Applied Mathematics & Information Sciences

Volume 10
Issue 5 *Sep. 2016*

Article 2

9-1-2016

Accelerating Finite Field Inversion in GF(3m) for Elliptic Curve Cryptography

Walid Mahmoud
Electrical & Computer Engineering, University of Windsor, Windsor, Ontario, Canada,
wmahmoud@gmail.com

Huapeng Wu
Electrical & Computer Engineering, University of Windsor, Windsor, Ontario, Canada,
wmahmoud@gmail.com

Recommended Citation

Mahmoud, Walid and Wu, Huapeng (2016) "Accelerating Finite Field Inversion in GF(3m) for Elliptic Curve Cryptography," *Applied Mathematics & Information Sciences*: Vol. 10: Iss. 5, Article 2.
DOI: http://dx.doi.org/10.18576/amis/100502
Available at: https://digitalcommons.aaru.edu.jo/amis/vol10/iss5/2

Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/100502

Accelerating Finite Field Inversion in $GF(3^m)$ for Elliptic Curve Cryptography

Walid Mahmoud[*] *and Huapeng Wu*

Electrical & Computer Engineering, University of Windsor, Windsor, Ontario, Canada

Received: 3 Apr. 2016, Revised: 2 Jun. 2016, Accepted: 3 Jun. 2016
Published online: 1 Sep. 2016

Abstract: Ternary extension fields $GF(3^m)$ have been used in cryptographic applications based on bilinear-mappings in elliptic curve cryptography. In this paper, we focus on accelerating inversion in $GF(3^m)$ which is an indispensable operation in such applications. We propose a fast execution-time inversion algorithm which decomposes $(m-1)$ of $GF(3^m)$ into several factors and a remainder and restricts the remainder to belong to the shortest addition chain of a suitable factor. Thus, unlike other algorithms that not decompose $(m-1)$ and search for large near-optimal addition chains for $(m-1)$ to compute the inverse, our algorithm relies on much smaller and known chains for the suitable factors. In decomposing $(m-1)$ with the use of small and known chains for the suitable factors, as far as we know, our proposal is the fastest polynomial-time inversion algorithm in comparison with its counterparts.

Keywords: Elliptic curve cryptography, Fermat's theorem, field inversion, normal basis representation, optimal decomposition, short addition chain, ternary extension field

1 Introduction

In ternary extension fields $GF(3^m)$ where the characteristic $p = 3$ and the extension degree $m > 1$, the field's elements are represented by vectors of size m with coefficients belonging to the underlying subfield $GF(3)$. As declared by Galbraith [1], such fields were widely used in cryptographic applications based on bilinear-mappings, like Tate or Weil pairings, and exhibit more bandwidth efficiency relative to other extension fields.

In $GF(3^m)$, the basic arithmetic operations are addition, multiplication, division, inversion, etc. Of particular interest is the multiplicative inverse (inversion) operation which has the highest execution-time in comparison with other field operations [2]. Inversion is required in scalar multiplication algorithm (SMA) that exists in most cryptographic applications those based on elliptic curve cryptography (ECC). Thus, accelerating the execution-time of inversion accelerates the execution of such applications [3].

In general, irrespective of the used representation basis for the elements in finite fields, inversion can be computed either using Euclidean division algorithm or one of its variants [4,5,6], or using Fermat's little theorem [7,8,9]. In particular, inversion algorithms based on Fermat's approach using normal basis representation (NB) for the field elements require computing a number of field multiplication operations, the second costly operation in finite fields [10]. Thus, such a number determines the inversion cost (IC) performance of respective inversion algorithm. Therefore, the final goal is to reduce such a number to achieve lower IC and to get inversion algorithms associated with fast execution-time.

In the literature, the common way to compute inversion using Fermat's approach in $GF(p^m)$, for prime characteristic $p > 2$ and $m > 1$, is based on the method proposed earlier in [11]. Many authors including [12,13,14] rely on such a method. Such a method simplifies inversion in extension field $GF(p^m)$ to inversion in the subfield $GF(p)$, in addition to computing a logarithmic number of multiplications in $GF(p^m)$.

For example, assume a nonzero element $\alpha \in GF(3^m)$, its multiplicative inverse using the method in [11] is computed as $\alpha^{-1} = (\alpha^r)^{-1} \times (\alpha)^{r-1} : r = \frac{p^m-1}{p-1}$ and (α^r) is a subfield element. Thus, finding the inverse requires $[\ell(m-1) + \omega(m-1)]$ multiplications in $GF(3^m)$, in addition to a subfield inversion, where $\ell(z)$ and $\omega(z)$ are the binary length and Hamming weight (number of 1s) of binary z, respectively.

[*] Corresponding author e-mail: wmahmoud@gmail.com

Alternatively, some authors rely on other inversion method based on short addition chains (SACs). Such a method attempts to find the SAC for $(m-1)$ of $GF(p^m)$ as the initial step to compute inversion. Unfortunately enough, the search for SACs for $(m-1)$ values lengthen the execution time of the respective inversion algorithm when dealing with large m values suitable for cryptographic purposes. Thus, authors rely on heuristic strategies to search for near-optimal addition chains (NACs) to search for $(m-1)$ values where such a search is a NP-hard problem as claimed in [15].

Given that finding a NAC is merely a hard optimization problem with large search space, some authors [16,17] rely on deterministic heuristics, whereas others [15,18,19] rely on probabilistic heuristics to find the NACs for large $(m-1)$ values. In the latter heuristics, the initial conditions are not kept fixed and repeated runs may produce more optimized results relative to the deterministic heuristics. After finding the appropriate NAC, the inverse is given by

$$\xi_{c_{i_1}+c_{i_2}}(\alpha) = [\xi_{c_{i_1}}(\alpha)]^{3^{c_{i_2}}} \times \xi_{c_{i_2}}(\alpha),$$

where the i^{th} NAC element $c_i = c_{i_1} + c_{i_2}$, and computed recursively with a number of multiplications equal to the length of the obtained NAC plus 1 extra multiplication in $GF(3^m)$.

In this paper, we propose inversion algorithm using normal basis representation for the field elements in $GF(3^m)$. In comparison with [11]-based algorithms, our algorithm has lower IC w.r.t the number of required multiplications and avoids the subfield inversion. Thus, our algorithm relies on only two field operations (3^j*-th* powers and NB multiplications) to compute the inverse. Algorithms based on [11] require extra field operations to compute the subfield inversion, thus, from hardware perspective our algorithm is regular, modular and suitable for VLSI implementation.

In comparison with NAC-based algorithms, using our optimal decomposition method for $(m-1)$ of $GF(3^m)$ and the reliance on small and known chains for the suitable factors further reduce the computational complexity in polynomial-time for inversion. By decomposing $(m-1)$ into several factors and a remainder h, using the SAC for a suitable factor r_1 (C_{r_1}) with h appropriately selected to belong to it, the inverse is computed fast. Note that C_{r_1} is an optimal SAC since r_1 is quite small compared to $(m-1)$. In NAC-based algorithms there is a need to store a number of variables during the running time equal to the chain length for $(m-1)$, however, using our algorithm such a number is reduced to a value that equal to the chain length for r_1.

This paper is organized as follows: In Section 2, we provide a brief mathematical background on finite fields. In Section 3, we show the relevant work and preview the available inversion methods. In Section 4, we present our proposed work and show the derived algorithms. In Section 5, we provide analysis and the obtained results. Lastly in Section 6 we draw the conclusion.

2 Mathematical Background

Odd-characteristic extension field is a Galois field with odd prime-power number of elements (field's order). In the literature, such a field is denoted by $GF(p^m)$, where $p > 2$ is the field's characteristic and $m > 1$ is the field's extension degree relative to the base field (subfield) $GF(p)$[1].

Definition 1 *Let $f(x)$ be a polynomial in $GF(p)[x]$ of degree $m > 0$. $f(x)$ is irreducible over $GF(p)$ if it has no proper factor over $GF(p)$, equivalently, if it has at least one root that is not in a proper subfield of $GF(p^m)$.*

Let $\mathfrak{R} = GF(p)[x]$ be a ring of polynomials in x with coefficients in the field $GF(p)$. Let $\kappa = \frac{\mathfrak{R}}{f(x)}$ denote the quotient ring. Now if $f(x)$ is irreducible over $GF(p)$ of degree $m > 0$ then κ is finite field with p^m elements and it is isomorphic to finite field $GF(p^m)$.

In $GF(p^m)$, for every value of m there exist at least one irreducible polynomial $f(x)$. Such a polynomial is necessary for generating the elements of the field. The elements are represented using any suitable basis such as NB or polynomial basis representation (PB), etc [21].

Definition 2 *Given a basis element $\xi \in GF(p^m)$ that is a root of $f(x)$, the set of basis elements*

$$\mathbb{N} = (\xi^{p^0}, \xi^{p^1}, \cdots, \xi^{p^{m-2}}, \xi^{p^{m-1}})$$

defines a normal basis for $GF(p^m)$, where none of its subsets adds to zero. Thus, the basis elements in $\mathbb{N}$ are linearly independent.

Using NB, a nonzero element $\alpha \in GF(p^m)$ can be expressed as

$$\alpha = \sum_{i=0}^{m-1} a_i \xi^{p^i} \quad : a_i \in GF(p).$$

In vector form α is given by $(a_0 a_1 \cdots a_{m-2} a_{m-1})_p$. In NB by convention, the least significant coefficient a_0 is located on left-hand side, whereas a_{m-1} is located on right-hand side.

From Fermat's little theorem, for a nonzero element $\alpha \in GF(p^m)$ we have $\alpha^{p^m-1} = 1$. Thus, diving both sides by α implies $\alpha^{p^m-2} = \alpha^{-1}$. Therefore, given

$$\alpha = \sum_{i=0}^{m-1} a_i \xi^{p^i} = (a_0 a_1 \cdots a_{m-2} a_{m-1})_p, \qquad (1)$$

then

$$\alpha^p = \sum_{i=0}^{m-1} a_i \xi^{p^{i+1}} = (a_{m-1} a_0 \cdots a_{m-3} a_{m-2})_p. \qquad (2)$$

[1] For detailed review on finite fields refer to [20].

From (1) and (2), in using NB the p^j-*th* powers are simply reduced to j-*th* right cyclic-shifts. In other words, raising a field element represented using NB to a prime-power, where p is the prime, is a linear operation and requires a free execution-time. On the contrary, in general, NB multiplication requires execution-time higher than that in some other representation bases [22].

From the properties of finite fields, a positive integer r can be defined by dividing the multiplicative group of the extension field $GF(p^m)$ on the multiplicative group of the subfield $GF(p)$, i.e., $r = \frac{p^m-1}{p-1}$, which is given by

$$r = \frac{p^m-1}{p-1} = p^{m-1} + p^{m-2} + \cdots + p + 1 \qquad (3)$$

In general, the unique properties of r are useful for different applications in finite fields, and specifically, r is useful in computing field inversion as we shall see in next Section.

Definition 3 *An addition chain for a positive integer r is a chain* (sequence) *of elements* (integers) *of length l. The last chain-element r is obtained by the sum of previous chain-elements [23].*

Short addition chain (SAC) is the shortest of such addition chain denoted as C_r. Mathematically, C_r is given by

$$C_r = (c_0, c_1, \cdots, c_{l-1}, c_l), \qquad (4)$$

with $c_0 = 1$, $c_l = r$ and the i^{th} chain-element is given by $c_i = c_{i_1} + c_{i_2}$ for $(0 \le i, i_1, i_2 \le l)$ and $(i > i_1, i_2)$. C_r is associated with another short sequence of integer pairs, with each pair representing the i^{th} subsequent chain-element c_i in C_r, and is given by

$$P_r = \big((c_{i_1}, c_{i_2}) \mid 0 \le i_1, i_2 \le l-1\big).$$

For example, given the positive integer $r = 13$, then

$$C_{13} = (1, 2, 3, 6, 7, 13) \qquad (5)$$

that is associated with

$$P_{13} = \big((1,1), (2,1), (3,3), (6,1), (7,6)\big),$$

which follows the governing rule $c_i = c_{i-1} + c_{i-1} = 2c_{i-1}$ for $i \in \{1,3\}$, $c_i = c_{i-1} + c_{i-2}$ for $i \in \{2,5\}$, and $c_i = c_{i-1} + c_{i-4}$ for $i = 4$. C_r, given that $r = 13$, can also be expressed as $C_{13} = (1, 2, 4, 8, 12, 13)$ using a different governing rule.

Given a nonzero $\alpha \in GF(p^m)$, C_r can be used to compute (α^{p^r-1}) term. Such a term is necessary for computing inversion. To make discussion general, assume C_r is given by

$$C_r = (c_0, c_1, c_2, c_3, c_4), \qquad (6)$$

where $c_0 = 1$ and $c_4 = r$. Given that the chain-elements $c_1 = (c_0 + c_0)$, $c_2 = (c_1 + c_0)$, $c_3 = (c_2 + c_2)$, and last element $c_4 = (c_3 + c_2)$, then, the integer pairs sequence is given by

$$P_r = \big((c_0, c_0), (c_1, c_0), (c_2, c_2), (c_3, c_2)\big).$$

The general steps in computing $(\alpha^{p^r-1}) = (\alpha^{p^{c_4}-1})$ term using C_r in (6) are given as follows

$$\begin{aligned}
\alpha^{p^{c_1}-1} &= (\alpha^{p^{c_0}-1})^{p^{c_0}} \times \alpha^{p^{c_0}-1} \\
\alpha^{p^{c_2}-1} &= (\alpha^{p^{c_1}-1})^{p^{c_0}} \times \alpha^{p^{c_0}-1} \\
\alpha^{p^{c_3}-1} &= (\alpha^{p^{c_2}-1})^{p^{c_2}} \times \alpha^{p^{c_2}-1} \\
\alpha^{p^{c_4}-1} &= (\alpha^{p^{c_3}-1})^{p^{c_2}} \times \alpha^{p^{c_2}-1}
\end{aligned} \qquad (7)$$

From (7), it is apparent that 4 multiplications in $GF(p^m)$ are required to compute the term. This exactly equal the length of C_r which is given by the number of commas separating its elements.

Note that the required number of multiplications is also given by $[\ell(r) + \omega(r) - 2]$. Assume $p = 3$, $r = 13$ and C_{13} as given in (5). Using similar steps to that in (7), $(\alpha^{p^r-1}) = (\alpha^{3^{13}-1})$ is computed with 5 multiplications in $GF(3^m)$. This is because $[\ell(13) + \omega(13) - 2] = 5$ which is equal to C_{13} length.

3 Relevant Work

The method in [11] is commonly used in the literature to solve inversion in $GF(p^m)$, including the extension field $GF(3^m)$. Many authors followed this method with further attempts to improve it. It works as follows: consider (3), then

$$(r-1) = p^{m-1} + p^{m-2} + \cdots + p^2 + p^1,$$

which is the *p-dic* representation of $(r-1) = (11\ldots110)_p$. Given a nonzero $\alpha \in GF(p^m)$, its inverse based on [11] is given by

$$\alpha^{-1} = (\alpha^r)^{-1} \times \alpha^{(r-1)} = (\alpha^r)^{-1} \times \alpha^{p^{m-1}+\cdots+p^2+p^1}. \qquad (8)$$

The inverse in (8) is computed using the following steps:

Step 1. $GF(p^m)$-Exponentiation to get $\alpha^{(r-1)}$ term
Step 2. $GF(p^m)$-Multiplication to get $\alpha^r = \alpha.\alpha^{(r-1)}$
Step 3. $GF(p)$-Inversion to get $\beta = (\alpha^r)^{-1}$ term
Step 4. $GF(p^m)$-Multiplication to get $\alpha^{-1} = \beta.\alpha^{(r-1)}$

Note that α^r is a subfield element, i.e., $\alpha^r \in GF(p)$. This evident from Step 3 above for subfield inversion. Although in $GF(3^m)$ the subfield inversion is not costly, however, some extra overhead still necessary to perform such operation in the main inversion algorithm.

When r (3) is used as the exponent of an extension field element α, we basically compute its **norm**. The norm is a map from an extension field to a subfield element. Note also that the computation of $\alpha^{(r-1)}$ term requires the computation of the *j-th* powers using PB, or the *j-th* right cyclic-shifts using NB, by following similar steps as in (2).

To compute $\alpha^{(r-1)}$ term in Step 1 above using the straightforward approach, $(m-2)$ multiplications in $GF(p^m)$ are required as directly evident from (8). Thus, with extra two multiplications in $GF(p^m)$ and a subfield inversion the multiplicative inverse is computed. Thus, using m multiplications in $GF(p^m)$ plus a subfield inversion α^{-1} is computed, which is the inversion cost (IC) of such approach.

In [14] the algorithm was proposed mainly for use with PB and, $GF(p^m)$ is generated using irreducible polynomial of the form $f(x) = x^m + \sum_{i=m-1}^{0} f_i x^i$ defined over $GF(p)$. In using PB, the Frobenius map is an indispensable operation in inversion algorithms necessary for generating p^j-*th* powers. The *j-th* iterate of Frobenius map for a nonzero $\alpha \in GF(p^m)$ is given by

$$\alpha^{p^j}(x) = \left(\sum_{i=0}^{m-1} a_i x^i\right)^{p^j} = \sum_{i=0}^{m-1} a_i x^{ip^j} \mod f(x). \tag{9}$$

In [14], the *j-th* iterate of Frobenius map is computed using $M \times M$ matrix with complexity of m^2 multiplications in the subfield. The algorithm was slightly modified to perform as many subsequent *p-th* powers as possible between $GF(p^m)$-multiplications, in order to reduce the required number of the *j-th* iterates of Frobenius map to a logarithmic value. Other authors [24] proposed fast implementation of Frobenius map operation using different approach.

It is mentioned in the literature that using SACs is the best alternative for computing $\alpha^{(r-1)}$ term with a logarithmic value of $GF(p^m)$-multiplications. Although there is no dedicated algorithm to compute the term in $GF(p^m)$, however, most authors assumed the use of the ingenious method that proposed earlier in [9] for binary extension fields $GF(2^m)$. Such a method is not optimal in most cases except when r is a power of 2 and the method requires some modification for suitability with $GF(3^m)$. Using such a method, $\alpha^{(r-1)}$ term, where $r = 2^{(m-1)}$, is computed with polynomial-time and requires

$$\left[\ell(m-1) + \omega(m-1) - 2\right] \tag{10}$$

multiplications in $GF(2^m)$. Thus in [14] for any p, with extra two multiplications in $GF(p^m)$ and a subfield inversion, using SACs approach for computing inversion, the IC is given by

$$\left[\ell(m-1) + \omega(m-1)\right] \; GF(p^m)\text{-Multiplications},$$

$$\text{Subfield } GF(p)\text{-Inversion},$$

$$\left[\ell(m-1) + \omega(m-1) - 1\right] \; \text{Frobenius-Maps}. \tag{11}$$

In [12], SACs approach was used again for inversion, but the authors have focused on accelerating Frobenius map operation for use with PB. Referring to (9), the authors focused on the affected terms by the action of the *j-th* iterate of Frobenius map which are not kept fixed, such as x^{ip^j} terms for $(1 \le i \le m-1)$.

By utilizing field polynomial of special type, namely the irreducible binomial $f(x) = x^m - w$, the authors expressed the affected terms as

$$x^e \equiv w^q x^s \mod f(x),$$

where $s \equiv e \mod m$, $q = \frac{e-s}{m}$ and $e = ip^j$. The authors in [12] and based on Corollary 2, they further reduced the expression to

$$x^e \equiv w^q x^i \mod f(x).$$

From above discussion, the *j-th* iterate of Frobenius map is computed faster than in [14], however, the required number of multiplications in $GF(p^m)$ has not changed. Thus, the IC is identical to that given in (11).

The advantage of using NB for field inversion in $GF(p^m)$, more specifically in $GF(3^m)$, is utilized in [13]. From Section 2, the *j-th* iterate of the Frobenius map in using NB is simply reduced to either *j-th* right cyclic-shifts in software, or *j-th* permutations in hardware (no need for dedicated hardware). Thus, the *j-th* iterates of Frobenius map are operations associated with free execution-time when using NB for the field elements.

In [13], the IC is exactly identical to that in (11), however, Frobenius maps are reduced to free execution-time cyclic-shift operations as a result of using NB. Currently, accelerating NB multiplication is an active research topic [25]. This is because NB multiplication has a higher execution-time than multiplication using PB or any other basis. Thus, a minor reduction in the required number of NB multiplications has a noticeable effect in improving the performance in inversion algorithms.

Given the fact that the problem of finding a SAC for a large r value is merely an optimization problem, the authors in [18] proposed a probabilistic heuristic strategy which produce NACs for r values up to 512-bit. In their approach they divided the search problem into two phases. Firstly, in phase 1, the computation of the SAC for large r is reduced to the computation of an addition sequence that consists of a set of windows, or integers much smaller than r. Secondly, in phase 2, four different search criteria were used, although not clearly defined, to reduce the length of the obtained addition sequence, in attempt to find the desired NAC for r.

In [19], the authors proposed other probabilistic heuristic strategy for such optimization problem. By using their artificial immune system (AIS), they found moderate length NACs for the desired r values. Such NACs are then used to compute inversion in $GF(2^m)$ in applications other than cryptography. Similar to other NAC-based algorithms, their proposed AIS search engine requires execution-time that falls within the domain of NP-hard problems.

By again utilizing AIS, the authors in [15] proposed an improved probabilistic heuristic strategy to find NACs for a broad range of r values in $GF(2^m)$. They first considered small r values up to 12-bit for comparisons against other deterministic heuristic strategies. Since the SACs for such small r values are exactly known through exhaustive search, the performed comparisons confirmed the quality of the search strategy of their proposed solution as claimed by the authors. Subsequently, using two processing phases, as in [18], they firstly combined sliding window strategy together with AIS to utilize the efficient partitioning for large r values into smaller values, or windows. Then, they utilized AIS to group the obtained windows into a single addition sequence. Finally, their obtained NACs for r values applicable in cryptography were again compared against the chains of some known deterministic strategies.

4 Proposed Work

On the one hand, [11]-based inversion algorithms are polynomial-time, however, they still have relatively high IC which is amenable to further reduction. On the other hand, although NACs-based algorithms initially seem to have a comparable IC with our previously proposed inversion algorithm [26] and the one herein, however, the search for such NACs requires additional execution time especially for large m values as in cryptographic applications. In addition, such algorithms require storage for intermediate computations during run-time equal to the length of the sought NACs.

In this paper, we intend to fill the gap by proposing a modular low IC inversion algorithm in $GF(3^m)$ in comparison with [11]-based algorithms. Also, a reduction in execution time compared to NACs-based algorithms is achieved as our inversion algorithm relies on small and known addition chains for a suitable factor of m. In addition, the storage requirements in most cases and the IC in some cases are reduced using our inversion algorithm compared to NACs-based algorithms.

Our algorithm is based on Fermat's approach for inversion and proposed for $GF(3^m)$ using NB for the field elements. The algorithm implements our proposed optimal decomposition method (ODM) for m of $GF(3^m)$. Using such a method $(m-1)$ is decomposed into several factors and a remainder h, i.e., $(m-1) = \prod_{j=1}^{k} r_j + h$. Thus, in using our algorithm we need the SAC for a suitable factor, r_1, which include h as an element. Such a SAC is much smaller than that for m which has optimal length and easy to find a priori.

Based on above discussion, in applications where m is unknown a priori and change frequently, rather than searching for NACs for large m values, which requires extra execution time, in using our algorithm the suitable factor that is much smaller than m has its SAC a priori known and available in memory. Thus, fetching such a SAC during run-time costs nothing and the only thing that matter is the required NB multiplications (which represent IC) but not the extra search time needed for the sough NAC.

Our proposed inversion algorithm can be described as follows: Assume a nonzero $\alpha \in GF(3^m)$ in which its inverse $\alpha^{-1} = \alpha^{3^m-2}$ is required, the extension degree $(m-1) = \prod_{j=1}^{k} r_j + h$, i.e., decomposed into several factors and a remainder h, and since the expression

$$3^m - 2 = 3 \times \left(3^{(m-1)} - 1\right) + 1, \qquad (12)$$

by substituting $(m-1)$ in (12) we have

$$3^m - 2 = 3 \times \left(3^{\prod_{j=1}^{k} r_j + h} - 1\right) + 1. \qquad (13)$$

Expanding the exponent in (13) gives us

$$3^m - 2 = 3 \times \left[3^h \times (3^{r_1 \times \cdots \times r_k} - 1) + (3^h - 1)\right] + 1. \quad (14)$$

Finally, rearranging terms in (14) gives us

$$3^m - 2 = 3 \times \left[e \times 3^h \times (3^{r_1} - 1) + (3^h - 1)\right] + 1, \quad (15)$$

whereby the expression for variable e is given in the following. Therefore, based on (15) $\alpha^{-1} = \alpha^{3^m-2}$ is given by

$$\alpha^{-1} = \alpha^{3^m-2} = \left(\alpha^{3^{(m-1)}-1}\right)^3 \times \alpha =$$

$$\left(\alpha^{3^{\prod_{j=1}^{k} r_j + h}-1}\right)^3 \times \alpha = \left((\alpha^{3^{r_1 \times \ldots \times r_k}-1})^{3^h} \times (\alpha^{3^h-1})\right)^3 \times \alpha$$

$$= \left(((\alpha^{3^{r_1}-1})^e)^{3^h} \times (\alpha^{3^h-1})\right)^3 \times \alpha$$

$$= \left((\alpha^{3^{r_1}-1})^e\right)^{3^{h+1}} \times \left(\overbrace{(\alpha^{3^h-1})^3 \times \alpha}^{f-\text{ term}}\right),$$

given that

$$e = \left((3^{r_1})^{r_2-1} + \cdots + 1) \cdots ((3^{r_1 \times \cdots \times r_{k-1}})^{r_k-1} + \cdots + 1)\right). \quad (16)$$

Theorem 1 *Given that* $(m-1) = \prod_{j=1}^{k} r_j + h$, *with* h *appropriately selected to belong to* C_{r_1}, *the SAC for* r_1, *then, in computing* $(\alpha^{3^{r_1}-1})$ *term the availability of* (α^{3^h-1}) *term as an intermediate value is guaranteed, and the IC using our algorithm is given by*

$$\left(\sum_{j=1}^{k}\Big[\ell(r_j)+\omega(r_j)-2\Big]+1\right)\ GF(3^m)\text{-Multiplications.} \tag{17}$$

Proof. From (16), the inverse is given by

$$\alpha^{-1} = \left(\underbrace{(\alpha^{3^{r_1}-1})^e}_{k-\text{term}}\right)^{3^{h+1}} \times \left(\overbrace{(\alpha^{3^h-1})^3 \times \alpha}^{f-\text{ term}}\right) \tag{18}$$

For now, forget the computational cost of $(\alpha^{3^{r_1}-1})$ term by itself. The cost of computing $(\alpha^{3^{r_1}-1})^e$ by expanding e in (16) is given by

$$\sum_{j=2}^{k}\Big[\ell(r_j)+\omega(r_j)-2\Big], \tag{19}$$

and its computational details are given as follows: The cost of $(\alpha^{3^{r_1}-1})^{(3^{r_1})^{r_2-1}+\cdots+(3^{r_1})^0} = (\alpha^{3^{r_1\times r_2}-1})$ term is $[\ell(r_2)+\omega(r_2)-2]$ multiplications. Then, the cost of $(\alpha^{3^{r_1\times r_2}-1})^{(3^{r_1\times r_2})^{r_3-1}+\cdots+(3^{r_1\times r_2})^0} = (\alpha^{3^{r_1\times r_2\times r_3}-1})$ is $[\ell(r_3)+\omega(r_3)-2]$ multiplications. Finally, the cost of

$$(\alpha^{3^{r_1\times\cdots\times r_{k-1}}-1})^{(3^{r_1\times\cdots\times r_{k-1}})^{r_k-1}+\cdots+(3^{r_1\times\cdots\times r_{k-1}})^0}$$

$$= (\alpha^{3^{r_1\times\cdots\times r_k}-1}) = (\alpha^{3^{r_1}-1})^e,$$

is $[\ell(r_k)+\omega(r_k)-2]$ multiplications. Thus, the sum of all costs is exactly as given in (19) above.

Now, returning to $(\alpha^{3^{r_1}-1})$ term, its cost can be given as follows: Let the SAC for r_1 given by

$$C_{r_1} = \{c_0, c_1, c_2, c_3\},$$

where $c_0 = 1$ and $c_3 = r_1$. In addition, let the sequence of integer pairs of C_{r_1} given by

$$P_{r_1} = \{(c_0,c_0),(c_1,c_1),(c_2,c_2)\},$$

following the rule $c_1 = c_0 + c_0$, $c_2 = c_1 + c_1$, and $c_3 = c_2 + c_2$. Assume $\beta = \alpha^2$ is available, then it follows that $(\alpha^{3^{r_1}-1}) = (\beta^{3^{c_3}-1})$ is given by

$$\begin{aligned}
(\beta^{3^{c_0}-1})^{3^{c_0}} \times (\beta^{3^{c_0}-1}) &= (\beta^{3^{c_0+c_0}-1}) = (\beta^{3^{c_1}-1})\\
(\beta^{3^{c_1}-1})^{3^{c_1}} \times (\beta^{3^{c_1}-1}) &= (\beta^{3^{c_1+c_1}-1}) = (\beta^{3^{c_2}-1})\\
(\beta^{3^{c_2}-1})^{3^{c_2}} \times (\beta^{3^{c_2}-1}) &= (\beta^{3^{c_2+c_2}-1}) = (\beta^{3^{c_3}-1})
\end{aligned} \tag{20}$$

From (20), 3 multiplications are necessary to compute $(\alpha^{3^{r_1}-1})$ term using C_{r_1}, which represent the length of C_{r_1}. Thus, for any positive integer r_1, $(\alpha^{3^{r_1}-1})$ term requires number of multiplications given by the length of C_{r_1}. Mathematically, this number is given by

$$\Big[\ell(r_1)+\omega(r_1)-2\Big], \tag{21}$$

multiplications. Given $c_i = h$ for $i \in [0,1,2,3]$, then $(\alpha^{3^h-1}) = (\beta^{3^{c_i}-1})$ is one of the intermediate results as evident from (20) above. Thus, the computational cost of f-term in (18) which is equal to (α^{3^h-1}) is saved as a result of computing it in parallel with k-term. With 1 extra multiplication necessary to join the terms in (18), the IC of our proposed inversion algorithm is exactly as given in Theorem 1 above. □

From Theorem 1 above, we computed f-term in parallel with k-term to improve IC, however, this incurs slight increase in space complexity of our algorithm. Since the IC is independent of the remainder h that is not restricted in value, in using our decomposition method compared to other methods a wide range of $GF(3^m)$ are associated with minimal IC.

In the following we introduce, through a running example, our proposed main inversion algorithm along with other proposed auxiliary algorithms necessary for its functioning. We consider $GF(3^{167})$ that is of particular interest for use in cryptographic applications in ECC. Assume $\alpha \in GF^*(3^{167})$ in which its inverse is required, passing α and $m = 167$ as inputs to our main inversion algorithm (**Algorithm 1**), then we have $ms = (m-1) = 166 = (10100110)_2$ in binary.

Algorithm 1 *InvMain* Inversion Algorithm

```
Input: $\alpha \in GF^*(3^m)$, extension degree $m$
Output: $\psi = \alpha^{-1} \in GF^*(3^m)$
Initial: $ms := (m-1)$;
if ($ms == 2^k$ or $\omega(ms) == 2$) then
    return $\psi := ChnInv(\alpha, C_{ms}, P_{ms}, 0)$
else
    $[(r_1,n,h), C_{r_1}, P_{r_1}] := OpDecom(ms)$;
    $[\sigma,\rho] := ChnInv(\alpha, C_{r_1}, P_{r_1}, h)$;
    $\tau := FsCalc(\sigma, n)$;
    if ($h == 0$) then
        return $\psi := (\tau \times \rho)$
    else
        return $\psi := \left((\tau)^{3^{h+1}} \times \rho\right)$
    end if
end if
```

Given that ms is neither a power of 2 nor a value with Hamming weight equal to 2, then it should be passed as input to our proposed auxiliary optimal decomposition algorithm (**Algorithm 2**).

Algorithm 2 *OpDecom* Decomposition Algorithm

```
Input: $ms$
Output: $[(r_1,n,h),C_{r_1},P_{r_1}]$
Initial: $t := ms,\ st := \sqrt{t},\ cnt := 0;$
if $t$ is odd: $A \leftarrow A_o = \{1,3,\cdots,u := \lceil st \rceil\}$. If $st$ is odd, $u := \lfloor st \rfloor$;
if $t$ is even: $A \leftarrow A_e = \{2,4,\cdots,u := \lfloor st \rfloor\}$. If $st$ is odd, $u := \lceil st \rceil$;
for all $A[i]$ in the selected $A$ do
    $r_1 := t - A[i];$
    while $(r_1 \neq 2^k$ and $2|r_1$ and $\lceil r_1/2 \rceil \geq A[i])$ do
        $r_1 := \left(\frac{r_1}{2}\right),\ cnt := cnt + 1;$
    end while
    save resulted 3-tuple $(r_1,cnt,A[i])$ in array;
    $cnt := 0;$
    next $i$;
end for
find suitable $(r_1,cnt,A[i])$ and fetch $[C_{r_1},P_{r_1}]$;
return $[(r_1,n := 2^{cnt},h := A[i]),C_{r_1},P_{r_1}]$
```

Using our example, the output of Algorithm 2 is given by

$$[(r_1,n,h),C_{r_1},P_{r_1}] = [(10,16,6),C_{10},P_{10}],$$

where the 3-tuple $(10,16,6) = 10 \times 16 + 6 = ms$ and the SAC for r_1 is given by

$$C_{r_1} = C_{10} = \{1,2,4,6,10\}, \tag{22}$$

and

$$P_{r_1} = P_{10} = \{(1,1),(2,2),(4,2),(6,4)\}, \tag{23}$$

which follows the rule $c_i = c_{i-1} + c_{i-1}$ for $i \in \{1,2\}$, and $c_i = c_{i-1} + c_{i-2}$ for $i \in \{3,4\}$, where $c_0 = 1$ and $c_4 = 10$. By passing α, C_{10}, P_{10} and h as inputs to our proposed auxiliary algorithm (**Algorithm 3**), the computation then proceeds as follows:

$$\begin{aligned}
(\alpha^{3^1-1})^{3^1} \times (\alpha^{3^1-1}) = (\alpha^8) = (\alpha^{3^2-1}) \\
(\alpha^{3^2-1})^{3^2} \times (\alpha^{3^2-1}) = (\alpha^{80}) = (\alpha^{3^4-1}) \\
(\alpha^{3^4-1})^{3^2} \times (\alpha^{3^2-1}) = (\alpha^{728}) = (\alpha^{3^6-1}) \\
(\alpha^{3^6-1})^{3^4} \times (\alpha^{3^4-1}) = (\alpha^{59048}) = (\alpha^{3^{10}-1})
\end{aligned} \tag{24}$$

The output of Algorithm 3 based on our example is given by $[\xi_{c_l}, f]$. The first value $\xi_{c_l} = (\alpha^{3^{r_1}-1}) = (\alpha^{3^{10}-1})$. The second value $f = (\alpha^{3^h-1})^3 \times \alpha = (\alpha^{3^6-1})^3 \times \alpha$, or the *f-term* that given in (18). Given that $h = 6$ is an element in the chain $C_{r_1} = C_{10}$, and the computation of f-term is parallel with k-term, then, the cost of both values, i.e., $[\xi_{c_l}, f]$, is only 4 multiplications in $GF(3^{167})$.

To compute the portion that represents other factors in $(m-1)$, other than r_1, which is given by the value $n = 16$, both ξ_{c_l} and n are passed as inputs to our proposed auxiliary algorithm (**Algorithm 4**).

Algorithm 3 *ChnInv* Algorithm

```
Input: $\alpha$, $C_v$, $P_v$, $\kappa$
Output: $\phi = \alpha^{-1}$ if $ms$ not decomposed: else $[\xi_{c_l}, f]$
Given: $\xi_{c_i}(\alpha) = \alpha^{3^{c_i}-1}$, $\xi_{c_{i_1}+c_{i_2}}(\alpha) = [\xi_{c_{i_1}}(\alpha)]^{3^{c_{i_2}}} \times \xi_{c_{i_2}}(\alpha)$
Initial: $\kappa := 0$ if $ms$ not decomposed: else $\kappa := h$,
         $l := length(C_v), \xi_{c_0} := \alpha^2;$
for $i := 1$ to $l$ do
    $\xi_{c_i}(\alpha) := [\xi c_{i_1}(\alpha)]^{3^{c_{i_2}}} \times \xi_{c_{i_2}}(\alpha);$
    if $\kappa == c_{i-1}$ then
        $f := \left[(\xi_{c_i})^3 \times \alpha\right]; \Longleftarrow (f\text{-term})$
    end if
end for
if $\kappa \neq 0$ then
    return $[\xi_{c_l}, f]$
end if
return $\phi := [\xi_{c_l}^3 \times \alpha]$
```

Algorithm 4 *FsCalc* Algorithm

```
Input: $\lambda = \xi_{c_l}$, $r_1$, $n = \prod_{j=2}^{k} r_j : r_j = (1m_{q_j-2}^{(j)} \ldots m_0^{(j)})_2$
Output: $\theta = \lambda^e = (\alpha^{3^{r_1}-1})^e$
Initial: $r := 1;$
for $j := 2$ to $k$ do
    $\theta := \lambda;$
    $r := r \times r_{j-1};$
    for $i := q_j - 2$ to 0 do
        $\theta := \theta \times \theta^{3^{r3^i}};$
        if $m_i^{(j)} == 1$ then
            $\theta := \lambda \times \theta^{3^{r3^i}};$
        end if
    end for
    $\lambda := \theta;$
end for
return $\theta$
```

Since $n = 16$ is a power of 2, then we use it as if it a single factor. Thus, the outer **for** loop which depend on the number of factors in $(m-1)$ is executed only once. In addition, the loop counter i in the inner **for** loop which depends on the binary length of each factor will loop from 3 to 0, simply 4 loops. This because $n = 16 = (10000)_2$, thus, the maximum value for the loop counter is given by $i = (q-2) = (6-2) = 3$, where q is the binary length of n. Since the **if** statement is not going to be satisfied, and variable $r = r \times r_{j-1} = r \times r_1 = 1 \times 10 = 10$, then computing the output θ using $a = \xi_{c_l} = (\alpha^{3^{59048}})$ based on our example proceeds as follows:

$$\begin{aligned}
(a) \times (a)^{3^{10\times3^3}} &= a^{(3^{10\times3^3}+1)} = b \\
(b) \times (b)^{3^{10\times3^2}} &= b^{(3^{10\times3^2}+1)} = c \\
(c) \times (c)^{3^{10\times3^1}} &= c^{(3^{10\times3^1}+1)} = d \\
(d) \times (d)^{3^{10\times3^0}} &= d^{(3^{10\times3^0}+1)} = \theta
\end{aligned} \tag{25}$$

Algorithm 4 output $\theta = (\alpha^{3^{r_1}-1})^e$ is given by

$$\theta = (\alpha^{3^{10}-1})^{(3^{10\times 3^3}+1)(3^{10\times 3^2}+1)(3^{10\times 3^1}+1)(3^{10\times 3^0}+1)}. \quad (26)$$

With the availability of $(\alpha^{3^{10}-1})$ term, θ is computed with 4 multiplications in $GF(3^{167})$ that represented by the '+' signs in the expression given in (26) and as evident from (25) above.

Given the availability of θ with $h \neq 0$, by considering Algorithm 1, we need to perform $(h+1)$ right cyclic-shifts on $\tau = \theta$ and multiply it with $\rho = f$-term to get $\psi = \alpha^{-1}$, the final answer, as evident also in (18) above. Thus, the IC for computing $\psi = \alpha^{-1} = (\alpha^{3^{167}-2})$, in terms of the number of required multiplications, is given by 9 multiplications in $GF(3^{167})$.

Referring to Algorithm 1, in the other case where ms is either a power of 2 or a value with Hamming weight equal to 2, finding its SAC is very quick and straightforward. Although $ms = (m-1)$ is a large value, the search space for such a SAC is not a NP-hard problem. In other words, the SAC in such a case is simply the power of 2 SAC, i.e., $\{1,2,4,8,16,\cdots\}$ or the power of 2 SAC with the addition of one of its elements to the last chain-element. Thus, in cases where ms is not decomposed the cost of computing a field inverse was further reduced using our inversion algorithm.

Fig. 1 illustrates the relation between our main inversion algorithm (Algorithm 1) with the auxiliary algorithms (its main building blocks) in the form of block diagram.

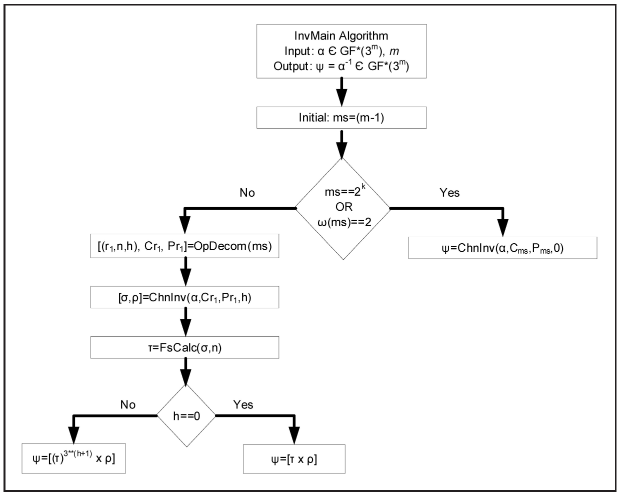


Fig. 1: Main Inversion Algorithm Block Diagram

5 Analysis and Results

In using Fermat's inversion approach with NB, the IC is defined as the performance measure that measures the ability of an inversion algorithm to compute the inverse with the minimal number of multiplications. Therefore, any algorithm that require less number of multiplications is associated with less IC, and thus has a faster execution-time relative to its counterparts. Although this is absolutely valid when comparing our algorithm with [11]-based algorithms, the scenario is completely different when comparing it with NAC-based algorithms, and this is because the search time for the sought NAC must be taken into account.

Recall that our algorithm is proposed mainly for $GF(3^m)$ using NB (Algorithm 1), as well as, it implements our ODM method for m of $GF(3^m)$ and relies on known SACs to compute the inverse, Table 1 show its IC performance compared to [11]-based inversion algorithms. The comparison is performed over a set of selected m values, or over a selected set of $GF(3^m)$ of particular interest in cryptographic applications in ECC.

Referring to Table 1, each m value that preceded by the Section symbol, §, has $ms = (m-1)$ equal to either a power of 2 or a value with Hamming weight of 2. In computing the inverse, such values are not decomposed and their SACs are directly used to solve inversion problem. The SACs for such values are easily obtained in a straightforward manner and known a priori as explained in previous Section.

From Table 1, in addition to the avoidance of subfield inversion (I_{sub}) operation using our inversion algorithm, the reduction in IC relative to [11]-based algorithms is up to 4 multiplications in $GF(3^m)$. This is a remarkable reduction in IC and in the execution-time of our algorithm. This because NB multiplications are associated with high execution-time when implemented in both software and hardware. Thus a minor reduction in the number of NB multiplications has a great effect on the overall performance.

Given that the search for NACs for large m values using heuristic methods is a NP-hard large search-space problem, in using our algorithm, the problem is reduced to the reliance on known SACs for the suitable factors, especially when m is optimally decomposed into several factors and a remainder h. Table 2 shows IC performance comparison of our algorithm against AIS algorithm [15], which is the best known NAC-based algorithm so far.

From Table 2, although the IC of our inversion algorithm initially seems equal to that of AIS algorithm, however, our algorithm is associated with much lower execution-time where the search-time for large chains is saved. In addition, m values that preceded by the Section symbol, §, are associated with less number of multiplications in $GF(3^m)$, or less IC compared to AIS inversion algorithm.

Consider $m = 320$. The inverse of $\alpha \in GF^*(3^{320})$ using our inversion algorithm requires 11 multiplications in $GF(3^{320})$. This because $ms = (m-1) = r_1 \times n + h$ which is equal to $319 = 19 \times 16 + 15$. Detailed steps are given as follows: $C_{r_1} = C_{19} = \{1,2,4,5,10,15,19\}$ which has $h = 15$ as one of its elements. Also, $n = 16$, the other

Table 1: Inversion Cost: Algorithm 1 *vs* [11] Based Algorithms

m of		Inversion Cost (IC)	
GF(3^m)	**OpDecom(ms)**	**Algm. 1**	**[11]-Based Algms.**
79	$9\times8+6$	8	11+I_{sub}
§97	C_{96}	7	9+I_{sub}
108	$13\times8+3$	9	12+I_{sub}
150	$9\times16+5$	9	12+I_{sub}
163	$5\times32+2$	9	11+I_{sub}
167	$10\times16+6$	9	12+I_{sub}
173	$20\times8+12$	9	12+I_{sub}
180	$11\times16+3$	10	13+I_{sub}
§193	C_{192}	8	10+I_{sub}
208	$24\times8+15$	10	14+I_{sub}
215	$13\times16+6$	10	13+I_{sub}
239	$14\times16+14$	10	14+I_{sub}
284	$17\times16+11$	11	14+I_{sub}
312	$19\times16+7$	11	15+I_{sub}
324	$5\times64+3$	10	13+I_{sub}

Table 2: Inversion Cost: Algorithm 1 *vs* AIS Algorithm [15]

m of		Inversion Cost (IC)	
GF(3^m)	**OpDecom(ms)**	**Algm. 1**	**AIS Algm. [15]**
96	$11\times8+7$	9	9
128	$15\times8+7$	10	10
160	$19\times8+7$	10	10
192	$23\times8+7$	11	11
224	$26\times8+15$	11	11
256	$15\times16+15$	10	10
288	$17\times16+15$	11	11
§320	$19\times16+15$	11	12
352	$21\times16+15$	11	11
384	$23\times16+15$	12	12
416	$25\times16+15$	11	11
§448	$27\times16+15$	11	12
480	$29\times16+15$	12	12
576	$35\times16+15$	12	12
§640	$39\times16+15$	12	13

factors in $(m-1)$, has $C_n = C_{16} = \{1,2,4,8,16\}$. Thus, computing n requires 4 multiplications in $GF(3^{320})$. From Section 2, using C_{19}, $(\alpha^{3^{r_1}-1}) = (\alpha^{3^{19}-1})$ term is computed with 6 multiplications in $GF(3^{320})$ (the length of C_{19}). Given that $h = 15 \in C_{19}$, then $(\alpha^{3^h-1}) = (\alpha^{3^{15}-1})$ term is available in intermediate computations, and thus, all we need is 1 extra $GF(3^{320})$ multiplication to get the final result. Thus, using our inversion algorithm the IC = $6+4+1 = 11$ multiplications in $GF(3^{320})$.

Alternatively, using AIS algorithm [15]

$$C_{319} = \{1,2,3,6,12,18,36,72,144,288,306,318,319\},$$

thus the IC $= 12$ multiplications in $GF(3^{320})$ necessary to compute the inverse. In addition to the higher number of multiplications necessary to compute inversion, in using AIS algorithm, extra search-time for C_{319} and storage for intermediate results are needed.

Eventually, in using [11]-based inversion algorithms, the IC is calculated by the direct application of (11), which requires 16 multiplications in $GF(3^{320})$ in addition to a subfield (I_{sub}) inversion.

The above discussion confirm the practicality of our proposal to accelerate field inversion in $GF(3^m)$ in comparison with other inversion algorithms. Thus, in executing the built-in curve-point operations of scalar multiplication algorithm that require field inversion, our proposal can be used to accelerate such algorithm and thus the concerned cryptographic application in ECC. To mention a few of such applications: elliptic curve diffie-hellman algorithm (ECDH) for key-exchange, elliptic curve el-gamal algorithm (ECEl-Gamal) for encryption.

6 Closing Remarks

In this paper, an inversion algorithm is proposed in ternary extension fields $GF(3^m)$ based on Fermat's theorem using normal basis representation for the field elements. Using our proposal, the extension degree $(m-1)$ of $GF(3^m)$ is decomposed into several factors and a remainder h to reduce the number of multiplications necessary for inversion, or the inversion cost.

Our proposal was compared against other algorithms that rely on different inversion methods. In comparison with the commonly used inversion algorithms (i.e, [11]-based algorithms), our algorithm has less inversion cost and avoids the subfield inversion. Thus, from hardware perspective our algorithm is regular, modular and suitable for VLSI implementation. Unlike the algorithms that search for near-optimal addition chains (NACs) for large $(m-1)$ values of $GF(3^m)$ to solve inversion, our proposal relied on known short addition chains for suitable factors much smaller than $(m-1)$ that include the remainder h as an element to solve inversion. Thus, combined with the decomposition for $(m-1)$, the inversion cost (IC) was further reduced in comparison with such algorithms.

The conducted comparisons confirmed the practicality of our inversion algorithm in comparison with its counterparts available in the literature. Our proposal, as far as we know, is the fastest polynomial-time inversion algorithm. It is suitable for accelerating cryptographic applications in elliptic curve cryptography (ECC).

Acknowledgment

The author would like to warmly thank peer reviewers for their help in reviewing this manuscript.

References

[1] S. Galbraith, "Supersingular curves in cryptography," 7th International Conference on Theory and Application of Cryptology and Information Security: Advances in Cryptology, pp. 495-513, Springer-Verlag 2001.

[2] M. Naseer and E. Savas, “Hardware implementation of a novel inversion algorithm,” *46th IEEE Midwest Symposium on Circuits and Systems*, vol. 2, pp. 798-801, 2003.

[3] M. Shiwei, H. Yuanling, P. Zhongqiao and C. Hui, “Fast Implementation for Modular Inversion and Scalar Multiplication in the Elliptic Curve Cryptography,” *2nd IEEE Symposium on Intelligent Information Technology Application*, vol. 2, pp. 488-492, 2008.

[4] K. Kobayashi and N. Takagi, “A Combined Circuit for Multiplication and Inversion in $GF(2^m)$,” *IEEE Transactions on Circuits and Systems*, volume 55, issue 11, pp. 1144-1148, Nov. 2008.

[5] K. Kobayashi, N. Takagi and K. Takagi, “An Algorithm for Inversion in $GF(2^m)$ Suitable for Implementation Using a Polynomial Multiply Instruction on $GF(2)$,” *18th IEEE Symposium on Computer Arithmetic*, vol. 6, pp. 105-112, 2007.

[6] M. Hasan, “Efficient computation of multiplicative inverses for cryptographic applications,” *15th IEEE Symposium on Computer Arithmetic*, vol. 6, pp. 66-72, 2001.

[7] Y. Li, G. Chen, Y. Chen and J. Li, “An Improvement of TYT Algorithm for $GF(2^m)$ Based on Reusing Intermediate Computation Results,” *Journal of Communications in Mathematical Sciences*, vol. 9, issue 1, pp. 277-287, Jan. 2011.

[8] N. Takagi, J. Yoshiki and K. Takagi, “A Fast Algorithm for Multiplicative Inversion in $GF(2^m)$ Using Normal Basis” *IEEE Transactions on Computers*, vol. 50, issue 5, pp. 394-398, May 2001.

[9] T. Itoh and S. Tsujii, “A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Basis,” *Journal of Information and Computing*, vol. 78, issue 7, pp. 171-177, July 1988.

[10] A. Reyhani-Masoleh, “Efficient algorithms and architectures for field multiplication using Gaussian normal bases,” *IEEE Transctions on Computers*, vol. 55, issue 1, pp. 34-47, January 2006.

[11] D. Hankerson, A. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography, page 67*, New York, USA: Springer-Verlag, 2004.

[12] D. Bailey and C. Paar, “Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography,” *Journal of Cryptology*, vol. 14, no. 3, pp. 153-176, March 2003.

[13] R. Granger, D. Page and M. Stam, “Hardware and software normal basis arithmetic for pairing-based cryptography in characteristic three,” *IEEE Transactions on Computers*, vol. 54, issue 7, pp. 852-860, July 2005.

[14] J. Guajardo and C. Paar, “Itoh-Tsujii Inversion in Standard Basis and Its Application in Cryptography and Codes,” *Designs, Codes and Cryptography Journal*, vol. 25, Issue 2, Springer-Verlag, pp. 1573-7586, Feb. 2002.

[15] N. Cruz-Cortes, F. Rodriguez-Henriquez and C. Coello, “An Artificial Immune System Heuristic for Generating Short Addition Chains,” *IEEE Trans. on Evol. Comp.*, vol. 12, no. 1, pp. 1-24, Jan. 2008.

[16] D. Gordon, “A survey of fast exponentiation methods,” *Journal of Algorithms*, vol. 27, no. 1, pp. 129146, Apr. 1998.

[17] D. Knuth, *The Art of Computer Programming*, 3rd. ed., Massachusetts, USA: Addison-Wesley, 1997.

[18] J. Bos and M. Coster, “Addition chain heuristics,” *proceedings of advances in cryptology*, vol. 435, pp. 400407, 1989.

[19] N. Cruz, F. Rodriguez and C. Coello, “On the optimal computation of finite field exponentiation,” *proc. of 9th conf. on Adv. in artif. intell.*, vol. 3315, pp. 747-756, 2004.

[20] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, University Press, Cambridge 1994.

[21] Standard specifications for public-key cryptography, draft version 1st ed. “http://grouper.ieee.org/groups/1363/”: IEEE standards documents, November 2009.

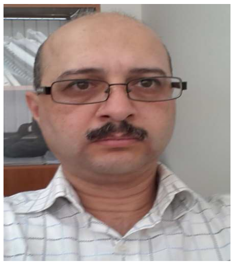
[22] A. Reyhani-Masoleh, “Efficient algorithms and architectures for field multiplication using Gaussian normal bases,” *IEEE Transctions on Computers*, vol. 55, issue 1, pp. 34-47, January 2006.

[23] F. Dong and Y. Li, “A Novel Shortest Addition Chains Algorithm Based on Euclid Algorithm,” 4th Intl. Conf. on Wireless Commus., Networking and Mobile Computing, pp. 1-4, Oct. 2008.

[24] Walid Mahmoud, “Fast Computation of Frobenius Map Iterates in Optimal Extension Fields,” 9th Annual International Joint Conferences on Computer, Information, Systems Sciences, and Engineering (TENE 2013), December 12-14, 2013, CT, USA.

[25] A. Reyhani-Masoleh and M. Hasan, “Fast normal basis multiplication using general purpose processors,” *IEEE Transactions on Computers*, vol. 52, issue 11, pp. 1379-1390 , Nov. 2003.

[26] Walid Mahmoud, “Reduced-Latency Algorithm for Finite Field Inversion in $GF(2^m)$,” *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 3, Issue 9, pp. 7857-7859, September 2014.



Walid Mahmoud received the B.E.Sc degree from University of Western Ontario, London, Ontario, Canada. His M.A.Sc and Ph.D degrees received from University of Windsor Ontario, Windsor, Ontario, Canada. He designed and implemented many academic research projects during his academic studies in Canada. Dr. Mahmoud has published academic research papers (Conferences and Journals) in the field of wireless communications and cryptography. Currently his research interests include communication networks, network security and cryptography, wireless communications, computer hardware and software design.

Huapeng Wu received the BS degree in electrical engineering, and the MSc degree in computer science, both from the University of Science and Technology of China (USTC), in 1987 and 1992, respectively, and the PhD degree in electrical engineering from the University of Waterloo in 1999. He was a visiting assistant professor with the Department of Electrical and Computer Engineering, Illinois Institute of Technology, for the academic year of 1999. He did postdoctoral work with the Centre for Applied Cryptographic Research at the University of Waterloo from 2000 to 2002. He is now an associate professor with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, Canada. His research interests include fast and efficient implementation of public key cryptography systems, data security, cyber security, and security applications in vehicles. Dr. Wu has authored or co-authored 20 journal papers including 15 IEEE transactions papers, and about 40 peer-reviewed conference papers. He is currently a senior member of IEEE and an associate editor for IEEE Transactions on Computers.