

Software Multiplication using Gaussian Normal Bases

Ricardo Dahab*, Darrel Hankerson†, Fei Hu‡,
Men Long§, Julio López¶, and Alfred Menezes||

Abstract

Fast algorithms for multiplication in finite fields are required for several cryptographic applications, in particular for implementing elliptic curve operations over binary fields \mathbb{F}_{2^m} . In this paper we present new software algorithms for efficient multiplication over \mathbb{F}_{2^m} that use a Gaussian normal basis representation. Two approaches are presented, direct normal basis multiplication, and a method that exploits a mapping to a ring where fast polynomial-based techniques can be employed. Our analysis including experimental results on an Intel Pentium family processor shows that the new algorithms are faster and can use memory more efficiently than previous methods. Despite significant improvements, we conclude that the penalty in multiplication is still sufficiently large to discourage the use of normal bases in software implementations of elliptic curve systems.

Key words Multiplication in \mathbb{F}_{2^m} , Gaussian normal basis, elliptic curve cryptography.

1 Introduction

Efficient implementation of field arithmetic is fundamental to the performance of common cryptographic mechanisms such as those based on elliptic curves. Of particular importance is the multiplication operation since it is a major building block in cryptographic applications. The speed of algorithms for multiplication depends greatly on the particular basis used to represent field elements. The two most common choices of bases for \mathbb{F}_{2^m} are normal and polynomial. For example, ANSI X9.62 [2] and the NIST elliptic curves over binary fields specified in FIPS 186-2 [7] allow Gaussian normal bases and polynomial bases, and hence it is of interest to compare the performance of field arithmetic in each basis.

Most papers on normal basis arithmetic have focused on hardware implementation. Our focus is software implementation. Previous work on software multiplication for normal basis representations include Ning and Yin [25] (for optimal normal bases and the development of a precomputation technique for speeding up the multiplication), Reyhani-Masoleh and Hasan [27] (for Gaussian normal bases), and recently Granger, Page and Stam [11] (for normal bases in \mathbb{F}_{3^m}). All methods known for binary field multiplication in software that use a normal basis representation are slow in comparison to the best methods for multiplication using a polynomial basis [21]. However, the improvements in normal basis multiplication in [25] (and natural generalizations) and a method based on results concerning Gauss periods [9, 33] suggest that the penalty may be much smaller than previously reported. We are interested in more precise estimates of the actual costs for field multiplication in a given basis, and the possibility of using multiple bases in a larger framework such as methods based on elliptic curves.

In this paper, we describe two new techniques for efficient software implementation of binary field multiplication in \mathbb{F}_{2^m} for Gaussian normal basis representations. The first of these approaches uses a direct method for a normal basis and is generally faster and requires less storage than previously published results. It is based, in part, on a technique to reduce the amount of evaluation-phase shifting normally associated

*Institute of Computing, University of Campinas, Brazil, rdahab@ic.unicamp.br

†Dept. of Mathematics, Auburn University, USA, hankedr@auburn.edu

‡Dept. of Electrical and Computer Engineering, Auburn University, USA, hufei01@auburn.edu

§Dept. of Electrical and Computer Engineering, Auburn University, USA, longmen@auburn.edu

¶Institute of Computing, University of Campinas, Brazil, jlopez@ic.unicamp.br

||Dept. of Combinatorics and Optimization, University of Waterloo, Canada, ajmenez@uwaterloo.ca

with algorithms having less precomputation. The second approach exploits a relatively fast mapping to an associated (but typically larger) ring where efficient polynomial-based multiplication methods can be employed.

1.1 Related works

There is a sizable catalog of papers on methods for normal basis arithmetic targeted at hardware implementations. In this section, we briefly survey recent work that is particularly applicable to software implementations.

Direct multiplication in a normal basis

Ning and Yin [25] focus on optimal normal bases, although they also give a general method based on their improvements. The results were faster than the vector-level algorithm in Reyhani-Masoleh and Hasan [27], although the faster algorithm has significantly larger data-dependent storage requirements.

During the development of our paper, independent work from Reyhani-Masoleh [26] and Fan and Dai [6] appeared. These papers and our paper contribute algorithms addressing normal basis arithmetic in software implementations; however, the specific approaches and conclusions are surprisingly different.

Reyhani-Masoleh [26] gives improvements and extensions to Ning and Yin that are motivated by the same considerations that led to our algorithms in §§2 and 3. Our fastest algorithm (Algorithm 7) using normal basis arithmetic can be regarded as a further extension of these ideas, and gives a significant improvement to the methods proposed by Reyhani-Masoleh. In addition, we show that the timings in [26] are far too pessimistic for algorithms based on these methods.

Similarly, Fan and Dai [6] present algorithms that can be regarded as enhancements to the approaches of Reyhani-Masoleh and Hasan [27] and Ning and Yin [25]. Their analysis in fact finds multiplication in a normal basis representation faster than polynomial-based multiplication in the cases that the field has an optimal normal basis. This would indeed be surprising, given other published results. As with [26], their reported timings are generally quite slow. Further, the comparison against a polynomial-based multiplication is with a Montgomery-like method from [19]. However, the comparison of interest is against fast methods such as comb multiplication [21, 13] with a suitable reduction polynomial. We present improved algorithms with significantly faster times, although in contrast to [6] we will argue that the evidence is strongly in favor of polynomial-based arithmetic even for optimal normal bases.

In the particular case of a type 1 optimal normal basis, Fan and Dai and the authors of this paper independently noted that the matrix decomposition of Hasan, Wang, and Bhargava [15] can be adapted to significantly improve on Ning and Yin [25]; in [6] this is called the “Hamming weight method” [12]. In particular, the analysis in [6] concludes that the decomposition is only beneficial for fields of sufficient size (roughly 2^{260} elements). We show that in fact the decomposition can be efficiently exploited regardless of field size.

Multiplication via map to an associated ring

Techniques involving essentially fast basis conversions are well-known for optimal normal bases, and are directly related to the methods in §4. For a type 1 basis, the conversion is a permutation. For type 2, there are two related approaches. The traditional approach, illustrated by Sunar and Koç [30], exploits basis conversion. The “palindromic representation” in Blake, Roth, and Seroussi [4] is a special case of the method in §4 and maps field elements into a larger ring where polynomial-based multiplication can be employed. This type of strategy where a mapping to a ring is exploited also appears in Drolet [5] and Katti and Brennan [16]. For an application to (hardware) exponentiation, see Kwon, Kim, and Hong [20].

The general method for Gaussian bases of mapping into an associated ring is described by Gao, von zur Gathen, Panario, and Shoup [9]; see also Wu, Hasan, Blake, and Gao [33]. The focus in the former is asymptotic complexity results, while the latter concentrates on hardware. Von zur Gathen and Nöcker [32] examine exponentiation with polynomial and normal bases representations and provide experimental results. The data and conclusions presented are compatible with our results, although their primary interest and estimates are for larger fields. Our contribution in §4 is a new algorithm suitable for software implementations that exploits the ring mapping and fast polynomial-based multiplication. The result is competitive with the fastest direct methods for multiplication in a low-complexity normal basis, and in fact faster for optimal normal bases.

1.2 Outline

The remainder of this paper is organized as follows. In §2 we describe bit-level and vector-level algorithms for multiplication in \mathbb{F}_{2^m} . A description of the new vector-level algorithm is presented in §2.4. In §3, we present a software implementation of the conventional method and our new method. We also include an improvement of the Reyhani-Masoleh and Hasan method [27]. §4 develops a method using polynomial-based multiplication after mapping to an associated ring, and includes an improvement to the method of Ning and Yin [25] for type 1 ONB. Timings and an analysis for the use of these methods in a larger framework are presented in §5. Summary conclusions appear in §6.

2 Multiplication in \mathbb{F}_{2^m}

In this section, we introduce some basic notions for binary fields \mathbb{F}_{2^m} and present several algorithms for multiplication using a normal basis representation.

It is well known that there exists an element $\beta \in \mathbb{F}_{2^m}$ such that the set $\{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{m-1}}\}$ is a basis of \mathbb{F}_{2^m} over \mathbb{F}_2 , called a *normal basis*. The binary vector associated with $a = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ is $A = (a_0 a_1 a_2 \dots a_{m-1})$. A squaring is a right cyclic shift in this representation.

Notation When the basis $\{\beta^{2^i}\}$ is understood, elements $a \in \mathbb{F}_{2^m}$ are written interchangeably in sum and vector form; e.g., $A = a = \sum_{i=0}^{m-1} a_i \beta^{2^i} = (a_0 a_1 \dots a_{m-1})$. In this context, $a_i \in \mathbb{F}_2$ is a coefficient and $A_s = A \ll s = A^{2^{-s}} = (a_s a_{s+1} \dots a_{s-1})$ will denote the s -fold left cyclic shift of A . Similarly, $A \gg s = A^{2^s}$ is a right cyclic shift of A .

2.1 Normal basis multiplication

Let $\{\beta^{2^i}\}$ be a normal basis for \mathbb{F}_{2^m} , and let $A = (a_0 a_1 \dots a_{m-1})$ and $B = (b_0 b_1 \dots b_{m-1})$ be two elements in \mathbb{F}_{2^m} represented in this normal basis. Let $C = (c_0 c_1 \dots c_{m-1})$ be their product. Let $\beta^{2^i} \beta^{2^j} = \sum_{s=0}^{m-1} \lambda_{ij}^{(s)} \beta^{2^s}$, where $\lambda_{ij}^{(s)} \in \mathbb{F}_2$. Then $c_s = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \lambda_{ij}^{(s)}$ [24]. Thus, multiplication in \mathbb{F}_{2^m} can be carried out using the *multiplication matrix* $M = [\lambda_{ij}^{(0)}]$. The complexity of M , denoted by C_M , is defined to be the number of 1s in M . It is well known that $C_M \geq 2m - 1$ [24]. The normal basis is said to be *optimal* if $C_M = 2m - 1$.

A generalization of optimal normal bases to normal bases of low complexity, known as Gaussian normal bases (GNB), was studied by Ash, Blake, and Vanstone [3]. Let $p = mT + 1$ be a prime. Let $K = \langle u \rangle$ where $u \in \mathbb{Z}_p^*$ has order T . Suppose that the index e of (2) in \mathbb{Z}_p^* satisfies $\gcd(e, m) = 1$. Then $\mathbb{Z}_p^* = \{2^i u^j \mid 0 \leq i < m, 0 \leq j < T\}$, and $K_i = K^{2^i}$ for $0 \leq i < m$ are the cosets of K in \mathbb{Z}_p^* . Since $p \mid 2^{mT} - 1$, there is a primitive p th root of unity $\alpha \in \mathbb{F}_{2^{mT}}$. The *Gauss periods* of type (m, T) are $\beta_i = \sum_{j \in K_i} \alpha^j$ for

$0 \leq i < m$. Let $\beta = \beta_0$. Then $\beta_i \in \mathbb{F}_{2^m}$, $\beta_i = \beta^{2^i}$, and $\{\beta^{2^i} \mid 0 \leq i < m\}$ is a normal basis for \mathbb{F}_{2^m} called a *type T GNB*.

For $T \geq 2$, the multiplication matrix M for a type T GNB satisfies $C_M \leq mT - 1$ [3]. Hence T is a measure of the complexity of the multiplication. Optimal normal bases are precisely the GNBs with $T \in \{1, 2\}$. For fields \mathbb{F}_{2^m} for which there are no optimal normal bases, the GNBs offer an alternative for implementing a normal basis. Gaussian normal bases are explicitly described in the ANSI X9.62 standard [2] for the Elliptic Curve Digital Signature Standard (ECDSA). The types of the five characteristic-two finite fields recommended by NIST [7] for use in the ECDSA are given in Table 1.

Table 1: NIST recommended binary fields \mathbb{F}_{2^m} and type of associated GNB.

m	163	233	283	409	571
Type	4	2	6	4	10

2.2 Conventional GNB bit-level multiplication in \mathbb{F}_{2^m}

We present the conventional algorithm for multiplication in \mathbb{F}_{2^m} using a GNB of type T as described in FIPS 186-2 [7] and ANSI X9.62 [2]. We shall assume that m is odd (so T is even). Define the sequence $J(1), J(2), \dots, J(p-1)$ by $J(k) = i$ if $k \in K_i$. Since $-1 \in \langle u \rangle$ we have $J(p-k) = J(k)$. Then the coefficient c_0 of $C = AB$ can be computed as $c_0 = F(A, B) = \sum_{k=1}^{p-2} a_{J(k+1)} b_{J(k)}$, and the remaining coefficients c_s for $1 \leq s \leq m-1$ are determined by the formula $c_s = F(A \ll s, B \ll s)$. This conventional method is shown in Algorithm 1 which corresponds to the bit-serial NB multiplication circuit introduced by Massey and Omura [22].

Algorithm 1 Conventional bit-level GNB multiplication

INPUT: $A, B \in \mathbb{F}_{2^m}$, $J(k) \in [0, m-1]$ for $1 \leq k \leq p-1$ where $p = mT + 1$.

OUTPUT: $C = AB$.

1. $C \leftarrow 0$
 2. For i from 0 to $m-1$ do
 - 2.1 For k from 1 to $p-2$ do: $c_i \leftarrow c_i + a_{J(k+1)} \cdot b_{J(k)}$.
 - 2.2 $A \leftarrow A \ll 1, B \leftarrow B \ll 1$.
 3. Return (C).
-

2.3 Vector-level multiplication in \mathbb{F}_{2^m}

In this section we describe a vector-level multiplication based on Algorithm 1. The main idea is to rewrite the coefficients c_s from §2.2 in the form $c_s = \sum_{i=0}^{m-1} (a_{i+s} \sum_{k=1}^T b_{w_{ik}+s})$ for some numbers w_{ik} . This reduces the number of AND operations and allows the use of some precomputation to speed the multiplication.

The field elements $\delta_i := \beta\beta^{2^i}$ play a central role. Let $1 \leq i < m$ and define $v_{ik} = J(1 + 2^i u^k)$ for $0 \leq k \leq T-1$. Then

$$\delta_i = \sum_{s,t=0}^{T-1} \alpha^{u^s + 2^i u^t} = \sum_{k,l=0}^{T-1} \alpha^{(1+2^i u^k)u^l} = \sum_{k,l=0}^{T-1} \alpha^{2^{v_{ik}} u^l} = \sum_{k=0}^{T-1} \beta^{2^{v_{ik}}}.$$

The last sum may have repeated terms, which will cancel in pairs. Let n_i denote the number of 1s in the normal basis representation of δ_i , and let $w_{i1} < w_{i2} < \dots < w_{in_i}$ denote the positions of these 1s. Note that n_i is even since T is, and also $2 \leq n_i \leq T$.

Lemma 1 Let m be odd and let $\{\beta, \beta^2, \dots, \beta^{2^{m-1}}\}$ be a type T GNB for \mathbb{F}_{2^m} .

(i) For $1 \leq i < m$, $\delta_i = \delta_{m-i}^{2^i}$ and hence $\{w_{ik} \mid 1 \leq k \leq n_i\} = \{w_{m-i,k} + i \mid 1 \leq k \leq n_i\}$.

(ii) $\{v_{ik} : 0 \leq k \leq T-1\} = \{J(l) : l+1 \in K_i\}$, where $\{\cdot\}$ denotes a multiset.

(iii) If $d \in \langle u \rangle$, $d \neq \pm 1$, then $J(d-1) = J(d^{-1}-1)$. Hence $\sum_{d \in \langle u \rangle \setminus \{\pm 1\}} b_{J(d-1)} = 0$ for all $b \in \mathbb{F}_{2^m}$.

Proof: (i) This statement is true because $\delta_{m-i}^{2^i} = (\beta\beta^{2^{m-i}})^{2^i} = \beta\beta^{2^i}$.

(ii) Using the fact that $-1 \in \langle u \rangle$, we have

$$\begin{aligned} \{v_{ik} : 0 \leq k \leq T-1\} &= \{J(1+2^i u^k) : 0 \leq k \leq T-1\} = \{J(l) : l-1 \in K_i\} \\ &= \{J(-l) : l-1 \in K_i\} = \{J(l) : -l-1 \in K_i\} = \{J(l) : l+1 \in K_i\}. \end{aligned}$$

(iii) Let $l = J(d-1)$. Then $1-d \in K_l$, so $d^{-1}(1-d) \in K_l$. Hence $J(d^{-1}-1) = l$. \square

Now, using Lemma 1(ii) and (iii), the coefficient c_0 from $C = AB$ can be written as follows:

$$c_0 = a_0 b_1 + \sum_{k=1}^{p-3} a_{J(k+1)} b_{J(k)} = a_0 b_1 + \sum_{i=1}^{m-1} a_i \sum_{J(k+1)=i} b_{J(k)} = a_0 b_1 + \sum_{i=1}^{m-1} a_i \sum_{k=0}^{T-1} b_{v_{ik}}.$$

If $n_i < T$, define $w_{ik} = m-1$ for $n_i < k \leq T$. The coefficients c_s can then be expressed as $c_s = a_s b_{1+s} + \sum_{i=1}^{m-1} a_{i+s} \sum_{k=1}^T b_{w_{ik}+s}$ where we have used the fact that n_i and T are even.

In order to derive an algorithm for multiplication whose implementation is more software-oriented, Ning and Yin [25] and Reyhani-Masoleh and Hasan [27] observed that the computation of AB can be carried out by processing simultaneously the computation of c_s for $s = 0, 1, \dots, m-1$. Then $C = AB$ can be written as

$$C = A \odot B_1 \oplus \sum_{i=1}^{m-1} A_i \odot S_B(i), \quad (1)$$

where “ \odot ” denotes the bitwise AND operation, “ \oplus ” denotes the bitwise XOR operation, and $S_B(i) = \sum_{k=1}^{n_i} B_{w_{ik}}$. Algorithm 2 is derived from this formula. For a hardware implementation of this method see [31].

Algorithm 2 Conventional vector-level GNB multiplication

INPUT: $A, B \in \mathbb{F}_{2^m}$, $w_{ik} \in [0, m-1]$, $1 \leq i \leq m-1$, $1 \leq k \leq T$.

OUTPUT: $C = AB$.

1. $C \leftarrow A \odot B_1$, $L_A \leftarrow A$.
 2. For i from 1 to $m-1$ do
 - 2.1 $L_A \leftarrow L_A \ll 1$, $S_B \leftarrow \sum_{k=1}^T B_{w_{ik}}$.
 - 2.2 $C \leftarrow C \oplus L_A \odot S_B$.
 3. Return (C).
-

The property $\delta_i = \delta_{m-i}^{2^i}$ of Lemma 1(i) was used by Reyhani-Masoleh and Hasan [27] to develop an efficient vector-level algorithm based on the formula

$$AB = (A \odot B)^2 \oplus \sum_{j=0}^{m-1} \sum_{i=1}^v (a_j b_{i+j} + a_{i+j} b_j) \sum_{k=1}^{n_i} \beta^{2^{j+w_{ik}}}, \quad (2)$$

where $v = (m-1)/2$. The description of this method is presented in Algorithm 3. Compared with Algorithm 2, 1/3 to 1/2 (depending on T) of the shifting has been eliminated.

Algorithm 3 Reyhani-Masoleh and Hasan vector-level GNB multiplication

INPUT: $A, B \in \mathbb{F}_{2^m}$, $w_{ik} \in [0, m-1]$, $1 \leq i \leq v$, $1 \leq k \leq T$.

OUTPUT: $C = AB$.

1. $C \leftarrow (A \odot B) \gg 1$, $L_A \leftarrow A$, $L_B \leftarrow B$.
 2. For i from 1 to v do
 - 2.1 $L_A \leftarrow L_A \ll 1$, $L_B \leftarrow L_B \ll 1$.
 - 2.2 $R \leftarrow (A \odot L_B) \oplus (B \odot L_A)$.
 - 2.3 For k from 1 to T do: $C \leftarrow C \oplus (R \gg w_{ik})$.
 3. Return (C).
-

2.4 A new method for multiplication in normal bases

In implementation, Algorithms 2 and 3 can have excessive shifting or storage requirements. We present a new algorithm with optimization advantages that will be discussed in §3. The following lemma gives a relation for computing $S_B(m-i)$ in terms of $S_B(i)$, a key result for the development of the new algorithm.

Lemma 2 Let $B \in \mathbb{F}_{2^m}$ and let $S_B(i) = \sum_{k=1}^{n_i} B_{w_{ik}}$ for $1 \leq i < m$. Then $S_B(m-i) = S_B(i)^{2^i}$.

Proof: Using the definition of $S_B(i)$ and Lemma 1(i), one obtains

$$\begin{aligned}
S_B(m-i) &= \sum_{k=1}^{n_i} B_{w_{m-i,k}} = \sum_{k=1}^{n_i} B^{2^{-w_{m-i,k}}} = \sum_{j=0}^{m-1} b_j \left(\sum_{k=1}^{n_i} \beta^{2^{-w_{m-i,k}}} \right)^{2^j} \\
&= \sum_{j=0}^{m-1} b_j \left(\sum_{k=1}^{n_i} \beta^{2^{-w_{ik}+i}} \right)^{2^j} = \left(\sum_{k=1}^{n_i} B^{2^{-w_{ik}}} \right)^{2^i} = S_B(i)^{2^i}. \quad \square
\end{aligned}$$

Theorem 3 Let m be odd and define $v = (m-1)/2$. Let $A, B \in \mathbb{F}_{2^m}$ and $C = AB$. Then

$$C = A \odot B_1 \oplus \sum_{i=v+1}^{m-1} \left[A_i \odot S_B(i) \oplus (A \odot S_B(i))^{2^{-(m-i)}} \right]. \quad (3)$$

Proof: By using the conventional formula (1) and Lemma 2, we can write AB as:

$$\begin{aligned}
AB &= A \odot B_1 \oplus \left(\sum_{i=1}^v A_i \odot (S_B(m-i))^{2^{-i}} \right) \oplus \left(\sum_{i=v+1}^{m-1} A_i \odot S_B(i) \right) \\
&= A \odot B_1 \oplus \left(\sum_{i=v+1}^{m-1} (A \odot S_B(i))^{2^{-(m-i)}} \right) \oplus \left(\sum_{i=v+1}^{m-1} A_i \odot S_B(i) \right). \quad \square
\end{aligned}$$

Algorithm 4 New vector-level GNB multiplication

INPUT: $A, B \in \mathbb{F}_{2^m}$, $w_{ik} \in [0, m-1]$, $v+1 \leq i \leq m-1$, $1 \leq k \leq T$.OUTPUT: $C = AB$.

1. $C \leftarrow A \odot B_1$, $L_A \leftarrow A_v$, $L_T \leftarrow 0$.
 2. For i from $v+1$ to $m-1$ do
 - 2.1 $S_B \leftarrow \sum_{k=1}^T B_{w_{ik}}$, $L_A \leftarrow L_A \ll 1$.
 - 2.2 $C \leftarrow C \oplus L_A \odot S_B$.
 - 2.3 $L_T \leftarrow L_T \oplus A \odot S_B$, $L_T \leftarrow L_T \ll 1$.
 3. Return $(C \oplus L_T)$.
-

A further optimization of the method given in Theorem 3 is the observation that $\sum_{i=v+1}^{m-1} (A \odot S_B(i) \ll m-i)$ can be expressed in terms of simple shift left rotations:

$$[\dots [[A \odot S_B(v+1)]^{2^{-1}} \oplus A \odot S_B(v+2)]^{2^{-1}} \oplus \dots \oplus A \odot S_B(m-1)]^{2^{-1}}.$$

Algorithm 4 is the new vector-level algorithm for computing AB .

Compared to Algorithm 2, Algorithm 4 has half the total computation for S_B (and the same total count for other operations in step 2). Although Algorithm 4 has the same operation count as Algorithm 3, the form will be useful in developing optimized versions for software in §3.

3 Software multiplication

In this section we consider improvements to the vector-level algorithms which are suitable for performing binary field multiplication in software. We assume that the target platform has a W -bit architecture where W is the word-size (in bits). Let $t_W = \lceil m/W \rceil$, and let $s = Wt_W - m$; then $a = (a_0, a_1, \dots, a_{m-1})$ can be stored in an array of t_W W -bit words, $A = (A[0], A[1], \dots, A[t_W - 1])$, where the leftmost bit of $A[0]$ is a_0 and the rightmost s bits of $A[t_W - 1]$ are unused.

Ning and Yin [25] introduced a method for accelerating a software implementation of optimal normal bases. We will see that this method can also be applied to speed Algorithms 2, 3, and 4 since all of these algorithms require, in each iteration, the sum of shift (left or right) rotations of a field element. The technique introduced by Ning and Yin is a method for doing the precomputation of $m-1$ shift left rotations of a field element. More precisely, for $i = 0, 1, \dots, m-1$ let $T_A[i]$ be word 0 of $A_i = A^{\gamma^{(m-i)}}$ (using the W -bit word representation of A_i). Then, for each j , $0 \leq j < m$, the element A_j can be expressed in terms of the lookup table $T_A[0..m-1]$ as follows:

$$A_j = (T_A[j], T_A[j + W \bmod m], T_A[j + 2W \bmod m], \dots, T_A[j + (t_W - 1)W \bmod m]).$$

In order to avoid all the modulo m computations, the size of table T_A can be extended from m words to $2m$ words via $T_A[i+m] = T_A[i] = \text{word 0 of } A_i$, and then “mod m ” may be discarded in the expression for A_j .

3.1 Speeding the conventional algorithm

In this section we show how to accelerate Algorithm 2 using precomputation. The main idea is to use two lookup tables, T_A and T_B , for computing and storing the m rotations of elements A and B , respectively. At a high level, the speedup using the Ning and Yin precomputation technique is summarized as follows:

- *Precomputation:* Compute and store m rotations of A and B ;
- *Initialization:* $C \leftarrow A \odot B_1$;
- *Accumulation:* For $i = 1, 2, \dots, m - 1$, compute $C \leftarrow C \oplus A_i \odot \sum_{k=1}^T B_{w_{ik}}$.

Algorithm 5 is a parallel implementation for the conventional method in terms of word operations.¹

Algorithm 5 Conventional GNB multiplication using the method of Ning and Yin

INPUT: $A, B \in \mathbb{F}_{2^m}$, $w_{ik} \in [0, m - 1]$, $1 \leq i \leq m - 1$, $1 \leq k \leq T$.

OUTPUT: $C = AB$.

1. Compute tables $T_A[0..2m-1]$ and $T_B[0..2m-1]$.
 2. For j from 0 to $t_W - 1$ do
 - 2.1 $C[j] \leftarrow T_A[jW] \odot T_B[1 + jW]$.
 3. For i from 1 to $m - 1$ do
 - 3.1 For j from 0 to $t_W - 1$ do

$$S_B[j] \leftarrow \sum_{k=1}^T T_B[w_{ik} + jW]$$

$$C[j] \leftarrow C[j] \oplus T_A[i + jW] \odot S_B[j].$$
 4. Return ($C = (C[0], C[1], \dots, C[t_W - 1])$).
-

3.2 Speeding the Reyhani-Masoleh and Hasan algorithm

The shift right rotation of R in step 2.3 of Algorithm 3 cannot be computed directly from the precomputed rotation of A and B . However this can be done as follows:

$$\begin{aligned} R \gg w_{ik} &= ((A \odot B_i) \gg w_{ik}) \oplus ((A_i \odot B) \gg w_{ik}) \\ &= (A_{m-w_{ik}} \odot B_{m-(w_{ik}-i \bmod m)}) \oplus (A_{m-(w_{ik}-i \bmod m)} \odot B_{m-w_{ik}}). \end{aligned}$$

Therefore, an improved method based on the observation that the shift right rotation $R \gg w_{ik}$ can be computed in terms of the m shift left rotations of A and B is described in Algorithm 6.²

3.3 Speeding the new algorithm

In this section we will consider two techniques to develop a fast software implementation of Algorithm 4. We first apply the Ning and Yin precomputation technique to accelerate the computation of step 2.1, i.e., we use a lookup table for the m rotations of B . Second, we reduce the amount of shifting required for other elements by simultaneously processing shifts that differ by the wordsize W . This strategy reduces the number of rotations of L_A and L_T and applies whenever $W < v$ (which is typical in our context). Since shifting tends to be expensive, the method can give significant improvements and uses less dynamic storage than Algorithms 5 and 6. The result is presented in Algorithm 7.

¹In [25], the order used to compute $C = AB$ was sequential: $C[0], C[1], \dots, C[t_W - 1]$. We observed that timings measured on Pentium family processors for the parallel version (Algorithm 5) were faster than the original method [25, Algorithm 3]. Tests on Intel Pentium and Sun SPARC processors showed that the results are sensitive to the compiler, code generation options, and specific family processor.

²Fan and Dai [6] independently proposed alternate improvements to the Reyhani-Masoleh and Hasan algorithm.

Algorithm 6 Improved Reyhani-Masoleh and Hasan GNB multiplication

INPUT: $A, B \in \mathbb{F}_{2^m}$, $w_{ik} \in [0, m-1]$, $1 \leq i \leq v$, $1 \leq k \leq T$.OUTPUT: $C = AB$.

1. Compute tables $T_A[0..2m-1]$ and $T_B[0..2m-1]$.
 2. For j from 0 to $t_W - 1$ do
 - 2.1 $C[j] \leftarrow T_A[jW] \odot T_B[1 + jW]$.
 3. For i from 1 to v do
 - 3.1 For j from 0 to $t_W - 1$ do
$$C[j] \leftarrow C[j] \oplus \sum_{k=1}^T (T_A[m - w_{ik} + jW] \odot T_B[m - (w_{ik} - i \bmod m) + jW] \oplus T_B[m - w_{ik} + jW] \odot T_A[m - (w_{ik} - i \bmod m) + jW]).$$
 4. Return ($C = (C[0], C[1], \dots, C[t_W - 1])$).
-

Algorithm 7 New vector-level GNB multiplication

INPUT: $A, B \in \mathbb{F}_{2^m}$, $w_{ik} \in [0, m-1]$, $v+1 \leq i \leq m-1$, $1 \leq k \leq T$.OUTPUT: $C = AB$.

1. Compute table $T_B[0..2m-1]$.
 2. $L_A \leftarrow A \ll v$, $L_T \leftarrow 0$, $s_W = \lceil v/W \rceil$.
 3. For j from 0 to $t_W - 1$ do
 - 3.1 $C[j] \leftarrow A[j] \odot T_B[1 + jW]$.
 4. For i from $v+1$ to $v + (v \bmod W)$ do
 - 4.1 $L_A \leftarrow L_A \ll 1$.
 - 4.2 For l from 0 to $s_W - 1$ do
 - For j from 0 to $t_W - 1$ do
$$S_B[j] \leftarrow \sum_{k=1}^T T_B[w_{(i+lW+jW),k} + jW].$$
 - $C \leftarrow C \oplus (L_A \ll lW) \odot S_B$.
 - $L_T \leftarrow L_T \oplus (A \odot S_B) \ll (s_W - 1 - l)W$.
 - 4.3 $L_T \leftarrow L_T \ll 1$.
 5. $C \leftarrow C \oplus L_T$, $L_T \leftarrow 0$.
 6. For i from $v+1 + (v \bmod W)$ to $v+W$ do
 - 6.1 $L_A \leftarrow L_A \ll 1$.
 - 6.2 For l from 0 to $s_W - 2$ do
 - For j from 0 to $t_W - 1$ do
$$S_B[j] \leftarrow \sum_{k=1}^T T_B[w_{(i+lW+jW),k} + jW].$$
 - $C \leftarrow C \oplus (L_A \ll lW) \odot S_B$.
 - $L_T \leftarrow L_T \oplus (A \odot S_B) \ll (s_W - 2 - l)W$.
 - 6.3 $L_T \leftarrow L_T \ll 1$.
 7. $L_T \leftarrow L_T \ll (v \bmod W)$, $C \leftarrow C \oplus L_T$.
 8. Return ($C = (C[0], C[1], \dots, C[t_W - 1])$).
-

4 Multiplication via ring mapping

In software implementations on common workstations, the fast multiplication method of López and Dahab [21] for polynomial basis representations has been observed to be significantly faster than any method using a normal basis. Although the timings for Algorithm 7 show that the difference is significantly smaller than previously reported, there is at least a factor 5 penalty over the corresponding polynomial basis method in [13].

In a larger framework such as elliptic curve point multiplication, basis conversion at the beginning and end of the expensive operation can be used to permit arithmetic in the desired representation. However, there are scenarios where multiple representations within the larger operation appears attractive. As an example, point multiplication methods via point halving [17, 28] require solutions for quadratic equations and square roots, operations which may be faster with a normal basis representation. Along with these operations is a field multiplication, where polynomial bases are preferred.

For Gaussian normal bases, there are fast methods to convert to a representation in a polynomial basis. The basic strategy has been applied in many contexts and with various descriptions, but in general involves a mapping from the normal basis to a ring where arithmetic can be performed modulo a cyclotomic polynomial. The most familiar is perhaps the case of type 1 ONB, where the mapping is a permutation. In, general, however, the mapping to a ring involves a factor T expansion, a significant hurdle in the practical applicability of the method to field arithmetic.

Gao, von zur Gathen, Panario, and Shoup [9] discuss the method of field operations via the ring mapping approach. Their interest was in asymptotic results concerning the efficiency of field operations. To compute the product, the idea is to map to the ring, and then use “the fast multiplication algorithms of Schönhage and Strassen (1971) and Cantor and Kaltofen (1991)...”. However, our interest is in multiplication for fields of practical interest in cryptography, where asymptotic results on efficiency are insufficient for even rough comparison.

In this section, we examine the multiplication method via a map to a ring. In the case of type 1 ONB, we also apply the matrix decomposition of Hasan, Wang, and Bhargava [15] to provide a significant improvement to the method of Ning and Yin [25]. In the more general case of Gaussian normal bases (as in the NIST recommended fields), we develop a new multiplication method (based on the comb method of [21]) for elements in a normal basis representation via a mapping ϕ to a polynomial representation in a suitable ring. A symmetry property [33] of elements mapped by ϕ is exploited, allowing significant optimizations in the multiplier.

4.1 The ring associated with Gauss periods

In the following, we will assume that β is a Gauss period of type (m, T) and is a normal element (as described in §2). For $a = \sum_{i=0}^{m-1} a_i \beta^{2^i} \in \mathbb{F}_{2^m}$, we have $a = \sum_{i=0}^{m-1} a_i \sum_{j \in K_i} \alpha^j = \sum_{j=1}^{mT} a'_j \alpha^j$ where $a'_j = a_i$ if $j \in K_i$. Let $R = \mathbb{F}_2[x]/(\Phi_p)$ where $\Phi_p(x) = (x^p - 1)/(x - 1)$. The mapping $\phi : \sum_{i=0}^{m-1} a_i \beta^{2^i} \mapsto \sum_{j=1}^{mT} a'_j x^j$ is a ring homomorphism from \mathbb{F}_{2^m} to R [9], and $\phi(\mathbb{F}_{2^m}) = \{c_1 x + \dots + c_{mT} x^{mT} \in R \mid c_j = c_k \text{ for } j, k \in K_i\}$. The mapping ϕ and its inverse are relatively inexpensive, and arithmetic in R benefits from the form of Φ_p . For field multiplication, a naïve approach maps into the ring and then exploits fast polynomial-based arithmetic. However, there is an expansion by a factor T , which can be significant.

If T is even (which is always the case if m is odd), then the last $mT/2$ coefficients for elements in $\phi(\mathbb{F}_{2^m})$ are a mirror reflection of the first $mT/2$ [33]. This property is perhaps best known in the case $T = 2$ where Gauss periods produce a type 2 optimal normal basis of the form $\{(\alpha + \alpha^{-1})^{2^i} \mid 0 \leq i < m\}$ and there is an associated basis $\{\alpha^i + \alpha^{2^{m+1-i}} \mid 1 \leq i \leq m\}$ [24].

In this section, we propose a strategy that exploits the symmetry property of the ring mapping ϕ to allow efficient use of the fast polynomial multiplication method of [21]. Although there is in general an expansion in the mapping, the polynomial multiplier need find only approximately half the coefficients of the product.

We begin in §4.2 with the special case $T = 1$ (a type 1 optimal normal basis). Although our principal interest is in prime m (as in the NIST fields), we describe an optimization specific to the type 1 case that significantly improves on the direct method of Ning and Yin [25]. In addition, we also compare against a mapping to the associated ring, which is a permutation in the type 1 case (and hence there is no expansion). In §4.3, we describe the new multiplier for even T (e.g., the NIST fields).

4.2 Type 1 optimal normal bases

Gaussian normal bases of types 1 and 2 are optimal in the sense that the multiplication matrix has the fewest number of nonzero entries. The type 1 case is particularly attractive, as the matrix has a special form that can be exploited. Further, the mapping ϕ to the associated ring is basis conversion in the type 1 case.

A type 1 basis necessarily has m even, and hence none of the NIST recommended fields has a type 1 basis. In fact, some standards such as the forthcoming revision of ANSI X9.62 explicitly forbid the use of elliptic curves over \mathbb{F}_{2^m} with m composite due to concerns that discrete logarithm problems over such curves are vulnerable to Weil descent attacks [10, 23]. Although our focus is on prime m , the type 1 case provides a benchmark for methods with low-complexity bases. Ning and Yin [25] exploit the special form of the multiplication matrix in the type 1 case. We show that their method can be significantly improved, both in storage required and in speed. Comparison data is presented for the method that maps to an associated ring (a permutation in type 1) and performs polynomial-based arithmetic.

The multiplication matrix M for the type 1 case has nonzero entry at (i, j) precisely when $2^i + 2^j \pmod{m+1} \in \{0, 1\}$. The pairs $(i, j = i + m/2 \pmod{m})$ satisfy $2^i + 2^j \equiv 0 \pmod{m+1}$, giving m of the $2m - 1$ solutions. Ning and Yin exploit this property to reduce the number of lookups in their algorithm corresponding to Algorithm 5 (step 3.1), at the expense of extending T_B by $m/2$ words.

The cost of the Ning and Yin approach can be significantly reduced (in both time and storage) by exploiting the special form of M more directly. Hasan, Wang, and Bhargava [15] suggest the decomposition $M = P + Q$ where P has a 1 at (i, j) precisely when $j = i + m/2 \pmod{m}$. Then P has the property that $(a_s, \dots, a_{s+m-1})P(b_s, \dots, b_{s+m-1})'$ is independent of s , and the value is given by $\sum a_i b_{m/2+i} = \text{Tr}(A \odot B_{m/2})$ where Tr denotes the trace. The decomposition reduces the computational cost to multiplication corresponding to the matrix Q (which has $m - 1$ nonzero entries). Compared to Algorithm 4 in Ning and Yin [25], precomputation uses $m/2$ fewer words, and speed improvement increases with t_W . Algorithm 8 illustrates the approach with two tables of precomputation; the techniques of Algorithm 7 can be applied to give a one-table version.

Algorithm 8 Multiplication for type 1 ONB

INPUT: $A, B \in \mathbb{F}_{2^m}$, position w_i of nonzero entry in row i of Q , $1 \leq i \leq m - 1$.

OUTPUT: $C = AB$.

1. Compute tables $T_A[0..2m - 1]$ and $T_B[0..2m - 1]$.
 2. $C \leftarrow \text{Tr}(A \odot B_{m/2})$.
 3. For i from 1 to $m - 1$ do:
 - 3.1 For j from 0 to $t_W - 1$ do: $C[j] \leftarrow C[j] \oplus T_A[i + jW] \odot T_B[w_i + jW]$.
 4. Return $C = (C[0], \dots, C[t_W - 1])$.
-

While the improvement to the Ning and Yin approach is more than 25% for $m = 162$, timings in Table

2 of §5 show that multiplication is significantly slower than the method of mapping to a ring and using polynomial-based multiplication. In the type 1 case, the mapping is a permutation, and hence relatively fast even in software. The polynomial Φ_p permits fast reduction, and the method (including the conversions between bases) is roughly a factor 2 faster than the best times with a normal-basis approach for $m = 162$. Approximately half the time in the faster approach is consumed by basis conversions (applications of ϕ and its inverse).

4.3 Gaussian normal bases of even type

Assume that we have Gauss periods of type (m, T) for T even, and β is a normal element (as defined in §2.1). The basic strategy proposed for multiplication of $a, b \in \mathbb{F}_{2^m}$ represented in the basis $\{\beta^{2^i}\}$ is outlined as Algorithm 9, with implementation considerations following in Note 1.

Algorithm 9 Multiplication via map to associated ring (outline)

INPUT: elements $a = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ and $b = \sum_{i=0}^{m-1} b_i \beta^{2^i}$ in \mathbb{F}_{2^m} .

OUTPUT: $c = ab = \sum_{i=0}^{m-1} c_i \beta^{2^i}$.

1. Calculate $a' = \phi(a) = \sum_{j=1}^{p-1} a'_j x^j$ and $b' = \phi(b) = \sum_{j=1}^{p-1} b'_j x^j$ in R .
 2. Apply a fast multiplication method for polynomial-based representations to find half the coefficients of $c' = a'b'$ in R .
 3. Return $c = \phi^{-1}(c')$.
-

Note 1 (*implementing Algorithm 9*) As discussed in §4.1, the mapping by ϕ involves a factor T expansion. However, the elements $c' = \sum_{i=1}^{p-1} c'_i x^i = (c'_1, \dots, c'_{p-1})$ in $\phi(\mathbb{F}_{2^m})$ are symmetric about $p/2$ and hence it suffices to find (at most) the first $(p-1)/2$ coefficients of the product in the ring R . This observation suggests the following optimizations in Algorithm 9. We assume the customary representation of elements as bitstrings of coefficients in a series of machine words.

1. If per-field precomputation is used, then the mapping ϕ can be optimized for a specific field. Each output coefficient is obtained with a word shift and mask.³ Only the first half are calculated in this fashion; the remaining coefficients are obtained by symmetry. An 8-bit lookup table can be used to reverse the order of bits. With this arrangement, the cost of applying ϕ is approximately $T/2$ times the cost of ϕ^{-1} .
2. The “comb” method [21] provides fast multiplication for polynomials and can be adapted to find only some of the output coefficients (with a corresponding improvement in performance). If the input is understood to be $a = x \sum_{i=0}^{p-2} a_i x^i = (a_0, \dots, a_{p-2})$ and similarly for b , then the full (polynomial) product is $x^2 \sum_{i=0}^{2p-4} c_i x^i$. If the input corresponds to field elements, then the coefficient of x^p in the product is zero. The comb method is modified to find c_i for $i \in \{0, \dots, (p-1)/2-2, p-1, \dots, p+(p-1)/2-2\}$. Then reduction via $\Phi_p(x)$ is applied, first with $x^p \equiv 1 \pmod{\Phi_p}$ to obtain

$$t \leftarrow x^2[(c_0 + c_p) + \dots + (c_{(p-1)/2-2} + c_{p+(p-1)/2-2})x^{(p-1)/2-2} + c_{p-1}x^{p-1}]$$

or, after a shift and appropriate renaming, $t = x[t_0 + t_1x + \dots + t_{(p-1)/2-1}x^{(p-1)/2-1} + t_px^p]$ where $t_0 = 0$. The reduction is applied again to obtain $c' = \sum_{i=1}^{(p-1)/2} c'_i x^i$ where $c'_1 = t_p$ and $c'_i = t_{i-1}$.

³Ideally, half of the masking operations could be eliminated by more careful organization of the code. This saves a little space and time, although may use registers less efficiently. The technique is likely to be less useful on processors where shifting is more expensive than masking.

$2 \leq i \leq (p-1)/2$. The remaining coefficients for $a'b'$ are not required, but can be found efficiently via the symmetry property of elements in $\phi(\mathbb{F}_{2^m})$.

3. $c = \phi^{-1}(c') = \sum_{i=0}^{m-1} c_i \beta^{2^i}$ is defined by $c_i = c'_j$ where $j = \min\{k2^i \bmod p : k \in K\}$. This formula guarantees that $j < p/2$ and hence c'_j is a coefficient obtained in the preceding step.

Example 1 (*Algorithm 9 for $\mathbb{F}_{2^{163}}$*) The NIST recommended field $\mathbb{F}_{2^{163}}$ has a type $T = 4$ normal basis, and hence the mapping in Algorithm 9 gives a factor 4 expansion. The algorithm uses a modified comb multiplication to find $mT/2 = 326$ coefficients of the product $a'b'$. If 32-bit words are in use, then field elements require 6 words, and ring elements require 21.

The comb method finds words 0, ..., 10, 19, ..., 30 of the complete 42-word polynomial product. In fact, the calculation for words 19 and 30 need not occur on every loop of the combing method; however, code expansion considerations may limit the ability to exploit this property. A small optimization is achieved by noting that word 10 is not required since the field element can be recovered from coefficients c'_1, \dots, c'_{309} of the product c' . Wu et al. [33, Table 2] give sample minima (for several $m \in [153, 235]$) for the number of consecutive coefficients of an R -element that will permit recovery of the associated field element.

Experimentally, times for Algorithm 9 for $m = 163$ on an Intel Pentium III are a factor 7 slower than field multiplication for a polynomial basis representation. The cost of an application of ϕ is approximately 10% of the total field multiplication time (ϕ^{-1} costs approximately half of ϕ). The algorithm is competitive with the best methods for multiplication with a normal basis representation on this platform.

Example 2 (*Algorithm 9 for $\mathbb{F}_{2^{233}}$*) The NIST recommended field $\mathbb{F}_{2^{233}}$ has a type $T = 2$ normal basis. As expected, Algorithm 9 is faster for $m = 233$ (where 233 coefficients in the ring product are found) than for $m = 163$ (where 309 coefficients are found). The method gives the fastest multiplication times for the type 2 case, and is approximately a factor 3 slower than multiplication in a polynomial basis [13].

5 Analysis

Algorithms 7, 8, and 9 for low-complexity normal bases are especially suitable for software implementation, and methods based on these algorithms can be significantly faster than previously published methods for normal basis representations and may have smaller dynamic storage requirements. “Interoperability” is sometimes cited as motivation for implementing arithmetic in a specific basis, and in this context performance improvements are always of interest. However, the interoperability argument has limited applicability in environments with sufficient resources to implement algorithms described in this paper (since such implementations would likely be able to arrange for basis conversions around the expensive operations). Hence, our focus is primarily on suitability in a larger framework.

Of interest is the following question: are the new algorithms sufficiently fast to encourage the use of normal basis representations for software implementations? We consider two well-known examples in methods based on elliptic curves where operations in a normal basis appear to be especially attractive. The first is Koblitz curves [18, 29], where point doubles are replaced by field squaring operations which are especially fast in a normal basis representation. As a second example, elliptic curve point multiplication methods based on point halving [17, 28] require solutions of quadratics, an operation that could presumably benefit significantly from normal basis representations. In this section, we first present timings and implementation issues, and then address the broader questions.

Table 2: Field multiplication times (in μs) of our implementations for \mathbb{F}_{2^m} on an 800 MHz Intel Pentium III. Input and output are in normal basis representation for the five rightmost columns. The compilers are GNU C 2.95 (gcc) and Intel 6.0 (icc) on Linux (kernel 2.4).

Compiler	m, T	Poly-basis mult [21] ^a	Basis conversion ^b	Ning & Yin 2 tables [25, Alg 2-4] ^c	R&H [27] 2 tables ^d Alg 6	§2.3 1 table Alg 2	§2.4 1 table Alg 7	Ring mapping Alg 9
icc 6.0	162,1	1.3	—	6.7	4.5 ^e	5.2 ^f	6.0 ^f	2.7
	163,4	1.3	3.1	9.5	15.6	10.4	8.2	10.4
	233,2	2.3	5.5	11.4	14.8	12.6	11.8	7.1
	283,6	2.9	6.9	33.3	54.3	33.8	24.1	—
gcc 2.95	162,1	1.8	—	8.2	6.4 ^e	7.3 ^f	8.0 ^f	3.7
	163,4	1.8	3.6	14.1	17.0	11.9	12.0	13.0
	233,2	3.0	5.4	13.9	17.3	14.5	15.3	8.9
	283,6	3.9	6.9	48.7	56.4	47.3	34.5	—

^aField multiplication for polynomial-based representations using a comb method (with reduction) [21].

^bTime for a single basis conversion using straightforward change-of-basis matrix. As a benchmark, multiplication of elements represented in a normal basis can be performed using three conversions and the polynomial-basis multiplier.

^cEssentially Algorithm 5; algorithm form and times suggest [25] used a different order of evaluation.

^dAlgorithms 3 and 7a of [27] do not use the precomputation of [25]. For odd m , the improvement is Algorithm 6.

^eEven m is slightly different; type 1 exploits the form of the multiplication matrix [15, 27].

^fAlgs 2 and 7 modified for even m . Includes the matrix decomposition optimizations of Alg 8.

5.1 Timings and implementation considerations

Although operation counts can give rough estimates of algorithm performance, they are often insufficiently precise to capture platform characteristics adequately for code at the level of field arithmetic. In order to evaluate the performance of our software oriented algorithms, we present timing and other results for implementations in C using the GNU compiler (gcc, version 2.95) and the Intel compiler (icc, version 6.0) on an Intel Pentium III running Linux (kernel 2.4). We implemented several algorithms for multiplication in the fields \mathbb{F}_{2^m} for $m \in \{162, 163, 233, 283\}$ and the implementation was optimized for each field using basic performance techniques such as loop unrolling and reducing register consumption (see [14, §5.1] for software optimizations). It has been observed that the GNU compiler often produces somewhat slower code than the Intel compiler on this platform, but can be coerced to produce better sequences by relatively small changes in the source code. The implementation here has received limited such tuning for gcc.

Table 2 shows the running times from our implementation. The fastest times show that Algorithm 7 is 13% to 29% faster than the other direct multiplication algorithms for the entries with $T \geq 4$, and competitive for $T = 2$. The improvement is larger if the type T is larger than 6 as in field $\mathbb{F}_{2^{571}}$ with $T = 10$. Despite our limited efforts to cooperate with compiler characteristics, significant differences in times between compilers remains. In particular, Algorithm 7 with gcc is superior to the other direct methods only in the case $T = 6$.

The method based on mapping to an associated ring (Algorithm 9) is competitive with the fastest direct methods, and significantly faster for optimal normal bases. Of the NIST recommended fields, only $\mathbb{F}_{2^{233}}$ has an optimal basis, and the mapping method is nearly 40% faster than the best direct method. For $\mathbb{F}_{2^{163}}$, the expansion by a factor $T = 4$ is a significant penalty in the mapping method, and in fact the times are slower than those for the larger field $\mathbb{F}_{2^{233}}$.

The comb multiplier in Algorithm 9 is defined for only a subset of the ring, and hence is significantly faster than a multiplier for the full ring. However, the expansion in the mapping implies that the method will be significantly more expensive than a multiplier for the field [21]. In particular, the precomputation is for

Table 3: Approximate code and storage requirements (in 32-bit words) for field multiplication in \mathbb{F}_{2^m} . Data for polynomial basis representations are from a comb method [21]. For normal bases, the data are for a direct approach (Algorithm 7) and with a ring mapping method (Algorithm 9). Storage for automatic variables is an estimate of the maximum stack consumption.

Storage type	$m = 163$			$m = 233$		
	Poly [21]	Alg 7	Alg 9	Poly [21]	Alg 7	Alg 9
object code & static data	544	2092	4144	792	2740	3172
automatic (stack) data	108	360	360	146	510	260
Total	652	2452	4504	938	3250	3432

ring elements, and an entire ring element is “combed.” On the positive side, only half of the product in the ring is calculated, and reduction is especially simple due to the form of Φ_p .

Table 3 compares the memory requirements for the specific implementation of Algorithms 7 and 9 used in the timings for multiplication in \mathbb{F}_{2^m} for $m \in \{163, 233\}$. Experimentally, we observed that the Pentium 4 benefits from certain loop unrolling techniques (especially in Algorithm 7) to a greater degree than the Pentium III. The code size in the table is slightly inflated by these optimizations which give only minor speed improvements on the Pentium III.

The code size in Algorithm 9 is larger for $m = 163$ than $m = 233$ due to the factor T expansion. The code for the mapping ϕ and its inverse consume a significant portion of the total memory requirement. For the type 2 case, however, the total memory consumption is comparable to that of Algorithm 7.

Algorithms 7 and 9 have significantly larger memory requirements than the method from [21] for polynomial basis representations. However, if total memory consumed by field arithmetic is the measurement of interest, then the operations of squaring, square root, and solving $x^2 + x = c$ for normal basis representations will likely have significantly smaller memory requirements than their counterparts for a polynomial basis.

5.2 Normal bases in practice

Beyond the mathematical elegance, normal bases have been of practical interest primarily for the promise of very fast calculations in trace, squaring, square root, and solving $x^2 + x = c$. Although there was some early enthusiasm for normal bases arithmetic in software, the consensus is that the penalty for multiplication compared with methods for polynomial bases will overwhelm savings in other operations in common elliptic curve point operations. For point operations involving only field addition, multiplication, and squaring, a polynomial-based squaring operation is sufficiently fast relative to multiplication that the squarings are typically ignored in rough estimates of point operation cost.

The times in Table 2 are significantly faster than in earlier papers, and suggest (at least on this platform) that multiplication for Gaussian normal bases is much closer in performance to multiplication in a polynomial basis than previously believed. While the difference is still sufficiently large to discourage the use of normal bases for “traditional” elliptic curve point operations of addition and doubling, we consider the implications for Koblitz curves and methods based on point halving.

Koblitz curves

Koblitz curves [18] are elliptic curves defined over \mathbb{F}_2 . The primary advantage of these curves is that point multiplication algorithms can be devised that replace point doubles by field squaring operations. We consider the case of point multiplication kP where P is not known in advance. Windowing methods [29]

reduce the number of point additions required, at the cost of storage for a few points of precomputation. A width- w window requires 2^{w-2} points of storage, and an associated point multiplication method will have approximately $m/(w+1)$ point additions.

For values of m under consideration, $w = 5$ is likely near-optimal. Scalar multiplication then has an average of 6 applications of τ for each point addition, where τ maps points by squaring each coordinate. In projective coordinates, this is an average of 18 field squarings per point addition. In polynomial basis representations, the squarings are not completely free. However, if $a = \sum_{i=0}^{m-1} a_i x^i \in \mathbb{F}_{2^m}$ then $a^2 = \sum_{i=0}^{m-1} a_i x^{2i}$ and a fast squaring is obtained via an 8-to-16 bit expansion table (followed by reduction). Experimentally, the ratio of multiplication to squaring costs (for polynomial basis representations) is estimated between 6.5 and 10 for $m \in \{163, 233, 283\}$ on a Pentium III [14].

In short, the 18 field squarings between point additions have cost below 3 multiplications in a polynomial basis. Point addition requires 8 multiplications (assuming mixed coordinates). Regardless of method (basis conversion, direct, or ring mapping), Table 2 suggests that the added costs of normal basis multiplication in point addition will overwhelm the relatively small savings in squarings.

Point halving

Halving-based methods [17, 28] replace most point doubles by a potentially faster halving operation. Given a point P on an elliptic curve, point halving produces a point Q with $2Q = P$. We consider only curves $y^2 + xy = x^3 + ax^2 + b$ over \mathbb{F}_{2^m} with $\text{Tr}(a) = 1$ where halving methods are especially attractive; this includes the NIST recommended random curves over binary fields.

Each halving requires a multiplication, a trace computation, a solution to a quadratic $x^2 + x = c$, and a square root. Point multiplication methods typically perform several consecutive halvings between point additions; each addition requires an extra field multiplication to convert coordinates from the halvings. The trace of a field element is the sum of coefficients in a normal basis representation. If per-field precomputation can be performed, then the calculation for a polynomial basis is similarly fast; for the NIST fields, the trace is determined by a few coefficients of the polynomial representation [1], so is essentially free.

Let M denote the cost of a field multiplication in \mathbb{F}_{2^m} in a polynomial basis. The cost of a square root in a polynomial basis representation depends on the reduction polynomial. In the case of a pentanomial, the cost is estimated at $M/2$ provided that a small amount of per-field precomputation is done. The cost is significantly less for trinomials [8].

The quadratic $x^2 + x = c$ has a solution if and only if $\text{Tr}(c) = 0$; if x is a solution, then $x + 1$ is a solution. In a normal basis representation, the solution can be found bitwise. Efficient implementation of the solver is the difficult part of optimizing halving in a polynomial basis. The solution is given by the half-trace H defined by $H(c) = \sum_{i=0}^{(m-1)/2} c^{2^{2i}}$ for odd m . The basic strategy uses per-field precomputation along with the property $H(c) = H(c^2) + c + \text{Tr}(c)$ to reduce the cost of H [17]. The cost of solving the quadratic in a polynomial basis is estimated experimentally (on a Pentium III) to be roughly $2M/3$ for $m = 163$ with 30 elements of precomputation, and roughly $M/2$ for $m = 233$ with 43 elements of precomputation [8].

If we look at the times from $m = 233$ where a multiplication in a normal basis is roughly $3M$, then normal bases are preferred in the halving step only if the combined costs of solving the quadratic and finding a square root in a polynomial basis is more than $2M$. Further, point additions will be significantly more expensive in a normal basis. Hence, the data presents a compelling case for polynomial bases in methods based on point halving.

To be certain, there are considerations that may favor a normal basis implementation. In particular, implementing the solver for $x^2 + x = c$ in a polynomial basis is somewhat an art. Halving can benefit from significant amounts of per-field precomputation, but platform and application constraints can limit the practical applicability. Reducing the amount of precomputation in the solver below a threshold (determined

by the reduction polynomial) is not completely straightforward [8]. Finally, it should be noted that a point double can be done with approximate cost $4M$ or $5M$ for projective coordinates in a polynomial basis representation, and hence there is a rather small window of performance where halving-based methods can be faster.

6 Conclusions

This paper has presented two distinct approaches to multiplication for low-complexity normal bases. The first, in §2, performs a direct normal-basis multiplication. The second, in §4, exploits a mapping to an associated ring and then uses fast polynomial-based multiplication methods. The algorithms developed here lead to multiplication methods significantly faster than reported earlier and having smaller data-dependent storage requirements.

On the other hand, multiplication in a normal basis representation is still significantly slower than the fastest methods for polynomial basis representations. We conclude that the penalty appears to be sufficient in software implementations to overwhelm the advantages of fast and elegant operations of trace, squaring, square root, and solving $x^2 + x = c$ in normal basis representations in the larger framework of common methods for point multiplication on elliptic curves. We stress that our conclusions about the deficiencies of normal bases are only valid for software implementations, and not for hardware implementations.

Acknowledgments The authors wish to thank the anonymous referees for numerous suggestions improving this manuscript.

References

- [1] O. Ahmadi and A. Menezes. On the number of trace-one elements in polynomial bases for \mathbb{F}_{2^n} . *Designs, Codes and Cryptography*, (to appear).
- [2] ANSI X9.62. Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA). American National Standards Institute, 1999.
- [3] D. W. Ash, I. F. Blake, and S. A. Vanstone. Low complexity normal bases. *Discrete Applied Mathematics*, 25:191–210, 1989.
- [4] I. F. Blake, R. M. Roth, and G. Seroussi. Efficient arithmetic in $GF(2^n)$ through palindromic representation. Technical Report HPL-98-134, Hewlett-Packard, 1998.
- [5] G. Drolet. A new representation of elements of finite fields $GF(2^m)$ yielding small complexity arithmetic circuits. *IEEE Transactions on Computers*, 47(9):938–946, 1998.
- [6] H. Fan and Y. Dai. Two software normal basis multiplication algorithms for $GF(2^n)$. Cryptology ePrint Archive, Report 2004/126, 2004.
- [7] FIPS 186-2. Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2, National Institute of Standards and Technology, 2000.
- [8] K. Fong, D. Hankerson, J. López, and A. Menezes. Field inversion and point halving revisited. *IEEE Transactions on Computers*, 53(8):1047–1059, 2004.
- [9] S. Gao, J. von zur Gathen, D. Panario, and V. Shoup. Algorithms for exponentiation in finite fields. *Journal of Symbolic Computation*, 29:879–889, 2000.
- [10] P. Gaudry, F. Hess, and N. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *Journal of Cryptology*, 15:19–46, 2002.

- [11] R. Granger, D. Page, and M. Stam. Hardware and software normal basis arithmetic for pairing based cryptography in characteristic three. Cryptology ePrint Archive, Report 2004/157.
- [12] F. Haining. Simple multiplication algorithm for a class of $GF(2^n)$. *Electronics Letters*, 32(7):636–637, 1996.
- [13] D. Hankerson, J. López, and A. Menezes. Software implementation of elliptic curve cryptography over binary fields. *Cryptographic Hardware and Embedded Systems—CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 1–24. 2000.
- [14] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.
- [15] M. A. Hasan, M. Z. Wang, and V. K. Bhargava. A modified Massy-Omura parallel multiplier for a class of finite fields. *IEEE Transactions on Computers*, 42(10):1278–1280, 1993.
- [16] R. Katti and J. Brennan. Low complexity multiplication in a finite field using ring representation. *IEEE Transactions on Computers*, 52(4):418–427, 2003.
- [17] E. Knudsen. Elliptic scalar multiplication using point halving. *Advances in Cryptology—ASIACRYPT '99*, volume 1716 of *Lecture Notes in Computer Science*, pages 135–149. 1999.
- [18] N. Koblitz. CM-curves with good cryptographic properties. *Advances in Cryptology—CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 279–287. 1992.
- [19] Ç. Koç and T. Acar. Montgomery multiplication in $GF(2^k)$. *Designs, Codes and Cryptography*, 14:57–69, 1998.
- [20] S. Kwon, C. H. Kim, and C. P. Hong. Efficient exponentiation for a class of finite fields $GF(2^n)$ determined by Gauss periods. *Cryptographic Hardware and Embedded Systems—CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 228–242. 2003.
- [21] J. López and R. Dahab. High-speed software multiplication in \mathbb{F}_{2^m} . *Progress in Cryptology—INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science*, pages 203–212. 2000.
- [22] J. L. Massey and J. K. Omura. Computational method and apparatus for finite field arithmetic. US Patent No. 4,587,627, 1986.
- [23] M. Maurer, A. Menezes, and E. Teske. Analysis of the GHS Weil descent attack on the ECDLP over characteristic two finite fields of composite degree. *LMS Journal of Computation and Mathematics*, 5:127–174, 2002.
- [24] R. Mullin, I. Onyszchuk, S. Vanstone, and R. Wilson. Optimal normal bases in $GF(p^n)$. *Discrete Applied Mathematics*, 22:149–161, 1988/89.
- [25] P. Ning and Y. Yin. Efficient software implementation for finite field multiplication in normal basis. *Information and Communications Security 2001*, volume 2229 of *Lecture Notes in Computer Science*, pages 177–189. 2001.
- [26] A. Reyhani-Masoleh. Efficient algorithms and architectures for field multiplication using Gaussian normal bases. Technical Report CACR 2004-04, University of Waterloo, Canada, 2004.
- [27] A. Reyhani-Masoleh and M. A. Hasan. Fast normal basis multiplication using general purpose processors. *IEEE Transactions on Computers*, 52(11):1379–1390, 2003.
- [28] R. Schroepel. Elliptic curves: Twice as fast! Presentation at the CRYPTO 2000 Rump Session, 2000.
- [29] J. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, 19:195–249, 2000.
- [30] B. Sunar and Ç. K. Koç. An efficient optimal normal basis type II multiplier. *IEEE Transactions on Computers*, 50(1):83–87, 2001.
- [31] V. Trujillo, J. Velasco, and J. López. Design of an elliptic curve processor over $GF(2^{163})$. *IBERCHIP*. 2004.
- [32] J. von zur Gathen and M. Nöcker. Polynomial and normal bases for finite fields. *Journal of Cryptology*, 2005.
- [33] H. Wu, A. Hasan, I. F. Blake, and S. Gao. Finite field multiplier using redundant representation. *IEEE Transactions on Computers*, 51(11):1306–1316, 2002.