Arithmetic Operations in Finite Fields of Medium Prime Characteristic Using the Lagrange Representation

Jean-Claude Bajard, *Member*, *IEEE*, Laurent Imbert, *Member*, *IEEE*, and Christophe Nègre

Abstract—In this paper, we propose a complete set of algorithms for the arithmetic operations in finite fields of prime medium characteristic. The elements of the fields $\mathbb{F}_{p^k}$ are represented using the newly defined Lagrange representation, where polynomials are expressed using their values at sufficiently many points. Our multiplication algorithm, which uses a Montgomery approach, can be implemented in $O(k)$ multiplications and $O(k^2 \log k)$ additions in the base field $\mathbb{F}_p$. For the inversion, we propose a variant of the extended Euclidean GCD algorithm, where the inputs are given in the Lagrange representation. The Lagrange representation scheme and the arithmetic algorithms presented in the present work represent an interesting alternative for elliptic curve cryptography.

Index Terms—Finite field arithmetic, optimal extension fields, Newton interpolation, Euclidean algorithm, elliptic curve cryptography.

1 Introduction

Finite field arithmetic is an important prerequisite for many scientific applications. The main motivation of this work is in the context of elliptic curve cryptography (ECC), which was proposed independently by Koblitz [1] and Miller [2] in 1985 as an alternative to the existing group-based algorithms [3]. Since then, an immense amount of research has been dedicated to securing and accelerating its implementations. ECC has quickly received a lot of attention because of smaller key-length and increased theoretical robustness (there is no known subexponential algorithm to solve the ECDLP problem, which is the foundation of ECC). The relatively small key-size (the security provided by a 160-bit key is equivalent to an 80-bit symmetric-key for block ciphers or a 1,024-bit RSA modulus) is a major advantage for devices with limited hardware resources such as smartcards, cell phones, or PDAs. As a result, ECC has kept receiving commercial acceptance and has been included in numerous standards such as IEEE 1363 [4] and NIST FIPS 186.2 [5]. These standards recommend carefully chosen elliptic curves, allowing for secure and efficient implementations over either (large) prime or binary fields. With most of the research following the standard recommendations, alternative solutions and obvious areas of interest have received very little attention.

- *J.-C. Bajard is with the LIRMM, CNRS/UM2, 161 rue Ada, 34392 Montpellier cedex 5, France. E-mail: bajard@lirmm.fr.*
- *L. Imbert is with the LIRMM, CNRS/UM2, 161 rue Ada, 34392 Montpellier cedex 5, France, and the ATIPS labs. and the CISaC, University of Calgary, 2500 University Drive N.W., T2N 1N4, Calgary, AB, Canada. E-mail: Laurent.Imbert@lirmm.fr.*
- *C. Nègre is with the Équipe DALI, Laboratorie LP2A, Université de Perpignan, 52 avenue Paul Alduy, F-66860 Perpignan Cedex, France. E-mail: Christophe.Negre@univ-perp.fr.*

Manuscript received 5 Apr. 2005; revised 20 Dec. 2005; accepted 13 Feb. 2006; published online 20 July 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0097-0405.

The most studied alternatives to the recommended prime and binary fields are the optimal extension fields (OEF) $\mathbb{F}_{p^k}$, proposed by Bailey and Paar in [6], [7], where p is a prime of the form $2^n - c$, with $|c| \leq n/2$, and there exists an irreducible polynomial over $\mathbb{F}_p$ of the form $X^k - \omega$, with $\omega \in \mathbb{F}_p$. If $c = \pm 1$, then the OEF is said to be of *Type I* and, if $\omega = 2$, then the OEF is said to be of *Type II*. (See [8] for more details and examples of optimal extension fields of Types I and II.)

Elliptic curves defined over fields of characteristic three have recently been considered by Smart and Westwood [9]. Their conclusion is that such curves "could offer greater performance than currently perceived by the community." Already, in 1999, Smart proposed a generalization of Solinas' work [10] on anomalous binary curves (best known as Koblitz curves [11]) and explained how to use a Frobenius expansion method to speed up the scalar multiplication over fields of odd characteristic.

In this paper, we propose a complete set of arithmetic operations in finite extension fields of medium prime characteristic. We consider the fields $\mathbb{F}_{p^k} \simeq \mathbb{F}_p[X]/(N)$, where N is a monic irreducible polynomial of degree k, called the reduction polynomial. In other words, this isomorphism tells us that the elements of $\mathbb{F}_{p^k}$ can be expressed as the set of all polynomials of degree at most $k-1$, with coefficients in $\mathbb{F}_p$. The arithmetic operations (addition, multiplication) are carried out using polynomial arithmetic modulo N.

Compared to the previous works, the novelty comes from the fact that the operands are represented in the so-called *Lagrange representation* (LR), which means that our polynomials are not represented by their coefficients, but, rather, by using their values at sufficiently many points. We define the Lagrange representation and present the basic operations in LR in Section 2. After recalling Montgomery and the OEF algorithms for multiplication over a finite field $\mathbb{F}_{p^k}$ in Section 3, we propose a modified Montgomery algorithm

0018-9340/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society

for the multiplication modulo N in Section 4. We propose several variants and optimization strategies which can lead to very efficient implementations. For example, under certain conditions, it only requires a linear number of multiplications in $\mathbb{F}_p$. In Section 5, we propose a Lehmer-based GCD algorithm [12] for computing the inverse of an element $A \in \mathbb{F}_{p^k}$ modulo N. We discuss the advantages of the Lagrange representation in the context of ECC in Section 6.

2 Lagrange Representation

In this section, we define the Lagrange representation (LR) and we briefly present the basic operations in LR, namely, addition/subtraction, multiplication,[1] and the conversions between the coefficient-based and the Lagrange representations.

The Lagrange representation scheme can be defined by considering a special case of the Chinese Remainder Theorem (CRT). Let us consider the polynomial $\Psi(X) = \prod_{i=1}^{k}(X - e_i)$, where $e_i \in \mathbb{F}_p$ for $1 \leq i \leq k$ and $e_i \neq e_j$ for $i \neq j$. (Note that this clearly implies $k < p$; note also that since we shall also need $\Psi'(X) = \prod_{i=1}^{k}(X - e'_i)$ such that $\gcd(\Psi, \Psi') = 1$ for our multiplication algorithm, the condition will become $2k < p$.) For any arbitrary $U \in \mathbb{F}_p[X]$, we have $U \bmod (X - e_i) = U(e_i)$. By extension, the ring isomorphism given by the Chinese Remainder Theorem,

$$\begin{aligned} \mathbb{F}_p[X]/(\Psi) &\longrightarrow \mathbb{F}_p[X]/(X - e_1) \times \cdots \times \mathbb{F}_p[X]/(X - e_k) \\ U &\longmapsto (U \bmod (X - e_1), \ldots, U \bmod (X - e_k)), \end{aligned} \tag{1}$$

is the evaluation map of the polynomial U at all points $e_1, \ldots, e_k$: $U \mapsto (U(e_1), \ldots, U(e_k))$. Moreover, if $\deg U < k$, then the polynomial

$$U(X) = \sum_{i=1}^{k} u_i \, l_i(X) \tag{2}$$

is the (unique) Lagrange interpolation polynomial satisfying $u_i = U(e_i)$ for $1 \leq i \leq k$, where the l_is are the Lagrange interpolants such that, for all $1 \leq i \leq k$,

$$l_i(X) = \prod_{j=1, j \neq i}^{k} \frac{X - e_j}{e_i - e_j}. \tag{3}$$

In this case, the CRT is equivalent to the Lagrange interpolation theorem. In fact, it is useful to think of the CRT as a generalization of interpolation. What both the CRT and Lagrange interpolation theorem tell us is that the interpolation polynomial is unique modulo $\Psi = \prod_{i=1}^{k}(X - e_i)$ so that there is exactly one polynomial $U \in \mathbb{F}_p[X]$ of degree less than $\deg \Psi = k$, which satisfies $U(e_i) = u_i$ for $i = 1, \ldots, k$. In the following, we use this property in order to represent the elements of $\mathbb{F}_{p^k}$.

Definition 1 (Lagrange representation). *Let $U \in \mathbb{F}_p[X]$ with $\deg U < k$ and $\Psi = \prod_{i=1}^{k}(X - e_i)$, where $e_i \in \mathbb{F}_p$ for $1 \leq i \leq k$ and $e_i \neq e_j$ for $i \neq j$. If $u_i = U(e_i)$ for $1 \leq i \leq k$, we define the so-called Lagrange representation (LR) of U modulo Ψ as*

$$\mathrm{LR}_\Psi(U) = (u_1, \ldots, u_k). \tag{4}$$

1. A ring multiplication, different from the field operation.

One recognized advantage of the CRT is the fact that the costly arithmetic modulo Ψ can be split into several independent arithmetic units, each performing its arithmetic modulo a very simple polynomial (in our case, a binomial of degree one), thus leading to straightforward parallel implementations. For example, additions, subtraction, and multiplications in the ring $\mathbb{F}_p[X]/(\Psi)$ (i.e., modulo Ψ) can be performed independently for each modulus. Indeed, if $\mathrm{LR}_\Psi(U) = (u_1, \ldots, u_k)$ and $\mathrm{LR}_\Psi(V) = (v_1, \ldots, v_k)$, then we have

$$\mathrm{LR}_\Psi(U \diamond V) = (u_1 \diamond v_1, \ldots, u_k \diamond v_k), \tag{5}$$

where $\diamond$ belongs to $\{+, -, \times\}$ and $u_i \diamond v_i$ is performed over $\mathbb{F}_p$, i.e., modulo p. Note that the CRT also provides a natural, implicit way to perform the arithmetic modulo Ψ, which we shall exploit in our field multiplication algorithm presented in Section 4.

Since $\deg \Psi = \deg N = k$, field additions and subtractions are equivalent to their ring counterparts and can be easily computed using (5). The conversion from the coefficient-based representation into LR is the evaluation map of a polynomial at many points, with all the operations performed in $\mathbb{F}_p$. The conversion back from LR to the coefficient-based representation is an interpolation step that can be computed using (2) and (3). Fast multipoint evaluation and fast interpolation methods are covered in detail in [13, chapter 10]. Note that, since all the arithmetic operations can be performed in LR, the conversions steps are only required at the very beginning and the very end of the algorithms and do not affect the global computational cost. In some cases, it is even possible to avoid these conversion steps by performing all the computations in the Lagrange representation (see Section 6).

3 Background

In this section, we briefly recall the Montgomery modular multiplication algorithm for integers and its straightforward extension to finite extension fields. Then, we present a modified Montgomery multiplication algorithm, where the elements of the finite field $\mathbb{F}_{p^k}$ are represented in the Lagrange Representation (LR).

3.1 Montgomery Multiplication in $\mathbb{F}_{p^k}$

Montgomery modular multiplication for integers [14] —which, given a, b, n, and r such that $\gcd(r, n) = 1$, computes $abr^{-1} \bmod n$ without performing any division— has been generalized to binary fields $\mathbb{F}_{2^k}$ by Koç and Acar [15]. Their solution is a direct adaptation of the original Montgomery algorithm, where the polynomial X^k plays the role of the Montgomery factor r. Given $A, B \in \mathbb{F}_{2^k}$, it computes $ABX^{-k} \bmod N$, where N is the monic irreducible polynomial of degree k in $\mathbb{F}_2[X]$ which defines the field.

We first remark that Koç and Acar's algorithm easily extends to any extension field $\mathbb{F}_{p^k}$. In the polynomial basis representation, the elements of $\mathbb{F}_{p^k}$ can be modeled as the polynomials in $\mathbb{F}_p[X]$ of degree at most $k - 1$. Let N be a monic irreducible polynomial of degree k chosen as the reduction polynomial. We define $\Psi = X^k$ such that $\gcd(\Psi, N) = 1$. Then, given $A, B \in \mathbb{F}_p[X]/(N)$, Algorithm 1 can be used to compute $AB\Psi^{-1} \bmod N$.

Algorithm 1 Montgomery Multiplication over $\mathbb{F}_{p^k}$
Input: $A, B \in \mathbb{F}_p[X]$, with $\deg A, \deg B \leq k-1$; a monic irreducible polynomial $N \in \mathbb{F}_p[X]$, with $\deg N = k$; $\Psi = X^k$
Output: $AB\Psi^{-1} \bmod N$
1: $Q = -A \times B \times N^{-1} \bmod \Psi$
2: $R = (A \times B + Q \times N)/\Psi$

In this case, choosing $\Psi = X^k$ seems to be a perfect choice since the reduction modulo X^k (in Step 1) and the division by X^k (in Step 2) are easily implemented. Indeed, given two polynomials $U, V \in \mathbb{F}_p[X]$, with $\deg U, \deg V < k$, we compute $(U \times V) \bmod X^k$ by ignoring the coefficients of $U \times V$ of order larger than $k-1$. Similarly, $(U \times V)/X^k$ is given by the coefficients of $(U \times V)$ of order greater than or equal to k. These computations are easily expressed in terms of matrix operations.

Let us define

$$N = n_0 + n_1 X + \ldots + n_{k-1} X^{k-1} + X^k,$$

and N', the inverse of N modulo X^k, as

$$N' = N^{-1} \bmod X^k = n'_0 + n'_1 X + \ldots + n'_{k-1} X^{k-1}.$$

In Step 1 of Algorithm 1, we compute $Q = -ABN^{-1} \bmod \Psi$ as

$$Q = -\begin{pmatrix} n'_0 & 0 & \ldots & 0 \\ n'_1 & n'_0 & \ldots & 0 \\ \vdots & & \ddots & \\ n'_{k-1} & n'_{k-2} & \ldots & n'_0 \end{pmatrix} \begin{pmatrix} a_0 & 0 & \ldots & 0 \\ a_1 & a_0 & \ldots & 0 \\ \vdots & & \ddots & \\ a_{k-1} & a_{k-2} & \ldots & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{k-1} \end{pmatrix}. \quad (6)$$

Similarly, we evaluate $R = (A\,B + Q\,N)/\Psi$ as

$$R = \begin{pmatrix} 0 & a_{k-1} & \ldots & a_2 & a_1 \\ 0 & 0 & \ddots & & a_2 \\ \vdots & & & \ddots & \\ 0 & 0 & \ldots & 0 & a_{k-1} \\ 0 & 0 & \ldots & 0 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{k-2} \\ b_{k-1} \end{pmatrix} + \begin{pmatrix} 1 & n_{k-1} & \ldots & n_2 & n_1 \\ 0 & 1 & \ddots & & n_2 \\ \vdots & & \ddots & \ddots & \\ 0 & 0 & \ldots & 1 & n_{k-1} \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_{k-2} \\ q_{k-1} \end{pmatrix}. \quad (7)$$

Note that, because N is a monic polynomial of degree k, the diagonal of the second matrix in (7) is composed of ones.

The number of arithmetic operations over $\mathbb{F}_p$ is easily determined. The computation of Q in (6) requires $k(k+1)$ multiplications and $k(k-1)$ additions, whereas R in (7) is computed in $k(k-1)$ multiplications and $\frac{(k-1)(k-2)}{2} + \frac{k(k-1)}{2} + (k-1)$ additions. If M and A denote the costs of one multiplication and one addition in $\mathbb{F}_p$, respectively, the total cost of Algorithm 1 is

$$2k^2\,M + (2k^2 - 2k)\,A. \quad (8)$$

For most applications (including ECC), the finite field is fixed and we can reasonably assume that the reduction polynomial N and its inverse modulo X^k are known in advance. In this case, the multiplications by the n_is and n'_is in (6) and (7) can be simplified, using optimized algorithms for multiplication by a constant and by constant vectors [16]. The global cost of Algorithm 1 becomes

$$k^2\,M + k^2\,CM + (2k^2 - 2k)\,A, \quad (9)$$

where CM denotes the cost of one multiplication by a constant in $\mathbb{F}_p$.

3.2 Optimal Extension Fields

Optimal extension fields (OEFs) have been introduced by Bailey and Paar in [6], [7]. The main idea is to select the (prime) characteristic p of the field to closely match the underlying hardware characteristics and to simplify the arithmetic modulo p (for example, the Mersenne prime $p = 2^{31} - 1$ is a good choice for 32-bit architectures). Following the same idea, the reduction polynomial N of degree k is chosen to simplify the reduction modulo N as much as possible. The following definition is taken from [8]:

Definition 2 (OEF). *An* optimal extension field *(OEF) is a finite field $\mathbb{F}_{p^k}$ such that:*

1. *$p = 2^n - c$ for some integers n and c with $\log_2 |c| \leq n/2$ and*
2. *there exists an irreducible polynomial $f(X) = X^m - \omega$ in $\mathbb{F}_p[X]$.*

If $c \in \{\pm 1\}$, then the OEF is said to be of Type I (p is a Mersenne prime if $c = 1$); if $\omega = 2$, the OEF is said to be of Type II.

Examples of OEFs and discussions on how to find irreducible polynomials of the required form are given in [8]. The multiplication of A and B can be performed by an ordinary polynomial multiplication in $\mathbb{Z}[X]$, along with coefficient reductions in $\mathbb{F}_p$, followed by a reduction by the polynomial f. The number of reductions modulo p can be reduced by accumulation strategies on the coefficients of $C = AB$. As pointed out in [8], "the arithmetic resembles that commonly used in prime-field implementations, and multiplication cost in $\mathbb{F}_{p^k}$ is expected to be comparable to that in a prime field $\mathbb{F}_q$ where $q \simeq p^k$ and which admits fast reduction (e.g., the NIST-recommended primes)." This means that the multiplication in $\mathbb{F}_{p^k}$ can be performed in $O(k^2)$ multiplications in $\mathbb{F}_p$. When k is small (as in the case of finite fields used for ECC), fast multiplication algorithms, like Karatsuba-Ofman methods, are not likely to give faster implementations.

In the next section, we propose a modified Montgomery algorithm in $\mathbb{F}_{p^k}$ which, under certain conditions, only requires $O(k)$ multiplications in $\mathbb{F}_p$.

4 Modified Montgomery Multiplication in Lagrange Representation

In this section, we first modify Algorithm 1 by allowing the polynomial Ψ to be any polynomial of degree k satisfying $\gcd(\Psi, N) = 1$ and by replacing the division by Ψ in Step 2 by a multiplication by Ψ^{-1} modulo another given polynomial Ψ'. Note that this operation is possible only if $\gcd(\Psi, \Psi') = 1$. Then, we analyze a special case, where Ψ, Ψ' are the products of first-degree polynomials. Algorithm 2 computes $AB\Psi^{-1} \bmod N$ for any relatively

prime polynomials Ψ and Ψ' of degree k satisfying $\gcd(\Psi, N) = 1$ and $\gcd(\Psi, \Psi') = 1$.

Algorithm 2 Modified Montgomery Multiplication over $\mathbb{F}_{p^k}$
Input: $A, B \in \mathbb{F}_p[X]$, with $\deg A, \deg B \leq k-1$; a monic irreducible polynomial $N \in \mathbb{F}_p[X]$, with $\deg N = k$; Ψ, Ψ', with $\deg \Psi = \deg \Psi' = k$, and $\gcd(\Psi, \Psi') = \gcd(\Psi, N) = 1$
Output: $AB\Psi^{-1} \bmod N$
1: $T \leftarrow A \times B \bmod (\Psi \times \Psi')$
2: $Q \leftarrow T \times (-N^{-1}) \bmod \Psi$
3: $R \leftarrow (T + Q \times N) \times \Psi^{-1} \bmod \Psi'$

Remarks. In Step 1, the notation $A \times B \bmod (\Psi \times \Psi')$ simply means that we compute both $AB \bmod \Psi$ and $AB \bmod \Psi'$. Also, assume that N_Ψ^{-1} denotes the polynomial $N^{-1} \bmod \Psi$ of degree $\leq k-1$. We remark that Q computed in Step 2 is a polynomial of degree $\leq k-1$, whereas the polynomial $-ABN_\Psi^{-1}$ is of degree $\leq 3k-3$. Since the computations performed in Step 3 have to be carried out modulo Ψ', we have to compute $Q \bmod \Psi'$ from the knowledge of Q only. We note that it is impossible to compute $-ABN_\Psi^{-1} \bmod \Psi'$ exactly, but only the polynomial $(-ABN_\Psi^{-1} \bmod \Psi) \bmod \Psi'$, denoted $\tilde{Q}$ here. Since Q and $\tilde{Q}$ differ by a multiple of Ψ, both the values $(T + QN)$ and $(T + \tilde{Q}N)$ are multiples of Ψ and the multiplication by Ψ^{-1} modulo Ψ' gives the correct result (see Lemma 1). With this in mind, we shall abusively use Q in the following. Furthermore, if the result R computed in Step 3 has to be reused for another modular multiplication, as in the performance of an exponentiation, we must also compute $R \bmod \Psi$ from $R \bmod \Psi'$. We address the problem of converting polynomials between different Lagrange representations in Sections 4.1 and 4.2.

Lemma 1. *Algorithm 2 is correct; it returns* $AB\Psi^{-1} \bmod N$.

Proof. In Steps 1 and 2, we compute Q such that $(AB + QN)$ is a multiple of Ψ. Indeed, we have $(AB + QN) \equiv (AB - ABN^{-1}N) \equiv 0 \pmod{\Psi}$. This implies that there exists a polynomial f such that $(AB + QN) = f\Psi$, with $\deg f \leq k-1$. Now, in Step 3, we compute R modulo Ψ'. We have

$$(AB + QN)\Psi^{-1} \equiv (f\Psi)\Psi^{-1} \equiv f \pmod{\Psi'}.$$

Since $\deg \Psi' = k > \deg f$, we have

$$(AB + QN)\Psi^{-1} \bmod \Psi' = f.$$

Since $\deg N \geq k$, we have $R = f = AB\Psi^{-1} \bmod N$, which concludes the proof. □

Of course, Algorithm 2 is advantageous only if one can define polynomials Ψ, Ψ' such that the arithmetic operations modulo Ψ and Ψ' are easy to implement. The proposed solution takes advantage of the Lagrange representation (see Section 2).

Let $E = \{e_1, \ldots, e_k\}$ and $E' = \{e'_1, \ldots, e'_k\}$ be such that $E \cap E' = \emptyset$ and $e_i, e'_i \in \mathbb{F}_p$, for $1 \leq i \leq k$ (in other words, the e_is and e'_is are all distinct). We define $\Psi = \prod_{i=1}^{k}(X - e_i)$ and $\Psi' = \prod_{i=1}^{k}(X - e'_i)$, two polynomials of degree k such that $\gcd(\Psi, \Psi') = \gcd(\Psi, N) = 1$. We assume that the inputs A, B are given (or converted) into both LR_Ψ and $\mathrm{LR}_{\Psi'}$. We further suppose that the precomputed values are also known in Lagrange representation (modulo Ψ and/or Ψ'), more precisely, we need $\mathrm{LR}_\Psi(-N^{-1}) = (\tilde{n}_1, \ldots, \tilde{n}_k)$, $\mathrm{LR}_{\Psi'}(N) = (n'_1, \ldots, n'_k)$, and $\mathrm{LR}_{\Psi'}(\Psi^{-1}) = (\zeta_1, \ldots, \zeta_k)$.

As mentioned earlier, the arithmetic modulo Ψ (respectively, Ψ') is automatically and implicitly carried out in Lagrange representation by computing modulo the $(X - e_i)$ for $1 \leq i \leq k$ (respectively, modulo the $(X - e'_i)$ for $1 \leq i \leq k$). In the next two sections, we address the problem of converting a polynomial from LR_Ψ to $\mathrm{LR}_{\Psi'}$ (the reverse conversion is identical). We consider both Lagrange and Newton's interpolation formulae and we propose some implementation strategies that can be used to speed up the implementation in both cases.

4.1 Lagrange Interpolation

Assume that E, E', Ψ, and Ψ' are defined as above. If $\mathrm{LR}_\Psi(U) = (u_1, \ldots, u_k)$, then $\mathrm{LR}_{\Psi'}(U) = (u'_1, \ldots, u'_k)$ is given by the Lagrange interpolation theorem (or the CRT) by computing

$$u'_i = \sum_{j=1}^{k} u_j \left(\prod_{t=1, t\neq j}^{k} \frac{e'_i - e_t}{e_j - e_t} \right), \text{ for } 1 \leq i \leq k. \tag{10}$$

The computations can be easily expressed as a matrix-vector product. By defining the $k \times k$ constant matrix Ω with elements

$$\omega_{i,j} = \prod_{t=1, t\neq j}^{k} \frac{e'_i - e_t}{e_j - e_t}, \quad \text{for } 1 \leq i, j \leq k, \tag{11}$$

we have $\mathrm{LR}_{\Psi'}(U) = \Omega \times \mathrm{LR}_\Psi(U)$ or, equivalently,

$$\begin{pmatrix} u'_1 \\ u'_2 \\ \vdots \\ u'_k \end{pmatrix} = \begin{pmatrix} \omega_{1,1} & \omega_{1,2} & \ldots & \omega_{1,k} \\ \omega_{2,1} & \omega_{2,2} & \ldots & \omega_{2,k} \\ \vdots & & \ddots & \\ \omega_{k,1} & \omega_{k,2} & \ldots & \omega_{k,k} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_k \end{pmatrix}. \tag{12}$$

Similarly, for the reverse conversion from $\mathrm{LR}_{\Psi'}$ to LR_Ψ, we define the $k \times k$ constant matrix Ω' with elements

$$\omega'_{i,j} = \prod_{t=1, t\neq j}^{k} \frac{e_i - e'_t}{e'_j - e'_t}, \quad \text{for } 1 \leq i, j \leq k, \tag{13}$$

and we compute $\mathrm{LR}_\Psi(U) = \Omega' \times \mathrm{LR}_{\Psi'}(U)$.

The complexity of Lagrange interpolation is equal to

$$k^2\, CM + k(k-1)\, A. \tag{14}$$

Hence, in this case, the total cost of Algorithm 2 is

$$2k\, M + (2k^2 + 3k)\, CM + (2k^2 - k)\, A. \tag{15}$$

If one uses a general multiplier for the constant multiplications, then we must assume that $CM = M$ and a sequential implementation of our algorithm does not compare favorably either against the Montgomery approach (see Algorithm 1) or an OEF implementation. However, because the number of real multiplications is reduced from k^2 to $2k$, hardware implementations can take advantage of Lefèvre's multiplication by integer constants

[16] and of Boullis and Tisserand's approach for hardware multiplication by constant matrices [17]. These methods, based on number recoding and dedicated common subexpression factorization algorithms, have been implemented on FPGA for several applications. Based on their results, it is not unreasonable to expect savings of 20-to-40 percent in both area and time for the present application as well, but, of course, a precise analysis has to be done to support this claim.[2]

The operation count given by (15) does not take into account the fact that some optimization strategies are applicable. A first optimization we want to point out, which does not come from the Lagrange representation, concerns the base field arithmetic—i.e., the arithmetic modulo p—and the fact that complete reduction modulo p is not always necessary. Actually, it is only needed at the very end of the computational task. For all intermediate computations, several approaches are possible: partial reduction to maintain the values congruent to p and less than $2p$ (especially easy when p is a Mersenne prime), accumulation into two registers (of machine word size) assuming p fits into one register, or three registers can be employed, exactly as mentioned in [8, Example 2.56, p. 66] for OEFs. In the following, we will only refer to reduction, or partial reduction modulo p, but, for practical implementations, the best options have to be considered.

As for the OEFs, the cost of Algorithm 2 can be significantly reduced by looking for suitable sets of parameters. With OEFs, c and ω are the only parameters that can be adjusted (see Section 3.2). Our algorithm gives us more freedom; it allows for $2k$ values (the elements of E and E') to be adjusted. However, this higher degree of freedom makes the optimization process more difficult. Next, we present various optimization strategies that can be applied to reduce the cost of Algorithm 2 in time and/or space.

By noting that the reduction polynomial N does not directly influence the complexity of our algorithm, it is possible to define the points of E' such that $N = \Psi' + 1$ is irreducible.[3] It is very easy to find such polynomials. Indeed, for given prime p and $k > 0$, the number of irreducible polynomials of the form $\prod_{i=1}^{k}(X - e_i) + 1$ is equal to $\binom{p}{k}$. When p^k is not too small, the probability for a monic uniformly random monic polynomial of degree k in $\mathbb{F}_p[X]$ being irreducible is close to $1/k$. If we assume that our specific polynomials satisfy this estimate, the number of irreducible polynomials of this form is close to $\binom{p}{k}/k$. Using an exhaustive search, we have been able to verify this estimate for small fields (for example, the exact number of irreducible polynomial of degree 3 in $\mathbb{F}_{101}[X]$ of this form is equal to $56,661$, whereas $\binom{101}{3}/3 = 55,550$). For larger extensions, however, we do not know whether it is still valid. In the context of ECC, the degree k of the extension is usually small (see Table 3) and we believe that the estimate is correct.

If the points of E' are chosen such that $N = \Psi' + 1$ is irreducible, we have $\mathrm{LR}_{\Psi'}(N) = (1, \ldots, 1)$ and Step 3 of Algorithm 2 can be rewritten $R \leftarrow (T + Q) \times \Psi^{-1} \bmod \Psi'$. It saves kCM and, more importantly, it makes it possible to evaluate $r_i = (t_i + q'_i)\zeta_i \bmod p$ with only one reduction (or partial reduction) instead of two, by allowing the partial result $(t_i + q'_i)\zeta_i$ to be stored into two registers or without any reduction if one considers three registers.

Using the same idea, it is possible to define E such that $\mathrm{LR}_{\Psi}(-N^{-1}) = (\tilde{n}_1, \ldots, \tilde{n}_k)$ is composed of small integers. Hence, the multiplications by $\tilde{n}_i$ can be replaced by a small number of shifts and additions and the reduction is greatly simplified as $t_i \times \tilde{n}_i$ fits into a single register plus a few bits. Using a greedy algorithm, we have been able to find such polynomials and the corresponding sets of points. For example, for $p = 8,191$ and $k = 13$, the polynomial

$$N = X(X-1)\ldots(X-10)(X-2{,}089)(X-8{,}189) + 1$$

is the first irreducible polynomial (given by our greedy algorithm) such that there exists $k = 13$ points which satisfy $|\tilde{n}_i| \leq 3$; these points are given by

$$E = \{1{,}259, 1{,}872, 1{,}989, 3{,}215, 3{,}667, 3{,}791, 3{,}798, 4{,}197, 4{,}408, 4{,}589, 4{,}615, 4{,}900, 6{,}461\}.$$

The first irreducible polynomial such that $|\tilde{n}_i| \leq 4$ for $1 \leq i \leq 13$ was obtained even faster:

$$N = X(X-1)\ldots(X-11)(X-1558) + 1,$$
$$E = \{140, 286, 950, 1{,}315, 1{,}928, 2{,}293, 2{,}936, 3{,}086, 3{,}619, 5{,}187, 5{,}828, 7{,}374, 7{,}417\}.$$

Another possible optimization is to look for $\tilde{n}_i$s which are small powers of two. For example, with

$$N = X(X-1)\ldots(X-10)(X-1879)(X-8189) + 1,$$
$$E = \{269, 1{,}036, 1{,}086, 1{,}205, 1{,}484, 2{,}093, 2{,}672, 3{,}151, 3{,}517, 3{,}839, 4{,}111, 6{,}944, 7{,}651\},$$

we have $|\tilde{n}_i| \in \{1, 2, 4\}$ for $1 \leq i \leq k$.

Although it seems to be a difficult task, the freedom in the selection of the points of E and E' can be further exploited by trying to optimize the interpolation matrices Ω and Ω'. For example, one can try to construct matrices with as many small values (possibly 1 or small powers of 2 in absolute value) as possible. The only partial results we have at the moment, based on an exhaustive search for small fields, seem difficult to generalize to larger extension fields. However, other matrix optimizations are still possible. In [18], we have detected symmetries between the elements of Ω and Ω' that can contribute to simplified, smaller architectures. The following lemma holds (see [18] for a proof):

Lemma 2. *Assume $e_i = 2i$ and $e'_i = 2i + 1$. Then, from (11) and (13), we have*

$$\omega_{i,j} = \prod_{t=1, t\neq j}^{k} \frac{2i + 1 - 2t}{2j - 2t}, \quad \text{and}$$

$$\omega'_{i,j} = \prod_{t=1, t\neq j}^{k} \frac{2i - (2t+1)}{2j + 1 - (2t+1)}.$$

2. This work is part of a currently underway joint project with A. Tisserand.

3. Note that N is necessarily monic in this case as $N = \prod_{i=1}^{k}(X - e'_i) + 1 = X^k + \cdots + n_0$.

Hence, for every $1 \le i, j \le k$, *we have*

$$\omega_{i,j} = \omega'_{k+1-i,k+1-j}. \tag{16}$$

Lemma 2 tells us that the operation $\mathrm{LR}_\Psi(R) = \Omega' \times \mathrm{LR}_{\Psi'}(R)$, which has to be performed after Step 3 of Algorithm 2, can be replaced by $\overline{\mathrm{LR}_\Psi(R)} = \Omega \times \overline{\mathrm{LR}_{\Psi'}(R)}$ (with the matrix Ω instead of Ω'), where $\overline{U}$ denotes the vector composed of the elements of U in reverse order: $\overline{U} = (u_k, \ldots, u_1)$. In other words, we compute:

$$\begin{pmatrix} r'_k \\ r'_{k-1} \\ \vdots \\ r'_1 \end{pmatrix} = \begin{pmatrix} \omega_{1,1} & \omega_{1,2} & \ldots & \omega_{1,k} \\ \omega_{2,1} & \omega_{2,2} & \ldots & \omega_{2,k} \\ \vdots & & \ddots & \\ \omega_{k,1} & \omega_{k,2} & \ldots & \omega_{k,k} \end{pmatrix} \begin{pmatrix} r_k \\ r_{k-1} \\ \vdots \\ r_1 \end{pmatrix}.$$

Let us consider a small example. We define $p = 89$, $k = 5$, and we consider the sets $E = \{2, 4, 6, 8, 10\}$ and $E' = \{1, 3, 5, 7, 9\}$. We compute the constant interpolation matrix Ω that we are going to use for the two interpolation steps:

$$\Omega = \begin{pmatrix} 56 & 44 & 85 & 57 & 26 \\ 26 & 15 & 37 & 3 & 9 \\ 9 & 70 & 16 & 36 & 48 \\ 48 & 36 & 16 & 70 & 9 \\ 9 & 3 & 37 & 15 & 26 \end{pmatrix}.$$

Let $N = X^5 + 2X + 1$ be the irreducible polynomial defining the field $\mathbb{F}_{89^5}$. We need the following predefined constant vectors:

$$\begin{aligned} \mathrm{LR}_{\Psi'}(N) &= (4, 72, 21, 1, 61) \\ \mathrm{LR}_{\Psi}(N^{-1}) &= (77, 61, 60, 27, 83) \\ \mathrm{LR}_{\Psi'}(\Psi^{-1}) &= (55, 39, 87, 2, 50). \end{aligned}$$

Now, assume the inputs $A = 17X^4 + 6X + 35$ and $B = 59X^2 + 42X + 11$ are given in LR representation. We have

$$\begin{aligned} &\mathrm{LR}_\Psi(A) = (52, 50, 31, 28, 16), \mathrm{LR}_{\Psi'}(A) = (58, 6, 10, 43, 20), \\ &\mathrm{LR}_\Psi(B) = (64, 55, 73, 29, 12), \mathrm{LR}_{\Psi'}(B) = (23, 45, 5, 81, 6). \end{aligned}$$

In Step 1, we compute $T = A \times B \bmod \Psi \times \Psi'$:

$$\mathrm{LR}_\Psi(T) = (35, 80, 38, 11, 14), \mathrm{LR}_{\Psi'}(T) = (88, 3, 50, 12, 31).$$

Then, in Step 2, we compute

$$\mathrm{LR}_\Psi(Q) = (64, 15, 34, 59, 84),$$

which we interpolate modulo Ψ' by computing $\mathrm{LR}_{\Psi'}(Q) = \Omega \times \mathrm{LR}_\Psi(Q)$ to get

$$\mathrm{LR}_{\Psi'}(Q) = (43, 75, 49, 53, 53).$$

Next, we evaluate $R = (T + Q \times N) \times \Psi^{-1} \bmod \Psi'$:

$$\mathrm{LR}_{\Psi'}(R) = (60, 54, 67, 41, 63)$$

and we convert it back modulo Ψ using the same matrix Ω by computing $\overline{\mathrm{LR}_\Psi(R)} = \Omega \times \overline{\mathrm{LR}_{\Psi'}(R)}$:

$$\mathrm{LR}_\Psi(R) = (21, 13, 77, 5, 1).$$

One can easily check that the result is equal to $AB\Psi^{-1} \bmod N = 2X^4 + 15X^3 + 74X^2 + 49X + 9$ in the Lagrange representation.

4.2 Newton Interpolation

Assume that E, E', Ψ, and Ψ' are defined as above and $LR_\Psi(Q) = (q_1, \ldots, q_k)$. In order to compute $LR_{\Psi'}(Q) = (q'_1, \ldots, q'_k)$ using Newton's interpolation, we can precompute $k - 1$ constants

$$C_j = ((e_j - e_1)(e_j - e_2) \ldots (e_j - e_{j-1}))^{-1} \bmod p,$$

for $2 \le j \le k$, and we can evaluate $(\hat{q}_1, \ldots, \hat{q}_k)$ by setting

$$\begin{cases} \hat{q}_1 &= q_1 \bmod p, \\ \hat{q}_2 &= (q_2 - \hat{q}_1)\, C_2 \bmod p, \\ \hat{q}_3 &= (q_3 - (\hat{q}_1 + (e_3 - e_1)\hat{q}_2))\, C_3 \bmod p, \\ &\vdots \\ \hat{q}_k &= (q_k - (\hat{q}_1 + (e_k - e_1)(\hat{q}_2 + \cdots \\ &\qquad + (e_k - e_{k-2})\hat{q}_{k-1}) \ldots))\, C_k \bmod p. \end{cases} \tag{17}$$

As we shall see further, computing the $\hat{q}_i$s under this form allows for very interesting optimizations and can lead to very efficient implementation, with only a linear number of field multiplications. For parallel implementation, however, it is also possible to compute

$$\begin{aligned} \hat{q}_i = \big(\ldots ((q_i - \hat{q}_1)(e_i - e_1)^{-1} - \hat{q}_2)(e_i - e_2)^{-1} - \cdots \\ - \hat{q}_{i-1} \big)(e_i - e_{i-1})^{-1} \bmod p, \end{aligned} \tag{18}$$

for $2 \le i \le k$. For more details, see the discussion about the *mixed-radix representation* in [19, pp. 290-293] and exercise 5.11 in [13, page 125].

Once the $\hat{q}_i$s have been computed using (17) or (18), the polynomial

$$Q = \hat{q}_1 + \hat{q}_2\psi_1 + \hat{q}_3\psi_1\psi_2 + \cdots + \hat{q}_k\psi_1 \ldots \psi_{k-1} \tag{19}$$

satisfies the conditions

$$\deg Q \le k - 1, \quad Q(e_i) \equiv q_i \pmod p \quad \text{for } 1 \le i \le k. \tag{20}$$

We then evaluate $\mathrm{LR}_{\Psi'}(Q) = (q'_1, \ldots, q'_k)$ using Horner's rule. For $1 \le i \le k$, we compute $q'_i = Q \bmod (X - e'_i) = Q(e'_i)$. From (19), we have

$$\begin{aligned} q'_i = \big((\ldots (\hat{q}_k(e'_i - e_{k-1}) + \hat{q}_{k-1})(e'_i - e_{k-2}) + \cdots \\ + \hat{q}_2)(e'_i - e_1) + \hat{q}_1 \big) \bmod p. \end{aligned} \tag{21}$$

If the C_j are precomputed and if we do not take into account the cost of computing the values $(e_i - e_j)$, the complexity of (17) is equal to $k(k-1)/2\, CM + k(k-1)/2\, A$. Under the same assumptions, the computations in (21), performed for $1 \le i \le k$, require $k(k-1)\, CM + k(k-1)\, A$. The total cost of Newton interpolation is thus $3k(k-1)/2\, CM + 3k(k-1)/2\, A$, which, at first, seems very inefficient. However, as for Lagrange interpolation, this general complexity estimate can be significantly reduced by carefully choosing the points of interpolation.

Let us consider the first $2k$ integers: We define $E = \{0, \ldots, k-1\}$ and $E' = \{k, \ldots, 2k-1\}$. In this case, (17) rewrites

$$\begin{cases} \hat{q}_1 &= q_1 \bmod p, \\ \hat{q}_2 &= (q_2 - \hat{q}_1)C_2 \bmod p, \\ \hat{q}_3 &= (q_3 - (\hat{q}_1 + 2\hat{q}_2))C_3 \bmod p, \\ \vdots & \\ \hat{q}_k &= (q_k - (\hat{q}_1 + (k-1)(\hat{q}_2 + (k-2)(\hat{q}_3 + \cdots \\ & \quad +2\hat{q}_{k-1})\ldots)))\, C_k \bmod p. \end{cases} \tag{22}$$

By taking a closer look at (22), we notice that it requires $k-3$ multiplications by 2, $k-2$ multiplications by 3, ..., two multiplications by $k-2$, and one multiplication by $k-1$. Since the largest constant is equal to $k-1$ and k is usually small (see Table 3), all these operations can be readily computed with only a few number of shifts and additions. Thus, only $k-1$ constant multiplications by the C_is are actually needed.

The same applies for (21): The $k(k-1)$ constant multiplications by numbers of the form $(e'_i - e_j)$ can be performed with a small number of additions and shifts. More precisely, we have to compute

$$\begin{cases} q'_1 &= ((\ldots(\hat{q}_k \times 2 + \hat{q}_{k-1}) \\ & \quad \times 3 + \cdots + \hat{q}_2) \times k + \hat{q}_1) \bmod p, \\ q'_2 &= ((\ldots(\hat{q}_k \times 3 + \hat{q}_{k-1}) \\ & \quad \times 4 + \cdots + \hat{q}_2) \times (k+1) + \hat{q}_1) \bmod p, \\ \vdots & \\ q'_k &= ((\ldots(\hat{q}_k \times (k+1) + \hat{q}_{k-1}) \\ & \quad \times (k+2) + \cdots + \hat{q}_2) \times (2k-1) + \hat{q}_1) \bmod p, \end{cases} \tag{23}$$

which requires one multiplication by 2, two multiplications by 3, ..., $k-1$ multiplications by k, $k-1$ multiplications by $k+1$, ..., two multiplications by $2k-2$, and one multiplication by $2k-1$. As before, since the largest constant is equal to $2k-1$ and k is small, these operations can be evaluated with only a few shifts and additions. For $2 \le k \le 23$ (see Table 3), the numbers of additions required in the multiplications by the constants $c = 1, 2, \ldots, 2k-1$ are given in Table 1. We remark that 43 is the first number in the range which requires three additions. We also note that the nonadjacent form (NAF) does not always give the optimal number of addition; for example, the multiplication by $45 = (10\bar{1}0\bar{1}01)_2$ can be done with three additions if one considers the NAF or with only two if one considers its factorization $45 = 9 \times 5$.

Moreover, if we assume that p fits in a single machine word and the $\hat{q}_i$s are also reduced to fit into a single word, then the q'_is can be computed with a single reduction (or partial reduction) modulo p by allowing the partial result (before reduction) to be accumulated into two machine words. Let w denote the size (in bits) of one machine word. Since $q'_i = Q(e'_i)$, for $1 \le i \le k$, we remark that the size of the largest summand in (19) is equal to

$$w + \left\lfloor \log_2 \prod_{j=1}^{k-1} (e'_i - e_j) \right\rfloor + 1.$$

Moreover, since k terms need to be added to compute the q'_is and by noticing that the largest value before reduction in (23) is q'_k, we have

TABLE 1
Number of Additions ($\#A$) Required in the Multiplication by Some Small Constants c

c	$\#A$	c	$\#A$	c	$\#A$
1	0	16	0	31	1
2	0	17	1	32	0
3	1	18	1	33	1
4	0	19	2	34	1
5	1	20	1	35	2
6	1	21	2	36	1
7	1	22	2	37	2
8	0	23	2	38	2
9	1	24	1	39	2
10	1	25	2	40	1
11	2	26	2	41	2
12	1	27	2	42	2
13	2	28	1	43	3
14	1	29	2	44	2
15	1	30	1	45	2

$$\left|q'_i\right| < w + \sum_{t=k}^{2k-1} \lfloor \log_2 t \rfloor + k - 1,$$

where $|q|$ denotes the size of q (in bits). Hence, accumulation into two machine words is possible if the following condition is satisfied:

$$\sum_{t=k}^{2k-1} \lfloor \log_2 t \rfloor + k - 1 \le w. \tag{24}$$

As an example, if we consider 32-bit registers ($w = 32$), with $p = 2^{31} - 1$ and $k = 7$, then we have $\sum_{t=7}^{13} \lfloor \log_2 t \rfloor + 7 - 1 = 26 \le w = 32$ and (24) is satisfied.

The asymptotic complexity of the multiplication by a constant k is an open problem. Although Lefèvre conjectured it to be $O((\log k)^{0.85})$, if we consider the best known complexity of $O(\log k)$ additions, the cost of Newton interpolation in the context of Algorithm 2 becomes

$$k - 1\, CM + O(k^2 \log k)\, A, \tag{25}$$

and the total cost of Algorithm 2 is

$$2k\, M + (4k-1)\, CM + O(k^2 \log k)\, A. \tag{26}$$

Even if one uses a general multiplier for the multiplications by the large constants, our algorithm shows a better asymptotic complexity than the multiplication algorithms suggested for the OEFs [7], [8], with only a linear number of multiplications. Its complexity is

$$O(k)\, M + O(k^2 \log k)\, A. \tag{27}$$

5 Inversion

In this section, we present an algorithm for computing the inverse of an element A in $\mathbb{F}_{p^k}$ given in Lagrange representation. We use the same notations as before: N is a monic irreducible polynomial of degree k in $\mathbb{F}_p[X]$ and the elements of $\mathbb{F}_{p^k}$ are the polynomials in $\mathbb{F}_p[X]$ of degree less than or equal to $k-1$.

5.1 Polynomial GCD and Inverse Computation

Let us start with the more general case of polynomials defined over a field K. If A, B are two polynomials in $K[X]$ with $B \neq 0$, then there exists (unique) polynomials Q and R in $K[X]$ such that

$$A = QB + R \text{ and either } R = 0 \text{ or } \deg R < \deg B. \quad (28)$$

We define the polynomial GCD of A and B, not both zero, as a polynomial of greatest degree that divides both A and B. If the polynomial G satisfies this definition, then so does any polynomial of the form uG, where u is a unit[4] in $K[X]$. In other words, there is a set of greatest common divisors of A and B, each one being a unit multiple of the others. The ambiguity is removed by considering that "the" greatest common divisor of A and B is the (unique) monic polynomial of greatest degree which divides both A and B.

As for integers, the Bézout identity holds: For A, B not both equal to 0, there exist polynomials U and V such that

$$AU + BV = \gcd(A, B). \quad (29)$$

The polynomials U and V are called the Bézout coefficients of A and B. It is well known that the extended Euclidean algorithm can be used to compute the inverse of an element in K; from (29), we have $U \equiv A^{-1} \pmod{B}$ and $V \equiv B^{-1} \pmod{A}$. Many variants of the extended Euclidean algorithm for polynomials have been reported in the literature [19], [20], [21], [22]. A very thorough complexity analysis can be found in [13, pp. 46-53]. The presented algorithm is based on the classical Euclidean loop: While $B \neq 0$, $\gcd(A, B) = \gcd(B, R)$, where $R = A \bmod B$ is given by (28). The initial polynomials A, B and the partial quotients, remainders, and Bézout coefficients are kept monic to save some operations in the polynomial division. If $\deg A = n \geq \deg B = m$, then the algorithm requires at most $m + 2$ inversions and $\frac{13}{2}nm + O(n)$ additions and multiplications in K. Since we are only interested in one of the Bézout coefficients for the inversion and because one of the input polynomials is already monic in the case of finite field inversion, the complexity can be reduced to $\frac{9}{2}nm + O(n)$ additions and multiplications, plus at most $m + 1$ inversions.

5.2 Extended Euclidean Algorithm for Polynomials in LR

In this section, we propose an inversion algorithm, based on the extended Euclidean algorithm for polynomials defined over $\mathbb{F}_{p^k}$, where the input polynomials are given in the Lagrange representation. Given $A \in \mathbb{F}_p[X]$ with $\deg A \leq k - 1$ and $N \in \mathbb{F}_p[X]$, a monic irreducible polynomial of degree k, we compute $A^{-1} \bmod N$. More precisely, Algorithm 4 (below) receives $\mathrm{LR}_\Psi(A)$ and $\mathrm{LR}_\Psi(N)$ and returns $\mathrm{LR}_\Psi(A^{-1} \bmod N)$. We first notice that N, which is a polynomial of degree k, cannot be represented in Lagrange representation with only k values; its exact representation would require $k + 1$ values. However, by considering $\mathrm{LR}_\Psi(N)$, we have the exact (unique) representation of $N \bmod \Psi$, which is sufficient to compute $\mathrm{LR}_\Psi(A^{-1} \bmod N) = (A^{-1} \bmod N) \bmod \Psi$.

The main drawback of the Lagrange representation for performing a polynomial division is the ignorance of the degree and coefficients of the polynomials we are manipulating. To bypass this problem, we propose an algorithm which computes the degree and leading coefficient of a polynomial U given in the Lagrange representation.

Assume $U \in \mathbb{F}_p[X]$ with $\deg U < k$ is given in LR, i.e., $\mathrm{LR}_\Psi(U) = (u_1, \ldots, u_k)$. From (2) and (3), we remark that

$$\begin{aligned} U(X) &= \sum_{i=1}^{k} u_i \prod_{j=1, j\neq i}^{k} \frac{X - e_j}{e_i - e_j} \\ &= \sum_{i=1}^{k} \frac{u_i}{\prod_{j=1, j\neq i}^{k} (e_i - e_j)} \prod_{j=1, j\neq i}^{k} (X - e_j) \quad (30) \\ &= \sum_{i=1}^{k} \frac{u_i}{\prod_{j=1, j\neq i}^{k} (e_i - e_j)} X^{k-1} + \cdots. \end{aligned}$$

Thus, the coefficient of degree $k - 1$ of U is given by

$$\ell(U) = \sum_{i=1}^{k} u_i \left(\prod_{j=1, j\neq i}^{k} (e_i - e_j) \right)^{-1} \bmod p. \quad (31)$$

Thanks to the Lagrange interpolation theorem, we know that, if $\deg U < k$, it is uniquely defined by $(u_1, \ldots, u_k)$. Hence, if $\deg U < k - 1$, we clearly get $\ell(U) = 0$ in (31). A straightforward solution to finding the degree and leading coefficient of U in this case is to repeat the process for the degrees $k - 2$, $k - 3$, etc., until one finds a nonnull coefficient. If (31) tells us that $\deg U \neq k - 1$, we know that $k - 1$ values are sufficient to define U uniquely and we can consider any subset of E of size $k - 1$ to compute $\ell(U)$. In Algorithm 3, the sum in (31) is evaluated for $1 \leq i \leq t$, where t is initially set to $m + 1$ and m is the largest possible degree for U, and decremented by 1 each time the tested coefficient is equal to 0. At the end, the degree of U is equal to $t - 1$.

Algorithm 3 Leading term—LT(U,m)
Precomputed: $\zeta_{i,t} = (\prod_{j=1, j\neq i}^{t} (e_i - e_j))^{-1} \bmod p$, for $i \leq t$ and $t = 1, \ldots, k$
Input: A polynomial U of degree at most $m \leq k - 1$ given in Lagrange representation: $\mathrm{LR}_\Psi(U) = (u_1, \ldots, u_k)$
Output: (d, c), where $d = \deg U$ and $c = \ell(U)$ such that $U = cX^d + \cdots$

```
1:  if m = 0 then
2:      c ← u_1
3:  else
4:      t ← m + 1
5:      c ← 0
6:      while c = 0 do
7:          for i ← 1 to t do
8:              c ← c + u_i ζ_{i,t} mod p
9:          if c = 0 then
10:             t ← t − 1
11: return (t − 1, c)
```

4. In our case, the units will be the elements of $\mathbb{F}_p^*$.

TABLE 2
Iterations of Extended Euclid's Algorithm 4 in LR, with $\mathrm{LR}_\Psi(A) = (5, 10, 3)$ and $\mathrm{LR}_\Psi(N) = (5, 4, 4)$

U_1	U_3	V_1	V_3	$d(U_3)$	$\ell(U_3)$	t	q
$(1, 1, 1)$	$(5, 10, 3)$	$(0, 0, 0)$	$(5, 4, 4)$	2	11	-1	
$(0, 0, 0)$	$(5, 4, 4)$	$(1, 1, 1)$	$(5, 10, 3)$			1	14
$(3, 6, 9)$	$(3, 13, 14)$			2	4	0	5
$(15, 1, 4)$	$(12, 14, 16)$			1	2	-1	
$(1, 1, 1)$	$(5, 10, 3)$	$(15, 1, 4)$	$(12, 14, 16)$			1	14
$(12, 7, 3)$	$(7, 9, 11)$			1	2	0	1
$(14, 6, 16)$	$(12, 12, 12)$			0	12		
$(4, 9, 7)$	$(1, 1, 1)$						

We assume that the values

$$\zeta_{i,t} = \left(\prod_{j=1, j\neq i}^{t} (e_i - e_j) \right)^{-1} \bmod p$$

for $1 \leq i \leq t \leq k$ are precomputed. This requires the storage of $k(k+1)/2$ integers less than p. If $\deg U = m \leq k-1$, the cost of $LT(U, m)$ (in terms of the number of operations in $\mathbb{F}_p$) is

$$(m+1)\,CM + m\,A. \tag{32}$$

In Algorithm 4, all the variables U_i, V_i are represented in the Lagrange representation. The variables $d(U), \ell(U)$ denote the degree and leading coefficient of U, respectively. We use a polynomial version of Lehmer's Euclidean GCD algorithm [12], [19], [22], where one step of each polynomial division is performed, i.e., if $\deg U \geq \deg V$ and $t = \deg(U) - \deg(V)$, then we compute $q = \ell(U)/\ell(V)$ and $R = U - qX^tV$. The process is repeated until a zero remainder is encountered.

Algorithm 4 Inversion over $\mathbb{F}_{p^k}$ in LR
Precomputed: $X_t = \mathrm{LR}_\Psi(X^t)$, for $0 \leq t \leq k$
Input: $\mathrm{LR}_\Psi(A) = (a_1, \ldots, a_k)$ and $\mathrm{LR}_\Psi(N) = (n_1, \ldots, n_k)$ such that $\gcd(A, N) = 1$,
Output: $\mathrm{LR}_\Psi(A^{-1} \bmod N)$.
1: $(U_1, U_3) \leftarrow (\mathrm{LR}_\Psi(1), \mathrm{LR}_\Psi(A))$
2: $(V_1, V_3) \leftarrow (\mathrm{LR}_\Psi(0), \mathrm{LR}_\Psi(N))$
3: $(d(V_3), \ell(V_3)) \leftarrow (k, 1)$ {N is monic of degree k}
4: $(d(U_3), \ell(U_3)) \leftarrow LT(U_3, k-1)$ {$\deg U_3 \leq k-1$}
5: **while** $U_3 \neq 0$ **do**
6: $t \leftarrow d(U_3) - d(V_3)$
7: **if** $t < 0$ **then**
8: $(U_1, U_3) \leftrightarrow (V_1, V_3)$
9: $(d(U_3), \ell(U_3)) \leftrightarrow (d(V_3), \ell(V_3))$
10: $t \leftarrow -t$
11: $q \leftarrow \ell(U_3)\,\ell(V_3)^{-1} \bmod p$
12: $U_1 \leftarrow U_1 - q\,X_tV_1$
13: $U_3 \leftarrow U_3 - q\,X_tV_3$
14: $(d(U_3), \ell(U_3)) \leftarrow LT(U_3, d(U_3) - 1)$
15: **return** U_1

To illustrate our inversion algorithm in LR, we consider a small example, using the following parameters: $p = 17$, $k = 3$, $E = \{1, 2, 3\}$, and $N = X^3 + 3X^2 + 1$. We compute the inverse of $A = 11X^2 + 6X + 5$ modulo N in Lagrange representation. We have $\mathrm{LR}_\Psi(A) = (5, 10, 3)$ and $\mathrm{LR}_\Psi(N) = (5, 4, 4)$. Note that

$$\mathrm{LR}_\Psi(N) = \mathrm{LR}_\Psi(N \bmod \Psi) = \mathrm{LR}_\Psi(9X^2 + 6X + 7).$$

The initialization step gives

$$\mathrm{LR}_\Psi(U_1) = (1, 1, 1), \mathrm{LR}_\Psi(U_3) = (5, 10, 3), \mathrm{LR}_\Psi(V_1) = (0, 0, 0),$$

and $\mathrm{LR}_\Psi(V_3) = (5, 4, 4)$. We know that $d(V_3) = d(N) = 3$ and $\ell(V_3) = \ell(N) = 1$. The iterations of Algorithm 4 are summarized in Table 2. We remark that $\gcd(A, N) = 1$ (given in LR by U_3) and that the inverse of A modulo N, given by U_1, is $\mathrm{LR}_\Psi(A^{-1} \bmod N) = (4, 9, 7)$. It is easy to verify that $A^{-1} \bmod N$ is equal to $5X^2 + 7X + 9$, which, evaluated at $\{1, 2, 3\}$, gives the same result.

Let us now evaluate the complexity of Algorithm 4. Since $\deg R < \deg U$, the number of (partial) division steps is at most $\deg(N \bmod \Psi) + \deg A = 2k - 2$. If we omit the calls to $LT(U, m)$ for now, each iteration requires: one inversion plus one multiplication for the computation of q in Step 10, plus $3k$ multiplications and $2k$ additions for the computations of U_1 and U_3 in Steps 11 and 12 (we need k multiplications for qX_t and $2k$ for qX_tV_j for $j = 1, 3$). How many calls to $LT(U_3, m)$ do we have? Since $\deg U_3, \deg V_3 \leq k-1$ and both U_3 and V_3 have to be reduced (their values are swapped whenever $t < 0$) to polynomials of degree zero, there are exactly two calls to $LT(U_3, i)$, for $1 \leq i \leq k-1$ (we note that the calls to $LT(U, 0)$ are free). The total complexity is thus $2k-2$ inversions plus $(2k-2)M + (2k-2)(3kM + 2kA) + 2\sum_{i=1}^{k-1} LT(U_3, i)$

$$\begin{aligned} &= (6k^2 - 4k - 2)M + (4k^2 - 4k)A \\ &\qquad + 2\sum_{i=1}^{k-1}(i+1)CM + 2\sum_{i=1}^{k-2} i\,A \\ &= (6k^2 - 4k - 2)M + (5k^2 - 5k)A + (k^2 + k - 2)CM, \end{aligned} \tag{33}$$

which can be simplified to $2k-2$ inversions, plus $12k^2 - 8k - 4$ operations in $\mathbb{F}_p$.

A more careful analysis shows that some operations can be saved. Since the degree of U_3 is decreasing from $k-1$ to 0, it is not necessary to perform the computations in Step 12 ($U_3 - qX_tV_3$) for all k values representing $\mathrm{LR}_\Psi(U_3) = (u_1, \ldots, u_k)$. Note that qX_t and U_1 in Step 11 must always be computed entirely, i.e., for all k values. In the worst case, the degree of U_3 is decremented by one every two iterations. Thus, we can save $1\,M + 1\,A$ for the first two iterations, $2\,M + 2\,A$ for the next two, and so on, up to $(k-1)\,M + (k-1)\,A$ for the last two iterations. This represents a savings of $(k^2 - k)\,M + (k^2 - k)\,A$. Eventually, the total cost of Algorithm 4 is $2k-2$ inversions plus

$$(5k^2 - 3k - 2)\,M + (4k^2 - 4k)\,A + (k^2 + k - 2)\,CM \tag{34}$$

TABLE 3
Good Candidates for p and k Suitable for Elliptic Curve Cryptography and the Corresponding Key Lengths

p	form of p	k	l
59	2^6-2^2-1	29	170
67	2^6+3	29 ... 31	175 ... 188
73	2^6+2^3+1	29 ... 31	179 ... 191
127	2^7-1	23 ... 61	160 ... 426
257	2^8+1	23 ... 73	184 ... 584
503	2^9-2^3-1	19 ... 61	170 ... 547
521	2^9+2^3+1	19 ... 61	171 ... 550
8191	$2^{13}-1$	13 ... 43	168 ... 558
65537	$2^{16}+1$	11 ... 37	176 ... 592
131071	$2^{17}-1$	11 ... 31	186 ... 526
524287	$2^{19}-1$	11 ... 31	208 ... 588
2147483647	$2^{31}-1$	7 ... 19	216 ... 588
2305843009213693951	$2^{61}-1$	3 ... 7	182 ... 426

or, equivalently, $10k^2-6k-4$ additions and multiplications in $\mathbb{F}_p$.

Compared to the extended Euclidean algorithm presented in Section 5.1 whose complexity is k inversions and $\frac{9}{2}k^2+O(k)$ operations in $\mathbb{F}_p$, our inversion algorithm requires roughly twice as many operations. This is mainly due to the fact that, using the Lehmer approach, we are performing twice as many Euclidean steps. Unfortunately, the Lagrange representation does not allow us to perform a complete polynomial division at each iteration. For hardware implementations, the parallel nature of the Lagrange representation might compensate for this extra cost if several processing units are used. In the next section, we justify the interest of being able to perform an inversion in the Lagrange representation in the context of elliptic curve cryptography.

6 Discussions

For ECC, we usually prefer p and k to be prime. From a security point of view, it is not clear yet whether curves defined over such extension fields render the system less secure. Except for a family of well-defined weak curves, the best-known approaches to solving the ECDLP are generic algorithms, such as Pollard's Rho method [23]. Some attempts have recently been made to solve the ECDLP for curves defined over small extension fields. In [24], Gaudry proposed a solution which is asymptotically faster than Pollard's Rho when the degree of the extension is equal to zero mod 3 or 4. Explicitly, Gaudry's attack "can solve an elliptic curve discrete logarithm problem defined over $\mathbb{F}_{q^3}$ in time $O(q^{4/3})$, with a reasonably small constant, and an elliptic problem over $\mathbb{F}_{q^4}$ or a genus 2 problem over $\mathbb{F}_{q^2}$ in $O(q^{3/2})$ with a larger constant." In our case, we are only interested in k being a prime. With the light brought by these last results, it is thus preferable to avoid the case $k=3$. Table 3 gives some good candidates for p and k and the corresponding key length $l=\lfloor \log_2(p^k) \rfloor$ in bits. For each prime p, we give the form of p and the smallest and largest primes k satisfying the condition $p>2k$ required for our multiplication. For large p, we only give the extensions which lead to key sizes smaller than 600 bits. The number of possible combinations for the primes p and k is huge. It is, of course, impossible to list all of them. Since the form of the reduction polynomial does not directly influence the complexity of our multiplication algorithm, we can select a prime p even if there is no "good" reduction polynomial of the desired degree. For example, it is possible to choose $F_3=257$ and $F_4=65537$, the fourth and fifth Fermat primes, as well as all the Mersenne primes starting from $127=2^7-1$. Note that the only Type II OEF for Mersenne primes up to $2^{89}-1$ is obtained for $p=2^{13}-1$ (see [8]).

Our inversion algorithm requires twice as many operations as the classical Euclidean GCD algorithm for polynomials over a finite field. For ECC, this is not a very serious issue since projective coordinates can be used to avoid all the inversions except one at the end of the point multiplication, i.e., the computation of the point $kP=P+\ldots+P$ (k times), where k is a large integer and P is a point on the curve. (See [8] or [25] for more details about elliptic curve arithmetic.) Furthermore, we remark that all the computations of an ECC protocol (ECDH for example) could be performed in the Lagrange representation. Once Alice and Bob have agreed on a set of parameters (finite field, elliptic curve, and base point P on the curve) and have converted the coordinates of P in the Lagrange representation, then all the computations and exchanges of information could be done in LR. For ECDH, we further notice that there would be no need to perform an interpolation at the end since the results they both get in LR are identical. If k_1 and k_2 are their secret random scalars, they both end up with the point k_1k_2P whose coordinates, given in LR, can be considered as several[5] sets of k integers (elements of $\mathbb{F}_p$) as in the coefficient-based representation. In this context, we believe that it is advantageous to be able to perform an inversion in the Lagrange representation using, for example, the extended GCD algorithm presented in Section 5.

7 Conclusions

In this paper, we have presented a complete set of arithmetic operations for finite fields of the form $\mathbb{F}_{p^k}$. The elements of the field are modeled as polynomials of degree less than k by their values as sufficiently many points (instead of their coefficients). This representation scheme is

5. The number of coordinates depends on the type of projective coordinate.

called the Lagrange representation. Our multiplication, which is a modified Montgomery algorithm, works for $p > 2k$ and can be implemented with only a linear number of multiplication in $\mathbb{F}_p$. The inversion is performed using a variant of the extended Euclidean algorithm, where the degree and leading term of the coefficients of the polynomials manipulated in LR have to be computed at each iteration. The Lagrange representation is particularly attractive for ECC algorithms (with projective coordinates to reduce the number of inversions) since all the computations and exchange of information can possibly be performed within this system.

ACKNOWLEDGMENTS

This work was conducted during Dr. Imbert's leave of absence at the University of Calgary (Canada) with the ATIPS labs (Department of Electrical and Computer Engineering) and CISaC (Department of Mathematics and Statistics). It was funded by Canadian NSERC Strategic Grant #73-2048, *Novel Implementation of Cryptographic Algorithms on Custom Hardware Platforms* and by grant *ACI Sécurité Informatique, Opérateurs Cryptographiques et Arithmétique Matérielle* from the French Ministry of Education and Research.

REFERENCES

[1] N. Koblitz, "Elliptic Curve Cryptosystems," *Math. Computation,* vol. 48, no. 177, pp. 203-209, Jan. 1987.
[2] V.S. Miller, "Uses of Elliptic Curves in Cryptography," *Advances in Cryptology, Proc. CRYPTO '85,* H.C. Williams, ed., pp. 417-428, 1986.
[3] A. Menezes, P.C. Van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1997.
[4] IEEE, IEEE 1363-2000 Standard Specifications for Public-Key Cryptography, 2000.
[5] Nat'l Inst. of Standards and Technology, FIPS PUB 186-2: Digital Signature Standard (DSS), Jan. 2000.
[6] D. Bailey and C. Paar, "Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms," *Advances in Cryptography, Proc. CRYPTO '98,* H. Krawczyk, ed., pp. 472-485, 1998.
[7] D. Bailey and C. Paar, "Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography," *J. Cryptology,* vol. 14, no. 3, pp. 153-176, 2001.
[8] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography.* Springer-Verlag, 2004.
[9] N.P. Smart and E.J. Westwood, "Point Muliplication on Ordinary Elliptic Curves over Fields of Characteristic Three," *Applicable Algebra in Eng., Comm., and Computing,* vol. 13, no. 6, pp. 485-497, Apr. 2003.
[10] J.A. Solinas, "Improved Algorithms for Arithmetic on Anomalous Binary Curves," Research Report CORR-99-46, Center for Applied Cryptographic Research, Univ. of Waterloo, Canada, updated version of the paper appearing in the Proc. CRYPTO '97, 1999.
[11] N. Koblitz, "CM Curves with Good Cryptographic Properties," *Advances in Cryptography, Proc. CRYPTO '91,* pp. 279-287, 1992.
[12] D.H. Lehmer, "Euclid's Algorithm for Large Numbers," *Am. Math. Monthly,* vol. 45, no. 4, pp. 227-233, 1938.
[13] J. Von Zur Gathen and J. Gerhard, *Modern Computer Algebra.* Cambridge Univ. Press, 1999.
[14] P.L. Montgomery, "Modular Multiplication without Trial Division," *Math. Computation,* vol. 44, no. 170, pp. 519-521, Apr. 1985.
[15] Ç.K. Koç and T. Acar, "Montgomery Multiplication in $GF(2^k)$," *Designs, Codes, and Cryptography,* vol. 14, no. 1, pp. 57-69, Apr. 1998.
[16] V. Lefèvre, "Multiplication by an Integer Constant," Research Report 4192, INRIA, May 2001.
[17] N. Boullis and A. Tisserand, "Some Optimizations of Hardware Multiplication by Constant Matrices," *IEEE Trans. Computers,* vol. 54, no. 10, pp. 1271-1282, Oct. 2005.
[18] J.-C. Bajard, L. Imbert, C. Nègre, and T. Plantard, "Multiplication in $GF(p^k)$ for Elliptic Curve Cryptography," *Proc. 16th IEEE Symp. Computer Arithmetic,* pp. 181-187, 2003.
[19] D.E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms,* third ed. Reading, Mass.: Addison-Wesley, 1997.
[20] R. Crandall and C. Pomerance, *Prime Numbers. A Computational Perspective.* Springer-Verlag, 2001.
[21] J. Sorenson, "Two Fast GCD Algorithms," *J. Algorithms,* vol. 16, no. 1, pp. 110-144, 1994.
[22] J. Sorenson, "An Analysis of Lehmer's Euclidean Algorithm," *Proc. 1995 Int'l Symp. Symbolic and Algebraic Computation (ISSAC '95),* pp. 254-258, 1995.
[23] J.M. Pollard, "Monte Carlo Methods for Index Computation mod p," *Math. Computation,* vol. 32, no. 143, pp. 918-924, July 1978.
[24] P. Gaudry, "Index Calculus for Abelian Varieties and the Elliptic Curve Discrete Logarithm Problem," preprint, Oct. 2004.
[25] R.M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* CRC Press, 2005.

Jean-Claude Bajard received the PhD degree in computer science in 1993 from the École Normale Supérieure de Lyon (ENS), France. He taught mathematics in high school from 1979 to 1990 and served as a research and teaching assistant at the ENS in 1993. From 1994 to 1999, he was an assistant professor at the Université de Provence, Marseille, France. He joined the Université Montpellier 2, Montpellier, France, in 1999, where he is currently a professor. He is a member of the ARITH Group in the Department of Computer Science of the Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM). His research interests include computer arithmetic and cryptography. He is a member of the IEEE and the IEEE Computer Society.

Laurent Imbert received the MS and PhD degrees in computer science from the Université de Provence, Marseille, France in 1997 and 2000, respectively. He was a postdoctoral fellow at the University of Calgary, Alberta, Canada, from October 2000 to August 2001. Since October 2001, he has been with the ARITH Group of the Department of Computer Science at the Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), Montpellier, France, where he holds a senior researcher position for the French National Centre for Scientific Research (CNRS). His research interests includes efficient implementation of cryptographic systems, algorithmic number theory, and computer arithmetic. He is a member of the IEEE and the International Association for Cryptographic Research (IACR).

Christophe Nègre received the MS degree in mathematics and the PhD degree in computer science from the Université Montpellier 2, Montpellier, France, in 2001 and 2004, respectively. From September 2004 to October 2005, he was a research and teaching assistant. Since November 2005, he has been an assistant professor with the DALI Group at the Université de Perpignan, France. His research interests are in cryptography and number theory.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.