# SOFTWARE IMPLEMENTATION OF ARITHMETIC IN $\mathbb{F}_{3^m}$

OMRAN AHMADI, DARREL HANKERSON, AND ALFRED MENEZES

Abstract. Fast arithmetic for characteristic three finite fields $\mathbb{F}_{3^m}$ is desirable in pairing-based cryptography because there is a suitable family of elliptic curves over $\mathbb{F}_{3^m}$ having embedding degree 6. In this paper we present some structure results for Gaussian normal bases of $\mathbb{F}_{3^m}$, and use the results to devise faster multiplication algorithms. We carefully compare multiplication in $\mathbb{F}_{3^m}$ using polynomial bases and Gaussian normal bases. Finally, we compare the speed of encryption and decryption for the Boneh-Franklin and Sakai-Kasahara identity-based encryption schemes at the 128-bit security level, in the case where supersingular elliptic curves with embedding degrees 2, 4 and 6 are employed.

## 1. Introduction

Pairing-based cryptographic protocols are realized using algebraic curves of low-embedding degree. Several families of low-embedding degree elliptic curves have been considered, including supersingular curves with embedding degrees 2, 4, and 6, and ordinary curves with embedding degrees 2 [29], 6 [23], and 12 [5]. The family of supersingular elliptic curves with embedding degree 6 is defined over characteristic three finite fields $\mathbb{F}_{3^m}$. Consequently, the software and hardware implementation of arithmetic in these fields has been intensively studied in recent years [16, 25, 13, 20, 14].

The elements of $\mathbb{F}_{3^m}$ can be represented using a polynomial basis or a normal basis. We present some new structure results for Gaussian normal bases of $\mathbb{F}_{3^m}$, and use these results to devise faster multiplication algorithms. Our implementation on a Pentium 4 machine shows that our fastest algorithm for normal basis multiplication in $\mathbb{F}_{3^{239}}$ is about 50% faster than standard Ning-Yin multiplication [24], and about 4.4 times faster than the Ning-Yin implementation reported by Granger, Page and Stam [14]. Our experiments also suggest that the comb method for polynomial basis multiplication [22] (perhaps combined with shallow-depth Karatsuba-like techniques) is faster than Karatsuba multiplication. In particular, our implementation for polynomial basis multiplication in $\mathbb{F}_{3^{239}}$ is about 4.6 times faster than that reported in [14]. We conclude, as in [14], that polynomial bases are preferred over normal bases for the software implementation of characteristic three field arithmetic.

A recent IETF draft standard [8] for identity-based encryption (IBE) mandates use of supersingular elliptic curves over prime fields — these curves have embedding degree 2. We compare the speed of encryption and decryption for the Boneh-Franklin [7] and Sakai-Kasahara [27] identity-based encryption schemes, when the

*Key words and phrases.* Gaussian normal bases, finite field arithmetic, pairings.

underlying elliptic curves are supersingular and defined over a prime field (embedding degree 2), a characteristic two finite field (embedding degree 4), and a characteristic three finite field (embedding degree 6). We focus our attention on 1536-bit prime fields $\mathbb{F}_p$, the characteristic two field $\mathbb{F}_{2^{1223}}$, and the characteristic three field $\mathbb{F}_{3^{509}}$. Each of these choices achieves the 128-bit security level in the sense that the best attacks known on the discrete logarithm problem in the extension fields $\mathbb{F}_{p^2}$, $\mathbb{F}_{2^{4\cdot 1223}}$, and $\mathbb{F}_{3^{6\cdot 509}}$ have running time approximately $2^{128}$ [21].

We acknowledge that the Barreto-Naehrig (BN) ordinary elliptic curves [5] over 256-bit fields $\mathbb{F}_p$ with embedding degree 12 are ideally suited for pairing applications at the 128-security level, and can be expected to yield faster implementations than supersingular elliptic curves especially when the Ate pairing algorithm [18] is employed. However, some people are reluctant to use the BN curves because recent work by Schirokauer [28] has raised the possibility that the special number field sieve may be effective for computing discrete logarithms in $\mathbb{F}_{p^{12}}$. Furthermore, in some cases expensive hashing or the absence of an efficiently-computable isomorphism $\psi$ (cf. [12]) may be a concern. Thus it is worthwhile to consider the relative merits of supersingular elliptic curves in pairing-based cryptography.

The remainder of the paper is organized as follows. Methods for performing arithmetic in $\mathbb{F}_{3^m}$ using a polynomial basis representation are reviewed in §2. Our structure results for Gaussian normal bases are developed in §3. In §4, we present our implementation results for $\mathbb{F}_{3^m}$. Estimates for Boneh-Franklin and Sakai-Kasahara IBE are given in §5. Summary conclusions appear in §6.

## 2. Polynomial bases for $\mathbb{F}_{3^m}$

Elements $a \in \mathbb{F}_{3^m}$ can be regarded as polynomials $a = a_{m-1}x^{m-1} + \cdots + a_0$ where $a_i \in \mathbb{F}_3$ and arithmetic is performed modulo an irreducible polynomial $f$ of degree $m$. We associate $a$ with the vector of coefficients $(a_{m-1}, \ldots, a_0)$.

Harrison, Page, and Smart [16] considered two coefficient representations suitable for implementation. Their "Type II" representation is closer to common techniques used for binary fields. Each coefficient $a_i \in \mathbb{F}_3$ is represented uniquely in $\{0, 1, -1\}$ using a pair $(a_i^0, a_i^1)$ of bits, where $a_i = a_i^0 - a_i^1$ and not both bits are 1. Elements $a$ are represented by vectors $a^j = (a_{m-1}^j, \ldots, a_0^j)$, $j \in \{0, 1\}$. Addition $c = a + b$ is

$$t \leftarrow (a^0 \vee b^1) \oplus (a^1 \vee b^0), \quad c^0 \leftarrow (a^1 \vee b^1) \oplus t, \quad c^1 \leftarrow (a^0 \vee b^0) \oplus t.$$

The seven operations involve only bitwise "or" ($\vee$) and "exclusive-or" ($\oplus$), and it is easy to order the instructions to cooperate with processor pipelining. Negation is $-a = (a^1, a^0)$.

The analysis and experimental data in [16] strongly favour the Type II approach. This also has the advantage that representations are unique, and common techniques employed for multiplication in binary fields have direct analogues. Hence, as in [14], we consider only this representation.

### 2.1. **Field multiplication.**

Techniques for binary field multiplication that extend directly to the representation $a = (a^0, a^1)$ include table lookup and "comb" methods

[22], possibly combined with Karatsuba-like techniques to reduce the number of word-level multiplications at the expense of more additions.

Briefly, a common case of the comb [22] calculates $ab$ with a single table of precomputation containing $ub$ for polynomials $u$ of degree less than $w$ for some small $w$ (e.g., $w = 4$). The words of $a$ are then "combed" $w$ bits at a time to select the appropriate precomputed value to add at the desired location of the accumulator.

In binary fields, our experience and analysis suggests the comb method will be among the fastest on common processors. We also found this to be the case for characteristic three finite fields, contrary to the findings in [14] where the Karatsuba-Ofman style approach was the fastest choice. Indeed, the times in Table 1 (see §4) show that a comb method is dramatically faster on the processor used in [14] (an Intel Pentium 4).

The comb method, while not difficult to implement, requires attention to processor and compiler characteristics [15, Chapter 5]. It appears necessary to code a multiplication for each field size (in number of words to hold an element). With fields of interest in methods for elliptic curves, Karatsuba-Ofman can be used at shallow depth to reduce the multiplication to a few comb sizes, so code expansion can be controlled.

2.2. **Cubing and cube roots.** Since $a^3 = (\sum a_i x^i)^3 = \sum a_i x^{3i}$, cubing is an inexpensive operation when performed using an expansion table to "thin" the coefficients followed by a reduction. This is analogous to squaring in binary fields. The cost of cube roots depends on the reduction polynomial. Since the operation is linear, write

$$a^{1/3} = \sum_{i=0}^{\lceil m/3 \rceil - 1} a_{3i} x^i + x^{1/3} \left( \sum_{i=0}^{\lceil m/3 \rceil - 1} a_{3i+1} x^i \right) + x^{2/3} \left( \sum_{i=0}^{\lceil m/3 \rceil - 1} a_{3i+2} x^i \right).$$

If $x^{1/3}$ and $x^{2/3}$ are precomputed, then $a^{1/3}$ can be found using a table-lookup method to extract coefficients from $a$ followed by two multiplications where each has an operand that has all nonzero coefficients in the lower third of the vector representation. If the reduction polynomial can be chosen so that $x^{1/3}$ has only a few nonzero terms, then roots are especially inexpensive.

**Example 1.** Consider $\mathbb{F}_3[x]/(f)$ where $f(x) = x^m + f_k x^k + f_0$ is irreducible. If $m \equiv k \pmod 3$, then $x^{1/3}$ has $4 - (m \bmod 3)$ nonzero terms [3]. For $m = 239$ (of interest in [14], for example), the polynomial $f(x) = x^m - x^k + 1$ for $k = 5$ is irreducible and has $m \equiv k \equiv 2$ so that $x^{1/3} = -x^{80} + x^2$ has just two terms. However, [14] selected $f(x) = x^{239} + x^{24} - 1$, and [1] shows that $x^{1/3}$ will have $\lceil \frac{2m-1}{3k} \rceil + \lceil \frac{2m-1-k}{3k} \rceil + \lceil \frac{2m-1-2k}{3k} \rceil + j = 20 + j$ nonzero terms where $0 \le j \le 3$ (in fact, there are 23 nonzero terms in $x^{1/3}$ and 9 nonzero terms in $x^{2/3}$).

There are $m$ where no irreducible trinomial yields sparse $x^{1/3}$; an almost worst case is illustrated by $m = 163$ where $x^{1/3}$ has $m - 1$ nonzero terms. However, there is an irreducible tetranomial where $x^{1/3}$ has five terms, and an irreducible pentanomial where $x^{1/3}$ has 3 terms. The case $m = 509$ also has no trinomial giving a low-weight $x^{1/3}$. There is a tetranomial giving a 17-term $x^{1/3}$ and a pentanomial

where $x^{1/3}$ has 5 terms. To minimize the combined hamming weight of $x^{1/3}$ and $x^{2/3}$, a pentanomial can be chosen (e.g., $x^{509} + x^{294} - x^{215} + x^{79} - 1$) where $x^{1/3}$ has 6 terms and $x^{2/3}$ has 3 terms.

2.3. **Inversion.** Euclidean algorithm variants for inversion in binary fields adapt fairly directly to the representation $a = (a^0, a^1)$. As an alternative, the inverse can be found by exponentiation. Although [14] remark that "one cannot use Itoh-Tsujii type methods to reduce the cost", in fact such methods apply. To see this, note that

$$a^{3^k-1} = \begin{cases} a^2(a^{3^{k-1}-1})^3, & k \text{ odd}, \\ (a^{3^{k/2}-1})^{3^{k/2}+1} = c \cdot c^{3^{k/2}}, & k \text{ even}, \end{cases}$$

where $c = a^{3^{k/2}-1}$. Then $a^{-1} = a^{3^m-2} = a(a^{3^{m-1}-1})^3$. Since $a^2$ can be calculated once, the cost of inversion by a recursive approach is $\lfloor \log(m-1) \rfloor + \mathrm{wt}(m-1) + 1$ field multiplications (where wt is Hamming weight) along with many cubings. The technique is most applicable when $m-1$ has low weight and cubings are extremely cheap (as in normal basis representations). In a polynomial basis, it is expected to be more expensive than inversion based on the Euclidean algorithm, although it has the advantage that it requires very little additional code over multiplication (and is thus especially suitable for hardware).

## 3. Normal bases

Let $\alpha$ generate a normal basis $N$ for $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$, and let $\alpha_i = \alpha^{q^i}$ for $0 \le i \le m-1$. Let $\alpha_i \alpha_j = \sum_{k=0}^{m-1} t_{ij}^{(k)} \alpha_k$, where $t_{ij}^{(k)} \in \mathbb{F}_q$. For $a \in \mathbb{F}_{q^m}$ let $a_i \in \mathbb{F}_q$ be defined by $a = \sum_{i=0}^{m-1} a_i \alpha_i$ and let $A = (a_0, \ldots, a_{m-1})$. Then $c = ab$ is given by $c_k = \sum_{i,j} a_i b_j t_{ij}^{(k)} = A T_k B'$ where the collection of matrices $\{T_k = (t_{ij}^{(k)})\}$ is known as a multiplication table for $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. It is known that $t_{ij}^{(k)} = t_{i-k,j-k}^{(0)}$ and hence $c_k = A^{(k)} T_0 B^{(k)'}$ where $A^{(k)}$ denotes the left cyclic shift of the vector $A$ by $k$ positions.

Let $\alpha \alpha_i = \sum_{j=0}^{m-1} t_{ij} \alpha_j$, for $0 \le i \le m-1$. Then $t_{ij}^{(k)} = t_{i-j,k-j}$, and $T = (t_{ij})$ and $T_0$ have the same number of nonzero entries, known as the *complexity* $C_N$ of the normal basis $N$. It is known that $C_N \ge 2m-1$, and $N$ is said to be *optimal* if $C_N = 2m-1$.

3.1. **Gauss periods.** Let $k, m$ be such that $r = mk+1$ is prime and $\gcd(r, q) = 1$. Let $\beta$ be a primitive $r$-th root of unity in an extension of $\mathbb{F}_q$, and let $\gamma$ be a primitive $k$-th root of unity in $\mathbb{Z}_r$. The element $\alpha = \sum_{j=0}^{k-1} \beta^{\gamma^j}$ is a *Gauss period of type* $(m, k)$ for $\mathbb{F}_q$. In fact, $\alpha \in \mathbb{F}_{q^m}$ and is normal if and only if $\langle q^i \gamma^j \rangle = \mathbb{Z}_r^*$. In this case, every element of $\mathbb{Z}_r^*$ can be written uniquely as $q^i \gamma^j$ where $0 \le i \le m-1$ and $0 \le j \le k-1$ [2]. In the following, $\mathrm{Tr} : a \mapsto \sum_{i=0}^{m-1} a^{3^i}$ is the trace function of $\mathbb{F}_{3^m}$ over $\mathbb{F}_3$.

**Lemma 1.** *If $\alpha$ is a Gauss period and is normal, then* $\mathrm{Tr}(\alpha) = -1$.

*Proof.* In the notation of this section,

$$\mathrm{Tr}(\alpha) = \sum_{i=0}^{m-1} \left( \sum_{j=0}^{k-1} \beta^{\gamma^j} \right)^{q^i} = \sum_{i=0}^{m-1} \sum_{j=0}^{k-1} \beta^{\gamma^j q^i} = \sum_{\ell=1}^{r-1} \beta^{\ell}$$

since $\langle q^i \gamma^j \rangle = \mathbb{Z}_r^*$. The last sum is $(\beta^r - 1)/(\beta - 1) - 1 = -1$. □

Since $\mathrm{Tr}(\alpha) = \sum_{i=0}^{m-1} \alpha_i = -1$, the normal basis representation of the identity element in $\mathbb{F}_{3^m}$ is the vector all of whose entries are $-1$.

3.2. **Complexity and structure for $T$ when $q = 3$.** Fix $q = 3$ and assume that $\alpha$ is a Gauss period of type $(m, k)$ and that $\alpha$ is normal. Note that $m$ odd implies that $k$ is even. Since our main interest is prime $m > 2$, we shall henceforth assume $k$ is even. The normal basis $N$ generated by $\alpha$ is called a *Gaussian normal basis (GNB)* for $\mathbb{F}_{3^m}$, and is said to be of *type* $k$.

We are interested in the complexity of the multiplication and in the "structure" of the multiplication matrix $T$, in particular, the number of entries that are $-1$. As a direct consequence, we will obtain results for $T$ that are of practical interest, including a decomposition that accelerates the multiplication significantly when $k$ is 2 or 4.

We define the complexity $C_i$ of the $i$th row of the matrix $T$ to be its number of nonzero entries. Notice that by definition $C_N = \sum_{i=0}^{m-1} C_i$. Now we may write

$$\sum_{j=0}^{m-1} t_{ij}\alpha_j = \alpha\alpha_i = \alpha\alpha^{3^i} = \sum_{s=0}^{k-1} \sum_{\ell=0}^{k-1} \beta^{\gamma^\ell(1+\gamma^s 3^i)} = \sum_{s=0}^{k-1} f(i,s), \tag{1}$$

where $f(i,s)$ is defined to be $\sum_{\ell=0}^{k-1} \beta^{\gamma^\ell(1+\gamma^s 3^i)}$. Since $\alpha$ is a Gauss period which is normal, there is a unique pair $(i,s)$ such that $1 + \gamma^s 3^i = 0$, namely $(i = 0, s = k/2)$, and then $f(0, k/2) = k$. If $(i,s) \neq (0, k/2)$, then $1 + \gamma^s 3^i = \gamma^t 3^j$ for some $t$ and $j$, and then $f(i,s) = \alpha_j$. Hence, if $i \geq 1$, then $C_i \leq k$. These observations lead to well-known upper bounds on $C_N$.

**Theorem 2** ([6, Theorem 5.5])**.** *$C_N \leq (m-1)k + m = (k+1)m - k$ and $C_N \leq (m-1)k + k - 1 = mk - 1$ if $k \equiv 0 \pmod 3$. If $k = 2$, then $C_N = 3m - 2$.*

The next result establishes some lower bounds on $C_N$. Further, the number of $-1$ entries in the multiplication matrix is given for some cases; this number is of practical interest because it can affect implementation optimizations.

**Theorem 3.** *Suppose that $\alpha$ is a Gauss period of type $(m,k)$ for $\mathbb{F}_3$, $k$ is an even number, and $\alpha$ is normal. Let $f$ be defined by (1). Then a lower bound on the complexity of the Gaussian normal basis generated by $\alpha$ is*

$$C_N \geq \begin{cases} mk - 1 - (k/2-1)(k-1), & \text{if } k \equiv 0 \pmod 3, \\ (k+1)m - k - 1 - (k/2-1)(k-2), & \text{if } k \equiv 1 \pmod 3, \\ (k+1)m - k - (k/2-1)(k-1), & \text{if } k \equiv 2 \pmod 3. \end{cases}$$

*Furthermore, the lower bound is achieved if and only if*

(i) *there are no* $i \geq 1$ *and distinct* $s_1$, $s_2$, $s_3$, $s_4$ *such that* $f(i,s_1) = \cdots = f(i,s_4)$;
(ii) *if* $k \equiv 0 \pmod 3$, *then there are no distinct* $s_1$, $s_2$, $s_3$, $s_4$ *such that* $f(0,s_1) = \cdots = f(0,s_4)$; *and*
(iii) *if* $k \not\equiv 0 \pmod 3$, *then there are no distinct* $s_1$, $s_2$, $s_3$ *such that* $f(0,s_1) = f(0,s_2) = f(0,s_3)$.

*Also if there are no* $i$, $0 \leq i \leq m-1$, *and distinct* $s_1$, $s_2$, $s_3$ *such that* $f(i,s_1) = f(i,s_2) = f(i,s_3)$, *then the number of* $-1$ *entries in the matrix* $T$ *is*

$$\begin{cases} (k/2-1)(k-1), & \text{if } k \equiv 0 \pmod 3, \\ m + (k/2-1)(k-1) - k + 1, & \text{if } k \equiv 1 \pmod 3, \\ (k/2-1)(k-2) + 1, & \text{if } k \equiv 2 \pmod 3. \end{cases}$$

*Proof.* For $i = 0, \ldots, m-1$, let $E_i$ denote the number of triples $(i, s', s'')$ such that $s' \neq s''$ and $f(i,s') = f(i,s'')$. We have $C_i \geq k - E_i/2$ for $i \geq 1$. This is because each $f(i,s)$ is a basis element for every $s$ and $i \geq 1$. Moreover, if there exist distinct $s_1, \ldots, s_\ell$ such that $f(i,s_1) = \cdots = f(i,s_\ell)$ and $\ell \geq 2$, then the complexity of row $i$ is decreased by 1 if $\ell = 2$ and at most by $\ell$ if $\ell \geq 3$ while $E_i$ is increased by $\ell(\ell-1)$. Also it is easy to see that $C_i = k - E_i/2$ if and only if there are no distinct $s_1$, $s_2$, $s_3$, $s_4$ such that $f(i,s_1) = \cdots = f(i,s_4)$. In the following we obtain some inequalities involving $C_0$ which, together with the inequalities obtained for $C_i$, $i = 1, \ldots, m-1$, will allow us to establish a lower bound for $C_N$.

Let $i = 0$. Now, if $1 \leq s < k/2$ and $f(0,s) = \alpha_j$, then $1 + \gamma^s = \gamma^t 3^j$ for some $t$. Hence $1 + \gamma^{k-s} = 1 + \gamma^{-s} = \gamma^{-s}(\gamma^s + 1) = \gamma^{t-s}3^j$, and so $f(0, k-s) = f(0,s)$. Furthermore, $f(0,k/2) = k$ and $f(0,0)$ is a basis element, whence $f(0,0) \neq f(0,k/2)$. We have the following three cases:

(A) If $k \equiv 0 \pmod 3$, then we claim that $C_0 + E_0/2 \geq k-1$. As we mentioned above the sum (1) for $i = 0$ produces

$$f(0,1) = f(0,k-1),\ \ldots,\ f(0,k/2-1) = f(0,k/2+1),\ f(0,0),\ f(0,k/2) = k.$$

If $f(0,0), f(0,1), \ldots, f(0,k/2-1)$ are pairwise distinct, then we have $C_0 + E_0/2 = k-1$, and if they are not pairwise distinct then $C_0$ will decrease while there wll be an increase in $E_0$. It is easy to see that the increment in $E_0/2$ will be greater than or equal to the decrement in $C_0$. From this the claim follows. Also it is easy to see that $C_0 + E_0/2 = k-1$ if and only if there are no distinct $s_1$, $s_2$, $s_3$, $s_4$ such that $f(0,s_1) = \cdots = f(0,s_4)$.

(B) If $k \equiv 1 \pmod 3$, then $\mathrm{Tr}(\alpha) = -1 \equiv -k \pmod 3$. From the fact that the trace is the sum of the basis elements and an argument similar to above we obtain $C_0 + E_0/2 \geq m + k/2 - 2$. Furthermore we see that $C_0 + E_0/2 = m + k/2 - 2$ if and only if there are no distinct $s_1$, $s_2$, $s_3$ such that $f(0,s_1) = f(0,s_2) = f(0,s_3)$.

(C) If $k \equiv 2 \pmod 3$, then $\mathrm{Tr}(\alpha) = -1 \equiv k \pmod 3$. Similar arguments as above lead to $C_0 + E_0/2 \geq m$. Again it is easily seen that $C_0 + E_0/2 = m$ if and only if there are no distinct $s_1$, $s_2$, $s_3$ such that $f(0,s_1) = f(0,s_2) = f(0,s_3)$.

Using the inequalities we have obtained for $C_0$, $C_1, \ldots, C_{m-1}$, it suffices to compute $E_0 + E_1 + \cdots + E_{m-1}$ in order to obtain a lower bound for $C_N = \sum_{i=0}^{m-1} C_i$. This is done in the following through a double counting argument.

A triple $(i, s', s'')$ for $E_i$ exists if and only if there is some $j$ with

$$1 + \gamma^{s'} 3^i \equiv \gamma^j (1 + \gamma^{s''} 3^i) \pmod{r} \quad \text{or} \quad \gamma^{s''} 3^i (\gamma^{s'-s''} - \gamma^j) \equiv \gamma^j - 1 \pmod{r}.$$

Now $\gamma^j - 1$ and $\gamma^{s'-s''} - \gamma^j$ cannot both be zero because otherwise $s' = s''$. For a given $j$ and $s' - s''$, there is either no solution or exactly one solution $(i, s'')$. Solutions are obtained for $0 < j < k$ and $s' - s'' \notin \{0, j\}$, giving $(k-1)(k-2)$ triples $(i, s', s'')$.

The claim about the count of $-1$ entries follows from the fact that $\sum_{i=0}^{m-1} E_i = (k-1)(k-2)$ and by examining the first row of the matrix $T$. □

**Corollary 4.** *If $k = 4$, then $C_N = 5m - 7$.*

*Proof.* We verify that conditions (i) and (iii) of Theorem 3 are satisfied. Since $k = 4$, we have $f(0,1) = f(0,3)$ and $E_0 + \cdots + E_{m-1} = 6$. Thus there are no $i \geq 1$ and distinct $s_1$, $s_2$, $s_3$ such that $f(i, s_1) = f(i, s_2) = f(i, s_3)$. Suppose now that $f(0,0) = f(0,1)$. Then $1 + \gamma^0 = \gamma^\ell (1 + \gamma)$ for some $\ell$. Squaring both sides gives $4 = \pm 2\gamma$, and hence $2 = \pm\gamma$. Squaring again yields $4 = \gamma^2 = -1$ which is impossible if $m > 1$. Hence $f(0,0) \neq f(0,1)$ and the result follows. □

The lower bound of Theorem 3 is not always met with equality when $k \geq 6$. A computer search found that the values $(m, k)$ for which $k \leq 26$ is even, $m \in [k, 1000]$ is prime, a type $k$ GNB for $\mathbb{F}_{3^m}$ exists, but the lower bound of Theorem 3 is not met with equality are $(17, 14)$, $(53, 20)$, $(31, 22)$, and $(103, 24)$.

The proof of Theorem 3 yields "structure" results concerning the matrix $T_0$ that can lead to significant computational savings when $k$ is 2 or 4. The basic idea is that $T_0$ can be written as $P + Q$ where the total number of nonzero entries is essentially unchanged, but the multiplication $A^{(\ell)} P B^{(\ell)'}$ is independent of $\ell$. The complexity of the multiplication is then essentially the number of nonzero terms in $Q$.

This type of decomposition was shown in $\mathbb{F}_{2^m}$ for optimal normal bases of type 1 by Hasan, Wang, and Bhargava [17]. For their case, the corresponding $T_0$ has $2m-1$ nonzero entries, and $Q$ has $m - 1$ nonzero entries. Exploiting the decomposition gives significant speed (and possibly storage) improvements [26, 10]. However, this decomposition is for type 1 bases, and so $m$ is necessarily even.

The following decomposition result is obtained for characteristic 3. For $k = 2$, the multiplication complexity is essentially reduced from $3m$ to $2m$. This result can be applied, for example, to the Ning-Yin multiplication presented in [14]. For $k = 4$, the multiplication complexity is essentially reduced from $5m$ to $4m$.

**Theorem 5.** *Let $T_0$ correspond to a GNB of type $k$ for $\mathbb{F}_{3^m}$ with $m > k$. If $k = 2$, then $T_0 = I + Q$ where $Q$ has $2m - 1$ nonzero entries, each of which is 1. If $k = 4$, then $T_0 = -I + Q$ where $Q$ has $4m - 4$ nonzero entries, three of which are $-1$.*

*Proof.* The entries of $T_0$ are obtained from $T$ via $t^{(0)}_{i,j} = t_{i-j,-j}$ and hence the diagonal entries of $T_0$ are $t^{(0)}_{j,j} = t_{0,-j}$.

For $k = 2$, we have $f(0,0) = \alpha_\ell$ for some $\ell$, and $f(0,1) = 2$. Hence $m - 1$ of the $t_{0,-j}$ entries are 1 and one entry is $-1$. We can thus write $T_0 = I + Q$ where $Q$

receives a "correction term" corresponding to the $-1$ entry. It is easy to see that $Q$ will then have $2m-1$ nonzero entries.

For $k=4$, in the proof of Corollary 4 we showed that there cannot be an $i$ with $f(i,s)$ constant for three distinct $s$. Thus from $f(0,1)=f(0,3)$ and $f(0,2)=4$, we have $C_0 = m-1$ where $m-2$ entries are $-1$ and one entry is 1. The result then follows in a fashion similar to $k=2$ using the fact that $C_N = 5m-7$. □

A consequence of Theorem 5 is that a GNB of type 2 over $\mathbb{F}_{3^m}$ is essentially optimal in the sense that the complexity of the multiplication is effectively $2m-1$ (since $A^{(\ell)}IB^{(\ell)'} = A \cdot B$ is independent of $\ell$ and essentially cost-free).

## 4. Implementation notes and timings

In this section, we provide details on the implementation along with timings on a Pentium 4, a common platform chosen for such comparisons. This processor is briefly described as 32-bit with extensions for wider operations in special registers, and has relatively few general-purpose registers. Compared to the preceding generation Pentium III processors, the instruction pipeline is deeper and penalties are larger for branch misprediction [19].

The implementation language was C, and only general-purpose registers were employed. Cooperating with processor characteristics and compiler optimizing peculiarities can be a difficult task, and our efforts in this area were modest. In particular, the GNU C compiler can be weaker on register allocation strategies, and favours scalars over structures and arrays (even when array indices are known at compile-time). Limited effort was applied to cooperate with such weaknesses, but the timings in Table 1 show that significant differences between compiler performance remain.

4.1. **Field multiplication.** For polynomial multiplication, we employed a "comb" multiplier [22] suitably modified for characteristic 3. We used width $w=3$, which extracts 3 bits each of $a^0$ and $a^1$ at each step. Only 27 (rather than $2^6=64$) distinct values are possible due to the representation. A lookup was performed on the 6 bits in order to select from the table of (data-dependent) precomputation. Since half the elements are obtained by simple negation, precomputation is less expensive than it may appear. For fields of sufficient size, a shallow-depth Karatsuba split was used. At smaller field sizes this need not be faster; the results vary by platform, but typically the times are competitive with a "full comb" and the split has the advantage of less code size and dynamic memory consumption. For example, on the test platform, a full comb on eight 32-bit word pairs (e.g., $\mathbb{F}_{3^{239}}$) is 8-18% faster than a depth-1 split (giving three 4-word-pair multiplications).

Normal basis multiplication uses the precomputation strategy of Ning and Yin [24]. The basic idea is to calculate rotations required in a (data-dependent) precomputation phase to reduce costs in the main evaluation. For low-complexity Gaussian normal bases, the multiplication matrix is sparse and of regular structure, and the corresponding algorithm is relatively simple to code. Granger et al. [14] adapt the Ning and Yin algorithm directly. Our implementation apparently differs

TABLE 1. Timings (in $\mu$s) for field arithmetic on a 2.4 GHz Pentium 4.[a]

| | Polynomial Basis | | | | | | Type 2 Normal Basis | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | add | mult | $a^3$ | $a^{1/3}$ | invert by exp | invert by EEA | mult | mult Thm 5 | mult ring map[b] | $a^3$ or $a^{1/3}$ | invert by exp |
| $\mathbb{F}_{3^{239}} = \mathbb{F}_3[x]/(x^{239}+x^{24}-1)$ | | | | | | | | | | | |
| gcc | .05 | 5.0 | .32 | 1.6[c] | 137[d] | 55 | 21.0 | 16.2 | 13.9 | .04 | 195[d] |
| icc | .04 | 4.2 | .30 | 1.2[c] | 122[d] | 46 | 17.2 | 14.2 | 11.5 | .02 | 171[d] |
| GPS[e] | .69 | 23.0 | 1.59 | 19.3 | | 159 | 60.9 | | | .60 | 14182 |
| $\mathbb{F}_{3^{509}} = \mathbb{F}_3[x]/(x^{509}-x^{477}+x^{445}+x^{32}-1)$ | | | | | | | | | | | |
| gcc | .09 | 15.5 | .70 | 2.5[f] | 575[g] | 213 | 98.7 | 74.5 | 58.1 | .07 | 1034[g] |
| icc | .07 | 12.8 | .66 | 1.7[f] | 508[g] | 190 | 74.3 | 58.8 | 52.0 | .04 | 829[g] |
| $\mathbb{F}_{2^{1223}} = \mathbb{F}_2[x]/(x^{1223}+x^{255}+1)$ | | | | | | | | | | | |
| gcc | .06 | 17.9 | | | | | | | | | |
| icc | .06 | 15.6 | | | | | | | | | |

[a]Compilers are GNU C (gcc) 3.3 and Intel C (icc) 6.0. Timings done under Linux/x86.
[b]Map to $\mathbb{F}_3[x]/((x^{mk+1}-1)/(x-1))$ and use a modified comb polynomial multiplication [10]. Fields here have $k=2$ (the type of the Gaussian normal basis).
[c]Sparse multiplication; $x^{1/3}$ has 23 nonzero terms and $x^{2/3}$ has 9 nonzero terms.
[d]Addition-chain with 12 multiplications.
[e]Timings in [14] are given for a 2.8 GHz Pentium 4 running Linux with gcc 3.3; times here are obtained by scaling linearly to 2.4 GHz.
[f]Sparse multiplication; $x^{1/3}$ has 6 nonzero terms and $x^{2/3}$ has 3 nonzero terms.
[g]Addition-chain with 14 multiplications.

in the order of evaluation in that our outer loop is on the rows of the multiplication matrix, which reduces the number of lookups. We also give timings for the reduced-complexity version given by Theorem 5.

The "ring mapping" approach is detailed in [10]; only an outline is given here. For a field $\mathbb{F}_{p^m}$ with a type $k$ normal basis, there is a fast mapping $\phi$ from $\mathbb{F}_{p^m}$ to the ring $\mathbb{F}_p[x]/((x^{mk+1}-1)/(x-1))$. The basic idea is to perform normal basis multiplication by mapping into the ring and applying fast polynomial-basis methods and then map back. A downside in this approach is the expansion by a factor $k$. However, the last $mk/2$ coefficients for elements in $\phi(\mathbb{F}_{p^m})$ are a mirror reflection of the first $mk/2$ [34]. Hence, it suffices to find half the coefficients in the ring product.

Each coefficient in $\phi(\cdot)$ or $\phi^{-1}(\cdot)$ can be obtained with a shift and mask.[1] The comb multiplier is defined for only a subset of the ring. However, the expansion in the mapping means that the method will be significantly more expensive than polynomial multiplication in the field. In particular, precomputation is for ring elements, and an entire ring element is "combed." On the positive side, only half the product in the ring is calculated, and reduction is especially simple.

[1]More precisely, in our representation for characteristic 3, a shift and mask is applied to a pair of words to obtain an output coefficient. For the mapping into the ring, only half the coefficients are found this way—the remainder are obtained at low cost by symmetry.

4.2. **Cubing and cube roots.** For cubing in polynomial representations, we used an 8-to-24-bit lookup table to "thin" coefficients and then reduced the result. This is analogous to the common method for squaring in binary fields, and cubing is similarly inexpensive. Cube roots were obtained by the method described in §2.2, with a 64-word lookup table to extract coefficients. The cost depends on the number of nonzero terms in $x^{1/3}$ and $x^{2/3}$.

For $m = 239$, we used the reduction polynomial $x^{239} + x^{24} - 1$ so that direct comparisons could be made to [14]. This choice gives 23 terms in $x^{1/3}$ and 9 terms in $x^{2/3}$. As noted in Example 1, this is not optimal, but nonetheless leads to fairly fast cube roots via sparse multiplication. For $m = 509$, there is no trinomial giving sparse $x^{1/3}$. We searched for irreducible tetranomial or pentanomial $x^{509} + p(x)$ giving the lowest combined weight for $x^{1/3}$ and $x^{2/3}$ subject to $\deg p \leq 509 - 32$. There are multiple candidate pentanomials, but $x^{509} - x^{477} + x^{445} + x^{32} - 1$ was chosen for the fortuitous spacing between powers that permits optimizations. The choice gives $x^{1/3}$ with 6 terms and $x^{2/3}$ with 3. A possible implementation downside is the high degree of $p(x)$, although this is not a significant issue here.

In a normal basis representation, cubing and cube roots are rotations. In our specific representation, this is rotation of a pair, an inexpensive operation.

4.3. **Inversion.** Inversion is performed via a Euclidean algorithm variant and also using exponentiation. Although Euclidean algorithm variants can be faster, coding for speed typically involves code expansion (to efficiently track lengths of operands, etc.). The method using exponentiation is short and easy to code once field multiplication is done.

For the Euclidean algorithm approach in characteristic 3, [16] and [14] employ the "binary" Euclidean algorithm. We adapted the "usual" Euclidean algorithm [15, Algorithm 2.48]. Unlike the binary Euclidean algorithm, explicit degree calculations are required. Some processors have hardware support that can aid in these calculations. The Intel Pentium family has "bit scan" instructions to find the left- or right-most 1 bit in a word, and we used an assembly language fragment to exploit this capability. A binary search can be used on processors without such support,[2] and in fact "bit scan" on the Pentium is less effective for our code here than in [11] for characteristic 2, in part because of the difference in characteristic and also that the characteristic 2 code uses more optimization via code expansion.

For inversion via exponentiation, we used Itoh-Tsujii methods (see §2.3). Rather than the direct recursive approach, a few multiplications can sometimes be saved by choosing short addition chains. We used the following chains:

$$\mathbb{F}_{3^{239}} : 1,\ 2,\ 3,\ 6,\ \ 8,\ 14,\ 28,\ 56,\ 112,\ 224,\ 238$$
$$\mathbb{F}_{3^{509}} : 1,\ 2,\ 4,\ 8,\ 12,\ 24,\ 28,\ 56,\ 112,\ 224,\ 252,\ 504,\ 508$$

These give inversion via 12 and 14 multiplications, respectively, saving a multiplication in each case over the direct recursive approach. (The corresponding inversion code using these chains has low resource requirements.)

[2]Sun recommends using a "population count" (*popc*) instruction to build a seven-instruction bit-scan on SPARC [33]. However, *popc* is implemented via a trap, and bit-scan will be faster via binary search.

4.4. **Analysis.** Table 1 shows that the times in [14, Table 4] are unnecessarily pessimistic (on this platform) for both polynomial basis and normal basis field arithmetic in characteristic 3. For the example field $\mathbb{F}_{3^{239}}$, multiplication times in a polynomial basis are approximately a factor 5 faster than reported in [14], in part due to algorithm differences.

For normal basis representations, significant improvement can be obtained for the type 2 case exhibited by $\mathbb{F}_{3^{239}}$ by exploiting Theorem 5 to reduce the complexity. Further improvement is obtained by the "ring mapping" approach. Our results are consistent with [14] in the sense that normal basis multiplication is sufficiently expensive relative to multiplication in a polynomial basis to discourage the use of normal basis representations in this environment.

Nonetheless, normal bases continue to be of interest in some environments, and choosing between the Ning-Yin approach and the ring mapping method will depend, in part, on the type $k$ of the Gaussian basis. Type 2 bases are of course advantageous in both methods, but larger type may be the only choice if supersingular curves with low cofactor are demanded. For example, in the range $239 < m < 487$, only $m = 317$ and $m = 353$ give supersingular curves over $\mathbb{F}_{3^m}$ with small cofactor, and the corresponding types are 26 and 14, resp.

The Ning-Yin precomputation requires $4m$ words of data-dependent storage, and this amount is not affected by the type of the basis. The method is especially easy to code, and (roughly speaking) the running time increases linearly with $k$, although the number and location of $-1$ entries in the multiplication matrix complicates performance tuning. In contrast, the ring mapping approach has an expansion by a factor $k$, although symmetry lessens the impact. For the test platform, there will be a threshold $k$ where multiplication will be fastest via conversion to polynomial basis representation (at cost equivalent to a few multiplications provided the conversion matrix is known). Of less practical importance, [14] note the "exceptionally high cost of inversion in normal bases;" however, in fact the methods of Itoh and Tsujii apply and inversion cost is relatively modest.

For curve arithmetic, [16] provide comparisons for a supersingular curve over $\mathbb{F}_{3^{97}}$ against a curve over $\mathbb{F}_{2^{241}}$, which offer similar security in the context of pairing-based cryptography (the corresponding embedding degrees are 6 and 4, respectively). In [16, Table 4], the best times for point multiplication favour the characteristic three case by roughly a factor 2. However, the scalar recodings selected are binary, ternary, and nonary, and this favours the characteristic three curve; in fact, the ternary and nonary methods are not useful for the characteristic two case. Since the nonary method is permitted storage for a few points of precomputation, a more meaningful comparison would involve a similar-storage width-$w$ NAF method in the characteristic two case. In fact, the calculation of the usual width-$w$ NAF adapts directly to the base 3 case, and so we'd recommend that the nonary method be replaced by a method employing a (base 3) width-3 NAF (which uses the same amount of storage).

## 5. Identity-based encryption

In this section we compare the speed of encryption and decryption for the Boneh-Franklin (BF) [7] and Sakai-Kasahara (SK) [27] identity-based encryption schemes at the 128-bit security level.

5.1. **Symmetric pairings.** Let $E$ be a supersingular elliptic curve defined over $\mathbb{F}_q$. Let $n$ be a prime divisor of $\#E(\mathbb{F}_q)$, and suppose that $n^2 \nmid \#E(\mathbb{F}_q)$. The embedding degree of $E$ (with respect to $n$) is the smallest positive integer $k$ such that $n \mid q^k - 1$. Let $P \in E(\mathbb{F}_q)$ be a point of order $n$, and let $\mu_n$ denote the order-$n$ cyclic subgroup of $\mathbb{F}_{q^k}^*$. The (reduced) Tate pairing is a bilinear map $\hat{e} : \langle P \rangle \times \langle P \rangle \to \mu_n$.

The three pairings we consider are described next. We let $m$, $s$, $c$ denote the cost of multiplication, squaring, and cubing in the base field $\mathbb{F}_q$, and let $M$, $S$, $C$ denote the cost of multiplication, squaring and cubing in the extension field $\mathbb{F}_{q^k}$. Also, we let $A$, $D$, $T$ denote the cost of point addition (using mixed affine-projective coordinates), point doubling (using projective coordinates), and point tripling (using projective coordinates) in $E(\mathbb{F}_q)$.

5.1.1. *Type I pairing.* Let $q = p$ be a 1536-bit prime such that $p \equiv 3 \pmod 4$ and $p + 1$ has a 256-bit low Hamming weight prime divisor $n$. Then the elliptic curve

$$E_1/\mathbb{F}_p : Y^2 = X^3 - 3X$$

is supersingular and $n \mid \#E_1(\mathbb{F}_p)$. The embedding degree of $E_1$ is $k = 2$. The extension field $\mathbb{F}_{p^2}$ is represented as $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 + 1)$, and a distortion map is $(x, y) \mapsto (-x, iy)$. We have $m \approx s$, $M \approx 3m$, $S \approx 2m$, $A \approx 3s + 8m \approx 11m$, and $D \approx 4s + 4m \approx 8m$. The Tate pairing, computed using the algorithm described by Scott [29], costs $4s + 8m + S + M$ per bit of $n$ (for the Miller operation) plus $5s + 5m$ per bit of $n$ (for the final exponentiation by $(p^2 - 1)/n$). If one of the two input points is fixed then, as observed in [29], precomputing 768 $\mathbb{F}_p$-elements can reduce the cost of the Miller operation to $m + S + M$ per bit of $n$.

5.1.2. *Type II pairing.* Let $q = 2^{1223}$, $\mathbb{F}_{2^{1223}} = \mathbb{F}_2[x]/(x^{1223} + x^{255} + 1)$, and

$$E_2/\mathbb{F}_{2^{1223}} : Y^2 + Y = X^3 + X.$$

Then $E_2$ is supersingular, and $\#E_2(\mathbb{F}_{2^{1223}}) = 5n$ where $n$ is a 1221-bit prime. The embedding degree of $E_2$ is $k = 4$. The extension field $\mathbb{F}_{q^4}$ is represented using tower extensions $\mathbb{F}_{q^2} = \mathbb{F}_q[u]/(u^2 + u + 1)$ and $\mathbb{F}_{q^4} = \mathbb{F}_{q^2}[v]/(v^2 + v + u)$. We have $M \approx 9m$ and $A \approx 9m$, while $s$, $S$ and $D$ are essentially free. Inversion of an element $\alpha \in \mu_n$ is also essentially free since $\alpha^{-1} = \alpha^{q^2}$. The BGhS [4] algorithm for computing the Tate pairing costs approximately $612 \times 7m$.

5.1.3. *Type III pairing.* Let $q = 3^{509}$, $\mathbb{F}_{3^{509}} = \mathbb{F}_3[x]/(x^{509} - x^{477} + x^{445} + x^{32} - 1)$, and

$$E_3/\mathbb{F}_{3^{509}} : Y^2 = X^3 - X + 1.$$

Then $E_3$ is supersingular, and $\#E_3(\mathbb{F}_{3^{509}}) = 7n$ where $n$ is a 804-bit prime. The embedding degree of $E_2$ is $k = 6$. The extension field $\mathbb{F}_{q^6}$ is represented using tower extensions $\mathbb{F}_{q^3}[u] = \mathbb{F}_q[u]/(u^3 - u - 1)$ and $\mathbb{F}_{q^6} = \mathbb{F}_{q^3}[v]/(v^2 + 1)$. We have $M \approx 18m$

and $A \approx 9m$, while $c$, $C$ and $T$ are essentially free. Inversion of an element $\alpha \in \mu_n$ is also essentially free since $\alpha^{-1} = \alpha^{q^3}$. The BGhS [4] algorithm for computing the Tate pairing costs approximately $255 \times 15m$.

5.2. **Boneh-Franklin and Sakai-Kasahara IBE.** Let $P \in E(\mathbb{F}_q)$ be a point of order $n$, and let $\hat{e} : \langle P \rangle \times \langle P \rangle \to \mu_n$ be a (symmetric) bilinear pairing. Let $H_1 : \{0,1\}^* \to \langle P \rangle$, $H_2 : \{0,1\}^\lambda \times \{0,1\}^\lambda \to [1, n-1]$, $H_3 : \mu_n \to \{0,1\}^\lambda$, $H_4 : \{0,1\}^\lambda \to \{0,1\}^\lambda$, $H_5 : \{0,1\}^* \to [1, n-1]$ be hash functions. The Key Generator's private key is $t \in_R [1, n-1]$, while its public key is $T = tP$.

5.2.1. *Boneh-Franklin IBE..* Party $A$'s private key is $d = tQ$, where $Q = H_1(\mathrm{ID}_A)$.

To encrypt a message $m \in \{0,1\}^\lambda$ for $A$, a party $B$ does the following: Select $\sigma \in_R \{0,1\}^\lambda$, and compute $Q = H_1(\mathrm{ID}_A)$, $r = H_2(\sigma, m)$, $R = rP$, $V = \sigma \oplus H_3(\hat{e}(T,Q)^r)$, and $c = m \oplus H_4(\sigma)$. $B$ sends $(R, V, c)$ to $A$.

To decrypt, $A$ computes $\sigma = V \oplus H_3(\hat{e}(d, R))$, $m = c \oplus H_4(\sigma)$, and $r = H_2(\sigma, m)$. Finally, $A$ accepts $m$ provided that $R = rP$.

The BF scheme requires a hash function $H_1 : \{0,1\}^* \to \langle P \rangle$. For Type I pairings, $H_1$ can be implemented by first hashing to a point $Q'$ in $E(\mathbb{F}_p)$, and then multiplying $Q'$ by the cofactor $h = \#E(\mathbb{F}_p)/n$. As noted by Scott [29], the cofactor multiplication can be avoided; thus the essential cost of hashing is a square-root computation in $\mathbb{F}_p$. Square roots in $\mathbb{F}_p$ can be obtained by an exponentiation to the power $(p+1)/4$, an operation which requires about 1806 $\mathbb{F}_p$-multiplications using width-5 sliding windows. The hash function $H_1$ for the Type II pairing (and the Type III pairing) is relatively inexpensive since square roots (resp. cube roots) can be efficiently computed, and since the cofactor $h$ is small.

The dominant operations in BF encryption are the point multiplication $rP$ where $P$ is fixed, and the computation of $\gamma = \hat{e}(T,Q)^r$ (plus a square-root computation for Type I pairings). For Type I pairings, the fastest way to compute $\gamma$ is to first evaluate the Tate pairing $\hat{e}(T,Q)$ (where $T$ is fixed), and then perform the exponentiation to the power $r$ (where the base element $\hat{e}(T,Q)$ is unknown in advance). For Type II and III pairings, $\gamma$ should be computed by first computing $rT$ (where $T$ is fixed), and then evaluating $\hat{e}(rT, Q)$. The dominant operations in BF decryption are the Tate pairing evaluation $\hat{e}(d, R)$ and the point multiplication $rP$ where the points $d$ and $P$ are fixed.

5.2.2. *Sakai-Kasahara IBE..* Party $A$'s private key is $d = (1/(H_5(\mathrm{ID}_A) + t)P$.

To encrypt a message $m \in \{0,1\}^\lambda$ for $A$, a party $B$ does the following: Select $\sigma \in_R \{0,1\}^\lambda$, and compute $Q = H_5(\mathrm{ID}_A)P + T$, $r = H_2(\sigma, m)$, $R = rQ$, $V = \sigma \oplus H_3(\hat{e}(P,P)^r)$, and $c = m \oplus H_4(\sigma)$. $B$ sends $(R, V, c)$ to $A$.

To decrypt, $A$ computes $\sigma = V \oplus H_3(\hat{e}(d, R))$, $m = c \oplus H_4(\sigma)$, and $r = H_2(\sigma, m)$. Finally, $A$ accepts $m$ provided that $R = rQ$.

The dominant operations in SK encryption are the point multiplication $H_5(\mathrm{ID}_A)P$ where the base point $P$ is fixed, the point multiplication $rQ$ where the base point $Q$ is unknown in advance, and the exponentiation $\hat{e}(P,P)^r$ where the base element $\hat{e}(P,P)$ is fixed. The dominant operations in SK decryption are the same as for BF decryption.

TABLE 2. Cost (number of $\mathbb{F}_q$-multiplications) of the Tate pairing, point multiplication $rP$ in $E(\mathbb{F}_q)$, and exponentiation $\alpha^r$ in $\mu_n$. The bitlength of $n$ is denoted by $\ell$.

| Pairing type | $\mathbb{F}_q$ | $\ell$ | Tate pairing | $rP$ $P$ unknown | $rP$ $P$ fixed | Exp in $\mu_n$ $\alpha$ uknown | Exp in $\mu_n$ $\alpha$ fixed |
|---|---|---|---|---|---|---|---|
| I | 1536-bit $p$ | 256 | $6912/4096^a$ | $2602^b$ | $745^c$ | $512^d$ | $199^c$ |
| II | $\mathbb{F}_{2^{1223}}$ | 1221 | 4284 | $1895^b/447^e$ | $1832^b/384^e$ | $1895^b/447^e$ | $1832^b/384^e$ |
| III | $\mathbb{F}_{3^{509}}$ | 804 | 3825 | $1259^f/498^g$ | $1016^f/324^g$ | $2518^f/996^g$ | $2032^f/648^g$ |

[a]Applicable when one of the input points is fixed; requires 144 Kbytes for precomputed values.
[b]5-NAF point multiplication/exponentiation (see Algorithm 3.36 of [15]) with $\ell$-bit multiplier $r$.
[c]Width-5 two-table comb method (see Algorithm 3.45 of [15]) with $\ell$-bit multiplier $r$.
[d]Lucas-method of exponentiation [29].
[e]5-NAF point multiplication/exponentiation with 256-bit multiplier $r$.
[f]4-NAF point multiplication/exponentiation with $\ell$-bit multiplier $r$.
[g]3-NAF point multiplication/exponentiation with 256-bit multiplier $r$.

5.3. **Costs.** Table 2 lists the approximate number of $\mathbb{F}_q$-multiplications needed to compute the Tate pairing, and to perform point multiplication and exponentiation in $\mu_n$ for the Type I, II and III pairings. We acknowledge that these raw multiplication counts do *not* include the cost of other field operations such as additions, square roots, cube roots, and inversions that are either relatively cheap or few in number. Nonetheless, these multiplication counts are reasonably accurate estimates of the actual running times of the operations listed in Table 2. For example, the timings for our implementation of the Tate pairing, point multiplication, and $\mu_n$-exponentiation for the Type II pairing are 4578, 2163, and 2037 $\mathbb{F}_{2^{1223}}$-multiplications, which are close to the estimated costs of 4284, 1895, and 1895 in Table 2. Similarly, the timings for our implementation of the Tate pairing, point multiplication, and $\mu_n$-exponentiation for the Type III pairing are 4359, 1602, and 2695 $\mathbb{F}_{3^{509}}$-multiplications, which are close to the estimated costs of 3825, 1259, and 2518 in Table 2.

Table 3 lists the multiplication costs of the dominant operations in encryption and decryption for the BF and SK IBE schemes. The costs have been normalized to $\mathbb{F}_p$-multiplications (where $p$ is a 1536-bit prime), using our timings of 12.8$\mu$s for multiplication in $\mathbb{F}_{3^{509}}$, 15.6$\mu$s for multiplication in $\mathbb{F}_{2^{1223}}$, and 26.5$\mu$s for a multiplication in $\mathbb{F}_p$ (the latter obtained using Mike Scott's MIRACL multiprecision library [30]).[3] The first set of (I,II,III) timings in Table 3 use full-length multipliers $r$ for the Type II and III pairings, and do not include any precomputation of the

[3]Timings were obtained on a 2.4 GHz Pentium 4 running Linux/x86. Compilers were GNU C (gcc) 3.3 for $\mathbb{F}_p$ and the Intel compiler (icc) 6.0 for the others. The expensive portions in the $\mathbb{F}_p$ multiplication are written in assembly, and times for these fragments are not affected by compiler selection.

MIRACL has optimized code for several operations on the Pentium 4, the processor used in this comparison. In particular, the timing was obtained using general-purpose registers and a multiplier that uses Karatsuba down to a specified operand-size threshold. The threshold $t$ is specified in terms of words and so that the modulus size is $t \cdot 2^n$ words for some $n$. Code size grows quadratically

TABLE 3. Normalized cost (in terms of $\mathbb{F}_p$-multiplications) of encryption and decryption for BF and SK IBE using the Type I, II and III pairings.

| | I[a] | II[b] | III[b] | I[c] | II[d] | III[d] |
|---|---|---|---|---|---|---|
| BF encrypt | 9975 | 4679 | 2829 | 7159 | 2974 | 2161 |
| BF decrypt | 7657 | 3600 | 2338 | 4841 | 2748 | 2004 |
| SK encrypt | 3546 | 3272 | 2080 | 3546 | 715 | 710 |
| SK decrypt | 7657 | 3600 | 2338 | 4841 | 2748 | 2004 |

[a]No precomputation for Tate pairing computations.
[b]Full length multipliers $r$.
[c]144 Kbytes of precomputed values for the Tate pairing computation.
[d]256-bit multipliers $r$.

Type I Tate pairing computation. The second set of (I,II,III) timings, on the other hand, use 256-bit multipliers $r$ for the Type II and II pairings, and allow for the 144 Kbytes of precomputed values that accelerate the Tate pairing computation for Type I pairings. Short $2t$-bit multipliers (where $t$ is the security level) instead of $\ell$-bit multipliers (where $\ell$ is the bitlength of $n$) have been used in some previous papers (e.g., [31] and [32]). The drawback of using short multipliers is that the BF [7] and SK [9] security proofs are no longer applicable.

## 6. Conclusions

We devised faster multiplication algorithms for characteristic three finite fields when elements are represented using a Gaussian normal bases. Despite our structure results and fast implementations, our analysis confirms the conclusions of previous papers that multiplication is faster when a polynomial basis representation is employed. We also compared the relative speed of the BF and SK IBE schemes at the 128-bit security levels when a pairing based on a supersingular elliptic curve is used. The Type III pairing (over $\mathbb{F}_{3^{509}}$) yields the fastest encryption and decryption operations, which are several times faster than with a Type I pairing (over a 1536-bit prime field). Moreover, when using a Type III pairing, SK encryption is about 3 times as fast as BF encryption, while SK and BF decryption have similar running times.

## References

[1] O. Ahmadi, D. Hankerson and A. Menezes, "Formulas for cube roots in $\mathbb{F}_{3^m}$", *Discrete Applied Mathematics*, 155 (2007), 260-270.

with $t$, and $t$ between 8 and 16 is reasonable on this platform. Hence, for 1536-bit primes, we chose $t = 12$.

The Pentium 4 has special-purpose "multi-media" (SSE2) registers that can be employed for field multiplication. Roughly speaking, the basic advantage for prime fields is additional registers that participate in multiplications and accumulation can be on 64 bits, and the advantage for characteristic 2 and 3 is wider operations. Multiplication in MIRACL for prime fields is nearly a factor 2 faster with these registers, and [15] reports similar acceleration for characteristic 2 fields (on a Pentium III via the SSE subset); similar techniques apply to characteristic 3.

[2] D. Ash, I. Blake and S. Vanstone, "Low complexity normal bases", *Discrete Applied Mathematics*, 25 (1989), 191-210.
[3] P. Barreto, "A note on efficient computation of cube roots in characteristic 3", Technical Report 2004/305, Cryptology ePrint Archive, 2004.
[4] P. Barreto, S. Galbraith, C. hÉigeartaigh and M. Scott, "Efficient pairing computation on supersingular abelian varieties", *Designs, Codes and Cryptography*, 42 (2007), 239-271.
[5] P. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order", *Selected Areas in Cryptography (SAC 2005)*, LNCS 3897 (2006), 319-331.
[6] I. Blake, X. Gao, A. Menezes, R. Mullin, S. Vanstone and T. Yaghoobian, *Applications of Finite Fields*, Kluwer, 1993.
[7] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing", *SIAM Journal on Computing*, 32 (2003), 586-615.
[8] X. Boyen and L. Martin, "Identity-based cryptography standard (IBCS) #1: Supersingular curve implementations of the BF and BB1 cryptosystems", IETF Internet Draft, December 2006.
[9] L. Chen and Z. Cheng, "Security proof of Sakai-Kasahara's identity-based encryption scheme", *Cryptography and Coding*, LNCS 3796 (2005), 442-459.
[10] R. Dahab, D. Hankerson, F. Hu, M. Long, J. López and A. Menezes, "Software multiplication using Gaussian normal bases", *IEEE Transactions on Computers*, 55 (2006), 974-984.
[11] K. Fong, D. Hankerson, J. López and A. Menezes, "Field inversion and point halving revisited", *IEEE Transactions on Computers*, 53 (2004), 1047-1059.
[12] S. Galbraith, K. Paterson and N. Smart, "Pairings for cryptographers", Technical Report 2006/165, Cryptology ePrint Archive, 2006.
[13] P. Grabher and D. Page, "Hardware acceleration of the Tate pairing in characteristic three", *Cryptographic Hardware and Embedded Systems – CHES 2005*, LNCS 3659 (2005), 398-411.
[14] R. Granger, D. Page and M. Stam, "Hardware and software normal basis arithmetic for pairing based cryptography in characteristic three", *IEEE Transactions on Computers*, 54 (2005), 852-860.
[15] D. Hankerson, A. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag, 2004.
[16] K. Harrison, D. Page and N. Smart, "Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems", *LMS Journal of Computation and Mathematics*, 5 (2002), 181-193.
[17] M. Hasan, M. Wang and V. Bhargava, "A modified Massey-Omura parallel multiplier for a class of finite fields", *IEEE Transactions on Computers*, 42 (1993), 1278-1280.
[18] F. Hess, N. Smart and F. Vercauteren, "The eta pairing revisited", *IEEE Transactions on Information Theory*, 52 (2006), 4595-4602.
[19] Intel Corporation, *IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture*, 2002. Number 245470-007, available from http://developer.intel.com.
[20] T. Kerins, W. Marnane, E. Popovici and P. Barreto, "Efficient hardware for the Tate pairing calculation in characteristic three", *Cryptographic Hardware and Embedded Systems – CHES 2005*, LNCS 3659 (2005), 412-426.
[21] A. Lenstra, "Unbelievable security: Matching AES security using public key systems", *Advances in Cryptology–Asiacrypt 2001*, LNCS 2248 (2001), 67-86.
[22] J. López and R. Dahab, "High-speed software multiplication in $\mathbb{F}_{2^m}$", *Progress in Cryptology–Indocrypt 2000*, LNCS 1977 (2000), 203-212.
[23] A. Miyaji, M. Nakabayashi and S. Takano, "New explicit conditions of elliptic curve traces for FR-reduction", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E84-A (2001), 1234-1243.
[24] P. Ning and Y. Yin, "Efficient software implementation for finite field multiplication in normal basis", *Information and Communications Security 2001*, LNCS 2229 (2001), 177-189.
[25] D. Page and N. Smart, "Hardware implementation of finite fields of characteristic three", *Cryptographic Hardware and Embedded Systems – CHES 2002*, LNCS 2523 (2003), 529-539.

[26] A. Reyhani-Masoleh, "Efficient algorithms and architectures for field multiplication using Gaussian normal bases", *IEEE Transactions on Computers*, 55 (2006), 34-47.
[27] R. Sakai and M. Kasahara, "ID based cryptosystems with pairing on elliptic curve", Technical Report 2003/054, Cryptology ePrint Archive, 2003.
[28] O. Schirokauer, "The number field sieve for integers of low weight", Technical Report 2006/107, Cryptology ePrint Archive, 2006.
[29] M. Scott, "Computing the Tate pairing", *Topics in Cryptology–CT-RSA 2005*, LNCS 3376 (2005), 293-304.
[30] M. Scott, *MIRACL – Multiprecision Integer and Rational Arithmetic C Library*, http://www.computing.dcu.ie/~mike/miracl.html.
[31] M. Scott, "Implementing cryptographic pairings", preprint, 2006.
[32] M. Scott, N. Costigan and W. Abdulwahab, "Implementing cryptographic pairings on smartcards", *Cryptographic Hardware and Embedded Systems – CHES 2006*, LNCS 4249 (2006), 134-147.
[33] D. Weaver and T. Germond, editors, *The SPARC Architecture Manual, Version 9*, Prentice Hall, 1994.
[34] H. Wu, A. Hasan, I. Blake and S. Gao, "Finite field multiplier using redundant representation", *IEEE Transactions on Computers*, 51 (2002), 1306-1316.

Department of Electrical and Computer Engineering, University of Toronto
*E-mail address*: oahmadid@comm.utoronto.ca

Department of Mathematics and Statistics, Auburn University
*E-mail address*: hankedr@auburn.edu

Department of Combinatorics & Optimization, University of Waterloo
*E-mail address*: ajmeneze@uwaterloo.ca