Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms

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

[1] Computer Science Department, Worcester Polytechnic Institute, Worcester, MA 01609 USA.
Email: bailey@cs.wpi.edu

[2] ECE Department, Worcester Polytechnic Institute, Worcester, MA 01609 USA.
Email: christof@ece.wpi.edu

Abstract. This contribution introduces a class of Galois field used to achieve fast finite field arithmetic which we call an Optimal Extension Field (OEF). This approach is well suited for implementation of public-key cryptosystems based on elliptic and hyperelliptic curves. Whereas previous reported optimizations focus on finite fields of the form $GF(p)$ and $GF(2^m)$, an OEF is the class of fields $GF(p^m)$, for p a prime of special form and m a positive integer. Modern RISC workstation processors are optimized to perform integer arithmetic on integers of size up to the word size of the processor. Our construction employs well-known techniques for fast finite field arithmetic which fully exploit the fast integer arithmetic found on these processors. In this paper, we describe our methods to perform the arithmetic in an OEF and the methods to construct OEFs. We provide a list of OEFs tailored for processors with 8, 16, 32, and 64 bit word sizes. We report on our application of this approach to construction of elliptic curve cryptosystems and demonstrate a substantial performance improvement over all previous reported software implementations of Galois field arithmetic for elliptic curves.

Keywords

finite fields, fast arithmetic, pseudo-Mersenne primes, Optimal Extension Fields, OEF, binomials, modular reduction, hyperelliptic curves, elliptic curves, cryptographic implementation

1 Introduction and Motivation

Arithmetic in finite fields is an integral part of many public-key algorithms, including those based on the discrete logarithm problem in finite fields, elliptic curve based schemes, and emerging applications of hyperelliptic curves. Our ability to quickly perform arithmetic in the underlying finite field determines the performance of these schemes. Finite fields are identified with the notation $GF(p^m)$, where p is a prime and m is a positive integer. Essentially all previous work in this area has focused on two types of finite fields: GF(p^m) with $m = 1$,

p a prime; and $p = 2$, m some positive integer. In this paper, we consider the use of extension fields of large characteristic with the characteristic p a prime of special form, m some positive integer.

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 signals "0" and "1." However, $p = 2$ does not offer the same computational advantages in a software implementation, since modern workstation 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 $cm\frac{m}{w}$ 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^{150}. 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 accomodated, and reduction modulo p must be performed with operands that span multiple machine words.

In this paper we define a special class of choices of p and m and show that they can yield considerable computational advantages. Our primary motivation in what follows is to exploit the very high performance that modern RISC processors offer for integer arithmetic on single words, which alleviate many of the difficulties found with $GF(p)$ and $GF(2^m)$. Our focus in the present paper is on elliptic curve cryptosystems as introduced in [7] and [13]. However, the arithmetic introduced here can also be applied to hyperelliptic curve public-key systems as introduced in [8].

2 Our New Approach

Our new approach is based on the observation that several well-known optimizations exist for software implementation of finite field arithmetic and that when they are used in conjunction they yield significant performance gains for implementation of elliptic and hyperelliptic curve cryptosystems. To optimize arithmetic in $GF(p^m)$ we stipulate the following properties on the choice of p and m:

1. Choose p to be less than but close to the word size of the processor so that all subfield operations take advantage of the processor's fast integer arithmetic.
2. Choose p to be a *pseudo-Mersenne prime*, that is, of the form $2^n \pm c$ for some $\log_2 c \leq \frac{1}{2}n$ to allow for efficient subfield modular reduction.
3. Choose m so that we have an irreducible binomial $x^m - \omega$ for efficient extension field modular reduction. The extension degree m can be small if the processor word size allows for large values of p.

A field that offers these arithmetic optimizations we call an Optimal Extension Field (OEF). For a formal definition of OEF, see Section 7. We demonstrate that these optimizations can yield a substantial performance improvement over previous results as in [4,16,17,3]. As an example, when a modern RISC workstation with a 64-bit architecture such as the DEC Alpha family is our target platform, we would choose a p near 2^{64}. This approach has the advantage of fully exploiting the RISC CPU's ability to quickly perform 64 bit $\times$ 64 bit integer multiplication, thus performing a subfield multiplication with a single multiply instruction followed by a modular reduction. Due to the special form of p, we may perform this reduction without executing a traditional division algorithm. In order to gain this sort of computational advantage for public-key algorithms with field orders of more than 2^{64}, we use a field extension m of moderate degree. For example, the choice of $p = 2^{61} - 1$ together with an extension degree of $m = 3$ would result in an OEF with order approximately 2^{183}. Such a field is desirable in the construction of cryptosystems based on the discrete logarithm problem in elliptic curve groups. In this paper we demonstrate efficient methods to construct such fields, strategies for fast arithmetic in an OEF, and implementation results for an application of this work to elliptic curve cryptosystems.

3 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 involves the use of subfields of characteristic two. A paper due to DeWin et al. [17] 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 [3].

Schroeppel et al. [16] report an implementation of an elliptic curve analogue of Diffie-Hellman key exchange over $GF(2^{155})$ with an irreducible trinomial as the field polynomial. The arithmetic is based on a polynomial basis representation of the field elements. Elements of the field are each stored in three 64-bit registers.

Much optimization work has been done in selection of Optimal Normal Bases (ONB) to speed computations in $GF(2^m)$. Draft standards such as [18] [19], and [9] suggest use of ONB for elliptic curve systems.

Others have investigated use of pseudo-Mersenne primes to construct Galois fields $GF(p)$ in connection with elliptic curve cryptography as found in [2], [14] and some patents have been issued on their use.

Unlike the methods in [17,3] which use Cayley tables to implement subfield arithmetic, our approach requires no additional memory and is therefore attractive in memory-constrained applications. In addition, our system is faster in real-world tests as described in Section 8.

4 Optimal Extension Field Arithmetic

This section describes the basic construction for arithmetic in fields $GF(p^m)$, of which an OEF is a special case. The subfield is $GF(p)$ and the extension degree is denoted by m, so that the field can be denoted by $GF(p^m)$. This field 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 $A \in GF(p^m)$:

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

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.

All arithmetic operations are performed modulo the field polynomial. The choice of field polynomial determines the complexity of the operations required to perform the modular reduction. In this paper, we will only be concerned with the operations of addition, multiplication, and squaring.

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 p once from the intermediate result. Previous implementations in $GF(2^n)$ offer a slight computational advantage since addition or subtraction is simply an XOR that does not require modular reduction. When compared to the addition operation in $GF(p)$ for large p, we observe that an OEF does not require carry between computer words in computing a sum while $GF(p)$ does. This property results in a modest performance gain over $GF(p)$.

Algorithm 1 Optimal Extension Field Addition

```
Require: $A(x) = a_{m-1}x^{m-1} + \ldots + a_1 x + a_0, B(x) = b_{m-1}x^{m-1} + \ldots + b_1 x + b_0, A(x), B(x) \in GF(p^m)$.
Ensure: $A(x) + B(x) \equiv C(x) \in GF(p^m)$
  for $i \leftarrow 0$ to $m-1$ do
    $c_i = a_i + b_i$
    if $c_i \geq p$ then
      $c_i \leftarrow c_i - p$
    end if
  end for
```

4.2 Multiplication

Multiplication is 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 $2m-2$:

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

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 $GF(p)$.

Since field multiplication is the time critical task in many public-key algorithms this paper will deal extensively with fast multiplication methods, and later sections are devoted to aspects of this operation. In Section 4.4 we present an efficient method to calculate the residue $C(x) \equiv C'(x) \bmod P(x), C(x) \in GF(p^m)$. Section 5 gives a method to quickly perform the coefficient multiplication in $GF(p)$.

4.3 Squaring

Squaring may be implemented using the method for general multiplication outlined above. However, we observe that squaring a field element affords some additional computational efficiencies. For example, consider the field element $A(x) = a_2x^2 + a_1x + a_0, A(x) \in GF(p^3)$. We compute the square of $A(x)$ and obtain:

$$(a_2x^2 + a_1x + a_0)^2 = a_2^2x^4 + 2a_2a_1x^3 + [2a_2a_0 + a_1^2]x^2 + 2a_1a_0x + a_0^2 \qquad (3)$$

Multiplication by two may be implemented in a computer as a left shift operation by one bit. On many computer architectures, a left shift is faster than an explicit integer multiplication. Thus instead of requiring m^2 multiplications, we need only $m(m+1)/2$ explicit multiplications. The remainder may be performed as shifts.

4.4 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. We observe that we must perform subfield multiplications to implement the reduction, proportional to the number of terms in the field polynomial. However, if we construct a field polynomial with low coefficient weight, the modular reduction will require fewer subfield multiplications.

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:

$$P(x) = x^m - \omega \qquad (4)$$

The use of irreducible binomials as field polynomials yields major computational advantages as will be shown below. Observe that irreducible binomials do not exist over $GF(2)$.

In Section 6, we will demonstrate that such irreducible binomials can be constructed. Once such a binomial has been determined, modular reduction can be performed with the following complexity:

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

Proof. By assumption, $C'(x)$ has the form:

$$C'(x) = c'_{2m-2}x^{2m-2} + \ldots + c'_m x^m + c'_{m-1}x^{m-1} + \ldots + c'_1 x + c'_0 \tag{5}$$

Only the terms $c'_{m+i}x^{m+i}, i \geq 0$, must be reduced modulo P(x). We observe that:

$$c'_{m+i}x^{m+i} \equiv \omega c'_{m+i}x^i \bmod P(x); \quad i = 0, 1, \ldots, m-2 \tag{6}$$

Since the degree of $C'(x) \leq 2m-2$, we require at most $m-1$ multiplications by ω and $m-1$ additions to combine the reduced terms. □

A general expression for the reduced polynomial is given by:

$$C(x) \equiv c'_{m-1}x^{m-1} + [\omega c'_{2m-2} + c'_{m-1}]x^{m-2} + \cdots + [\omega c'_{m+1} + c'_1]x + [\omega c'_m + c'_0] \bmod P(x) \tag{7}$$

As an optimization, when possible we choose those fields with an irreducible binomial $x^m - 2$, allowing us implement the multiplications as shifts. OEFs that offer this optimization are known as Type II. A method to search for these Type II OEFs is given in Section 7.

5 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, which are previously reported in the literature, see for example [12]. For actual implementation of OEF arithmetic, optimization of subfield arithmetic is critical to performance, so we include these remarks in this paper for completeness.

We recall that multiplication of two elements $a, b \in GF(p)$ is performed by $a \times b \equiv c \bmod p$. 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 field characteristic 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 [12]. Many methods such as Barrett Reduction exist which offer computational advantages over traditional long division of 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.

It is well known that fast modular reduction is possible with moduli of the form $2^n \pm c$, where c is a "small" integer. Integers of this form allow modular reduction without division. We present a form of such a modular reduction algorithm, adapted from [12]. In this paper we consider only primes of the form $2^n - c$, although a trivial change to the following algorithm allows the use of primes $2^n + c$. The operators << and >> are taken to mean "left shift" and "right shift" respectively.

Algorithm 2 Fast Subfield Modular Reduction

```
Require: $p = 2^n - c, \log_2 c \leq \frac{1}{2}n$, $x < p^2$ is the integer to reduce
Ensure: $r \equiv x \bmod p$
  $q_0 \leftarrow x >> n$
  $r_0 \leftarrow x - q_0 2^n$
  $r \leftarrow r_0$
  $i \leftarrow 0$
  while $q_i > 0$ do
    $q_{i+1} \leftarrow q_i c >> n$
    $r_{i+1} \leftarrow q_i c - (q_{i+1} >> n)$
    $i \leftarrow i + 1$
    $r \leftarrow r + r_i$
  end while
  while $r \geq p$ do
    $r \leftarrow r - p$
  end while
```

Under these conditions, the algorithm terminates after a maximum of two iterations of the while loop, so we require at the most two multiplications by c, six shifts by n, and six additions and subtractions. 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 μsec for a multiplication in $GF(p^m)$. When we perform modular reduction using this algorithm, we require only 1.35 μsec, a fivefold savings.

If $c = 1$, this algorithm executes the first while loop only once. In addition, no multiplications are required for the modular reduction and the entire operation may be performed with 2 shifts and 2 adds if the intermediate result is contained in a single word, 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.

6 Irreducible Binomials

In Section 4.4 we showed that irreducible binomials allow modular reduction with low complexity. The following theorem from [11] describes the cases when an irreducible binomial exists:

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

An important corollary is given in [5]:

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 of order $(p-1)m$ over $GF(p)$.*

We present the following new corollary which follows directly from the above, since $p - 1$ is always an even number:

Corollary 2. *Let ω be a primitive element for $GF(p)$. Then $x^2 - \omega$ is irreducible over $GF(p)$.*

An extension degree of 2 is especially attractive for the implementation of cryptosystems based on hyperelliptic curves, since the field orders required are in the range 40-120 bits [15]. On a 32-bit or 64-bit architecture, the use of an OEF with $m = 2$ can form the basis for a very fast hyperelliptic curve implementation.

Irreducible binomials do not exist over $GF(2)$. Thus, previous approaches to this problem focusing on $GF(2^m)$ have been unable to use binomials. For an OEF, however, we require p and m such that an irreducible binomial can be constructed. An algorithm to find such choices of p and m is described in Section 7.

7 Optimal Extension Fields

In the following, we define a new class of finite field, which we call an Optimal Extension Field (OEF). To simplify matters, we introduce a new name for a class of prime numbers:

Definition 1. *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 $GF(p)$.*

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, as described in Section 5.

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

A Type II OEF allows for speedups in extension field modular reduction since the multiplications by ω in Theorem 1 can be implemented using shifts instead of explicit multiplications.

The choice of m depends on the factorization of $p - 1$ due to Theorem 2 and Corollary 1. In the following we describe an efficient construction method for OEFs. From a very high level, this method consists of three main steps: We choose a pseudo-Mersenne prime p first, then factor $p-1$, and then finally select an extension degree m. Since $p \leq 2^{64}$ due to current common processor word lengths, it is sufficient to use trial division to quickly factor $p-1$. This procedure does not exhaustively list all OEFs, rather it is designed to quickly locate a Type II OEF for a desired field order and machine word size. Further, this procedure considers only those primes $2^n - c$, although a prime $2^n + c$ is a valid choice for OEFs.

A high-level outline of our field construction algorithm, which is based on Corollary 1 is given as Algorithm 3.

There are other possible values for the order of ω that would lead to a greater number of fields that meet our criteria according to Theorem 2. However, the inclusion of these additional fields comes at the expense of an increase in complexity of our algorithm.

We found that even with the restriction of ω a primitive element on our search for fields, there are still enough Type II OEFs to construct fields for any application. Our computational experiments indicate that for $n = 32$ and $n = 64$ there are hundreds of fields that satisfy these criteria. Tables of OEFs for all $7 \leq n \leq 63$ are found in [1].

For example, suppose we wish to construct a field for use on a modern workstation with 64-bit integer arithmetic for use in an elliptic curve key exchange algorithm. We set $n \leftarrow 63$, $c \leftarrow 1$, $low \leftarrow 120$, $high \leftarrow 260$. Then we apply a probabilitstic primality test for the integers $2^n - c$, incrementing c by 2 until we locate a prime. Using this method, we discover that $p = 2^{63} - 259$ is prime. At this point, we factor $p - 1$ using trial division to obtain the factorization $2^2 \times 3^2 \times 7 \times 107 \times 342062455008707 = 9223372036854775548$. Given

Algorithm 3 Fast Type II Optimal Extension Field Construction Procedure

```
Require: $n$ bitlength of desired $p$; $low$, $high$ bounds on bit length of field order
Ensure: $p$, $m$ define a Type II Optimal Extension Field with field order between $2^{low}$
  and $2^{high}$.
  $c \leftarrow 1$
  for $c \leftarrow 1$ to $\frac{1}{2}n$ do
    $p \leftarrow 2^n - c$
    if p is prime then
      factor $p - 1$
      if 2 is primitive in $GF(p)$ then
        for $m \leftarrow low$ to $high$ do
          if $m|(p-1)$ then
            return $p, m$
          end if
        end for
      end if
    end if
  end for
```

this factorization we can easily perform a primitivity check and find that 2 is a primitive element. Algorithms to compute the order of a group element are well known, see [12]. It remains only to select an extension degree. By trial division, we observe that 2, 3, and 4 all divide $p - 1$ and thus $x^2 - 2$, $x^3 - 2$, and $x^4 - 2$ are all irreducible binomials over $GF(p)$. These binomials yield the fields $GF((2^{63} - 259)^2), GF((2^{63} - 259)^3)$, and $GF((2^{63} - 259)^4)$, respectively. The approximate orders of these fields are $2^{126}, 2^{189}$, and 2^{252}, respectively.

8 Implementation Results

8.1 Application to Elliptic Curve Cryptography

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 previous reported in the literature.

We implemented our algorithms on a 500 MHz DEC Alpha workstation in optimized C, only resorting to assembly to perform 64 bit $\times$ 64 bit multiplications, since these operations are not directly supported by Digital's C compiler. We executed the Type II OEF construction procedure to find Type II OEFs for the word sizes 8, 16, 32, and 63. These word sizes are representative of the CPUs found in typical applications, although OEFs may be constructed for any arbitrary word size. For each word size we attempted to construct an OEF with approximately 160, 190, and 240 bit length, as such fields are suggested for the implementation of practical elliptic curve systems [18] [19]. The OEF construction algorithm from Section 7 found the fields shown in Table 1 with the exception of fields for an 8-bit word size, and the field with $p = 2^{61} - 1$. In both cases, $\omega = 2$ is not primitive in $GF(p)$. We constructed these cases using Theorem 2. In order to obtain accurate timings, we executed field multiplication in $GF(p^m)$ one million times, observed the execution time, and computed the average. Table 1 shows the result of our field construction and subsequent timing measurements.

For each of our example OEFs, Table 1 lists nm, which is the approximate bit length of the field order, the prime p, the irreducible binomial, and the time in microseconds to perform the $GF(p^m)$ multiplication. In addition, we provide estimated time in milliseconds for a single elliptic curve group operation, elliptic curve point doubling, and estimated time for a full point multiplication, using the following assumptions.

The elliptic curve addition operation in projective coordinates may be performed with 15 multiplications in $GF(p^m)$, while doubling requires 12 multiplications [10]. Then we estimate the time required for an elliptic curve point multiplication as required in the elliptic curve analogue of Diffie-Hellman key exchange, assuming an implementation using the k-ary window method [6] with $k = 4$ to speed the repeated doubling and add operations. Note that in the estimations we ignored time required to perform additions in the finite field, but also did not employ better point multiplication algorithms such as signed-digit methods [10] and addition chains.

Most fields included here are Type II with the exception of the 8-bit fields and the field $GF((2^{61}-1)^3)$, which is Type I. This accounts for its very high performance: a field multiplication is performed in 0.52 microseconds. When applied to elliptic curve cryptosystems, this field results in a very fast implementation, requiring only 1.58 milliseconds for a full point multiplication.

8.2 Comparison

We also compared our implementation with three previously reported approaches. For ease in comparison, we report our timing results as measured on a 150 MHz DEC Alpha. Results are found in Table 2.

For each implementation, we give the timing for a field multiplication. It can be seen that our OEF $GF((2^{61} - 1)^3)$ yields field multiplication speeds which are more than twice as fast as the best previously reported approach. This is true even though our field has an order of 2^{183}, whereas the field in [16] has an order of 2^{155} and their workstation has a slightly higher clock rate.

Table 1. OEF arithmetic timings on a 500 MHz DEC Alpha

nm	p $= 2^n - c$	binomial $x^m - \omega$	GF mult (μsec)	EC add (μsec) (est.)	EC double (μsec) (est.)	αP (msec) (est.)
160	$2^8 - 15$	$x^{20} - 7$	48.3	725	580	130
200	$2^8 - 5$	$x^{25} - 6$	70.1	1050	841	231
240	$2^8 - 15$	$x^{30} - 7$	100	1500	1200	392
160	$2^{16} - 165$	$x^{10} - 2$	13.8	207	166	37.1
192	$2^{16} - 243$	$x^{12} - 2$	16.9	253	203	53.7
240	$2^{16} - 165$	$x^{15} - 2$	28.0	420	336	110
160	$2^{32} - 5$	$x^5 - 2$	1.35	20	16.2	3.62
192	$2^{32} - 387$	$x^6 - 2$	2.13	32	26	6.85
224	$2^{32} - 1053$	$x^7 - 2$	3.00	45	36	11.0
183	$2^{61} - 1$	$x^3 - 37$	0.52	7.8	6.24	1.58
189	$2^{63} - 259$	$x^3 - 2$	0.87	13	10	2.64
252	$2^{63} - 259$	$x^4 - 2$	1.49	22	18	6.12

Table 2. Comparison of arithmetic performance

Method	Field Size	Field Type	Platform	*GF* mult (μsec)
DeWin [17]	176 bits	$GF((2^n)^m)$	133 MHz Pentium	62.7
Guajardo-Paar [3]	176 bits	$GF((2^n)^m)$	175 MHz DEC Alpha	38.6
Schroeppel [16]	155 bits	$GF(2^m)$	175 MHz DEC Alpha	7.1
OEF	183 bits	$GF(p^m)$	150 MHz DEC Alpha	3.3

9 Conclusion

In this paper we have introduced a class of finite fields, known as Optimal Extension Fields, which take advantage of well-known optimizations for finite field arithmetic on microprocessors commonly found in workstations. OEFs are especially attractive for use in elliptic curve and hyperelliptic curve systems. The arithmetic speedups are due to the inherent properties of an OEF. An OEF may be constructed with a subfield close to the size of the host CPU. The field characteristic of an OEF is a pseudo-Mersenne prime, that is, of the form $2^n \pm c$

for small c, allowing fast subfield modular reduction. The extension degree of an OEF always allows for an irreducible binomial. Finally, the field polynomial of an OEF is chosen to have a constant term equal to 2. In real-world demonstrations, we have shown that an OEF yields a considerable speed advantage over previous software implementations of Galois field arithmetic for elliptic curve cryptography.

References

1. Daniel V. Bailey. Optimal extension fields. Major Qualifying Project (Senior Thesis), 1998. Computer Science Department, Worcester Polytechnic Institute, Worcester, MA, USA.
2. Richard E. Crandall. Method and apparatus for public key exchange in a cryptographic system. *US Patent 5463690*, 1995.
3. Jorge Guajardo and Christof Paar. Efficient algorithms for elliptic curve cryptosystems. In *Advances in Cryptology — Crypto '97*, pages 342–356. Springer Lecture Notes in Computer Science, August 1997.
4. G. Harper, A. Menezes, and S. Vanstone. Public-key cryptosystems with very small key lengths. In *Advances in Cryptology — EUROCRYPT '92*, pages 163–173, May 1992.
5. D. Jungnickel. *Finite Fields*. B.I.-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1993.
6. D.E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981.
7. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.
8. N. Koblitz. Hyperelliptic cryptosystems. *Journal of Cryptology*, 1(3):129–150, 1989.
9. J. Koeller, A. Menezes, M. Qu, and S. Vanstone. Elliptic Curve Systems. Draft 8, IEEE P1363 Standard for RSA, Diffie-Hellman and Related Public-Key Cryptography, May 1996. working document.
10. Kenji Koyama and Yukio Tsuruoka. Speeding up elliptic cryptosystems by using a signed binary window method. In *Crypto '92*. Springer Lecture Notes in Computer Science, 1992.
11. R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, Reading, Massachusetts, 1983.
12. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
13. 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.
14. Atsuko Miyaji and Makoto Tatebayashi. Method for generating and verifying electronic signatures and privacy communication using elliptic curves. *US Patent 5442707*, 1995.
15. S. Paulus. *Ein Algorithmus zur Berechnung der Klassengruppe quadratischer Ordnungen über Hauptidealringen*. PhD thesis, Institute for Experimental Mathematics, University of Essen, Essen, Germany, June 1996.
16. 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.

17. E. De Win, A. Bosselaers, S. Vandenberghe, P. De Gersem, and J. Vandewalle. A fast software implementation for arithmetic operations in $GF(2^n)$. In *Asiacrypt '96*. Springer Lecture Notes in Computer Science, 1996.
18. ANSI X9.62-199x. The Elliptic Curve Digital Signature Algorithm. Draft, January 1998. working document.
19. ANSI X9.63-199x. Elliptic Curve Key Agreement and Key Transport Protocols. Draft, January 1998. working document.