48	-93	281474976710563	3	144
23	-61	8388547	9	207	48	-165	281474976710491	5	240
23	-157	8388451	7	161	48	+61	281474976710717	4	192
23	+11	8388619	7	161	48	+75	281474976710731	3	144
23	+11	8388619	9	207	48	+235	281474976710891	5	240
23	+15	8388623	11	253	49	-81	562949953421231	5	245
24	-3	16777213	8	192	49	-123	562949953421189	4	196
24	-63	16777153	7	168	49	-139	562949953421173	3	147
24	-75	16777141	10	240	49	+69	562949953421381	4	196
24	+75	16777291	6	144	49	+69	562949953421381	5	245
24	+75	16777291	10	240	49	+191	562949953421503	3	147

24	+117	16777333	8	192	50	-27	1125899906842597	4	200
25	-61	33554371	6	150	50	-51	1125899906842573	3	150
25	-61	33554371	10	250	50	-113	1125899906842511	5	250
25	-91	33554341	8	200	50	+159	1125899906842783	3	150
25	+35	33554467	6	150	50	+205	1125899906842829	4	200
25	+69	33554501	8	200	50	+337	1125899906842961	5	250
25	+69	33554501	10	250	51	-139	2251799813685109	4	204
26	-27	67108837	8	208	51	-237	2251799813685011	5	255
26	-45	67108819	6	156	51	-397	2251799813684851	3	153
26	-45	67108819	9	234	51	+21	2251799813685269	4	204
26	+15	67108879	9	234	51	+65	2251799813685313	3	153
26	+69	67108933	6	156	51	+165	2251799813685413	4	204
26	+69	67108933	8	208	52	-183	4503599627370313	3	156
27	-79	134217649	9	243	52	-395	4503599627370101	4	208
27	-187	134217541	6	162	52	-635	4503599627369861	4	208
27	-231	134217497	7	189	52	+21	4503599627370517	3	156
27	+45	134217773	8	216	52	+21	4503599627370517	4	208
27	+53	134217781	6	162	52	+37	4503599627370533	4	208
27	+53	134217781	9	243	53	-145	9007199254740847	3	159
28	-57	268435399	7	196	53	-315	9007199254740677	4	212
28	-165	268435291	6	168	53	-339	9007199254740653	4	212
28	-165	268435291	9	252	53	+5	9007199254740997	4	212
28	+3	268435459	6	168	53	+41	9007199254741033	3	159
28	+3	268435459	9	252	53	+341	9007199254741333	4	212
28	+37	268435493	8	224	54	-33	18014398509481951	3	162
29	-3	536870909	7	203	54	-131	18014398509481853	4	216
29	-3	536870909	8	232	54	-195	18014398509481789	4	216
29	-43	536870869	6	174	54	+159	18014398509482143	3	162
29	+39	536870951	5	145	54	+373	18014398509482357	4	216
29	+39	536870951	7	203	54	+477	18014398509482461	4	216
29	+117	536871029	8	232	55	-55	36028797018963913	3	165
30	-35	1073741789	7	210	55	-67	36028797018963901	4	220
30	-35	1073741789	8	240	55	-99	36028797018963869	4	220
30	-83	1073741741	5	150	55	+11	36028797018963979	3	165
30	+7	1073741831	5	150	55	+461	36028797018964429	4	220
30	+7	1073741831	7	210	55	+629	36028797018964597	4	220
30	+85	1073741909	8	240	56	-27	72057594037927909	4	224
31	-19	2147483629	6	186	56	-57	72057594037927879	3	168
31	-19	2147483629	8	248	56	-147	72057594037927789	4	224
31	-85	2147483563	7	217	56	+81	72057594037928017	3	168
31	+45	2147483693	8	248	56	+177	72057594037928113	3	168
31	+209	2147483857	7	217	56	+201	72057594037928137	3	168
31	+245	2147483893	6	186	57	-13	144115188075855859	3	171
32	-5	4294967291	5	160	57	-195	144115188075855677	4	228
32	-17	4294967279	7	224	57	-363	144115188075855509	4	228
32	-99	4294967197	8	256	57	+35	144115188075855907	3	171
32	+15	4294967311	5	160	57	+141	144115188075856013	4	228
32	+61	4294967357	8	256	57	+189	144115188075856061	4	228
32	+75	4294967371	6	192	57	+701	144115188075856573	4	228