Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/090528

Speeding Up Finite Field Inversion for Cryptographic Applications

Walid Mahmoud [1,2,*] *and Huapeng Wu* [3]

[1] Electrical Engineering, University of Windsor, Windsor, Ontario, Canada
[2] Communication & Networks Eng., Prince Sultan University, Riyadh, KSA
[3] Electrical Engineering, University of Windsor, Windsor, Ontario, Canada

Received: 29 Jan. 2015, Revised: 30 Apr. 2015, Accepted: 1 May 2015
Published online: 1 Sep. 2015

Abstract: Finite field inversion is considered a very time-consuming operation in scalar multiplication required in elliptic curve cryptosystems. A fast inversion algorithm in binary extension fields using normal basis representation is proposed. It is based on Fermat's theorem. Compared to existing similar methods, it is shown that for a given extension degree m of the concerned field the proposed algorithm requires as few as or fewer multiplications than any other similar algorithm in the literature.

Keywords: Binary extension fields $GF(2^m)$, elliptic curve cryptography, Fermat's little theorem, field inversion, normal basis representation

1 Introduction

Elliptic curve cryptography (ECC) is one of the most popular public key cryptography technologies used today [1,2]. Finite field inversion generally regarded as probably the most time-consuming operation in computations involved in many elliptic curve cryptosystems [3,4]. Thus, the motivation behind our work is to accelerate the runtime of such cryptosystems using affine coordinates through fast inverse computation.

Finite field inversion can be solved with extended Euclidean algorithm or exponentiation with a constant exponent. The latter solution also referred to as Fermat's Little Theorem (FLT) based method. Extended Euclidean algorithm method considered not efficient in hardware implementation, since the number of computation steps varies significantly for different input and few variables must be stored throughout the computation. In FLT based method, an inverse can be computed with a series of squarings and multiplications [5,6].

Adoption of binary extension field with normal basis renders field squaring a free runtime operation, and inversion complexity (inversion cost) can be effectively measured as the number of required multiplications. In this paper, we will focus on FLT based method for solving an inverse. For a nonzero element $\alpha \in GF(2^m)$, its inverse can be given by

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

Inverse calculation without any attempt to modify (1) requires $(m-2)$ multiplications and $(m-1)$ squarings. Wang et al. implemented (1) in very large scale integration (VLSI) technology and reported their work in [7]. Itoh and Tsujii proposed an efficient inversion method with significantly reduced complexity by cleverly modifying (1). Their algorithm has inversion cost of $[\ell(m-1)+hw(m-1)-2]$ normal basis multiplications in $GF(2^m)$, where $\ell(x)$ is the binary length of x and $hw(x)$ is the Hamming weight of the binary x. Their algorithm, referred to as ITA [8], will be given a more detailed review in Section III. Chang *et al.* considered cases where $(m-1)$ can be factorized into two nontrivial divisors $(m-1 = x \times y)$ and proposed an algorithm, referred to as CEA [9], whose inversion cost is $[(\ell(x)+hw(x)-2)+(\ell(y)+hw(y)-2)]$. Takagi *et al.* proposed an improved version algorithm of CEA algorithm by allowing decomposition of $(m-1)$ into several factors plus a small remainder $(m-1 = \prod_{i=1}^{k} r_i + h)$. Their algorithm, referred to as

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

TYT [10], requires $\left[\sum_{i=1}^{k}(\ell(r_i)+hw(r_i)-2)+h\right]$ normal basis multiplications in $GF(2^m)$ to solve an inverse. Recently, Li *et al.* made incremental improvement on TYT and reduced inversion cost to $\left[\sum_{i=1}^{k}(\ell(r_i)+hw(r_i)-2)+hw(h)\right]$ normal basis multiplications in $GF(2^m)$. Their inversion algorithm referred to as LCA [11] in the rest of the paper. In summary, the problem of efficient computation of inverse with FLT method is to find a shortest addition chain (SAC) to reach the constant exponent [12].

In this paper, a new algorithm to solve inverse in $GF(2^m)$ using normal basis is presented. It is an incremental improvement on TYT and LCA algorithms. Given $(m-1)=\prod_{i=1}^{k} r_i + h$, our proposed algorithm has inversion cost of $\left[\sum_{i=1}^{k}(\ell(r_i)+hw(r_i)-2)+1\right]$. Note that it is more flexible to choose h in the proposed method, rather than restriction of h to be one in TYT algorithm and to be the least $hw(h)$ in LCA algorithm. The main idea is to decompose $(m-1)$ into several factors plus a remainder h and restrict h to belong to the SAC of any factor in $(m-1)$. Therefore, all multiplications relevant to remainder h are saved, thus, more reductions in the required multiplications for inversion can be achieved.

Given m of $GF(2^m)$ recommended either for governmental or personal use of ECC, we will show that inversion cost of our method is always as few as or fewer than other methods mentioned above. For example, when $m=216$ it requires 10 multiplications for inversion using our proposed algorithm, while it needs 11 multiplications for inversion with TYT and LCA algorithms, and it needs 12 multiplications for inversion with ITA algorithm.

The rest of the paper is organized as follows: A preliminary mathematics of finite fields is reviewed in Section 2. The relevant field inversion algorithms are previewed in Section 3. Our proposed inversion algorithm is presented in detail in Section 4. Comparison tables and the achieved results are provided in Section 5. Finally, conclusions are drawn in Section 6.

2 Mathematical Preliminaries

2.1 Finite field $GF(2^m)$ and normal basis

A field F is a commutative ring whose nonzero elements form a group under multiplication. Thus, every nonzero field element $\alpha \in F$ has a corresponding multiplicative inverse element $\alpha^{-1} \in F$. A finite field $GF(2^m)$ is a field containing 2^m elements that has characteristic of two.

A basis of $GF(2^m)$ over $GF(2)$ of the form $\{\theta^{2^0}, \theta^{2^1}, \cdots, \theta^{2^{m-2}}, \theta^{2^{m-1}}\}$, where $\theta \in GF(2^m)$ is suitable field element, is called a normal basis (NB) of $GF(2^m)$ over $GF(2)$ [13]. It is known that there always exists an NB for $GF(2^m)$ over $GF(2)$ for any value of $m \geq 1$ [1].

Any given field element $\alpha \in GF(2^m)$ can be represented with respect to the NB as follows

$$\alpha = a_0\theta^{2^0} + a_1\theta^{2^1} + \cdots + a_{m-2}\theta^{2^{m-2}} + a_{m-1}\theta^{2^{m-1}},$$

where $a_i \in \{0,1\}$ for $i=0,1,\cdots,m-1$. Using NB, α can also be given by a binary vector

$$\alpha = (a_0 a_1 \cdots a_{m-2} a_{m-1}). \tag{2}$$

For two arbitrary elements $\alpha, \beta \in GF(2^m)$, we have $(\alpha+\beta)^2 = \alpha^2 + \beta^2$, since the characteristic for $GF(2^m)$ is two and the term $2\alpha\beta = 0$. As mentioned above, nonzero elements in $GF(2^m)$ form cyclic group under multiplication with group order $|GF^{\times}(2^m)| = 2^m - 1$. Therefore, given $\alpha \in GF^{\times}(2^m)$, we have $\alpha^{2^m-1} = 1$ and $\alpha^{2^m} = \alpha$. Thus, field squaring of α w.r.t. NB can be obtained as follows:

$$\alpha^2 = \left[\sum_{i=0}^{m-1} a_i\theta^{2^i}\right]^2 = \sum_{i=0}^{m-1} a_i\theta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{((i-1))}\theta^{2^i}$$
$$= (a_{m-1}a_0a_1\cdots a_{m-3}a_{m-2}), \tag{3}$$

where the subscript $((i-1))$ is defined as $[(i-1) \bmod m]$. Comparing (3) to (2), α^2 can be obtained from cyclic shifting to the right the coefficients of α. Furthermore, the operation of α^{2^i} w.r.t. NB can be realized with i-bit right cyclic-shifts.

Based on above discussion $\alpha^{-1} = \alpha^{2^m-2}$. Knowing that

$$2^m - 2 = 2^1 + 2^2 + \cdots + 2^{m-2} + 2^{m-1},$$

then α^{-1} can be given by

$$\alpha^{-1} = \alpha^{2^m-2} = \alpha^{2^1+2^2+\cdots+2^{m-2}+2^{m-1}}$$
$$= \alpha^{2^1} \times \alpha^{2^2} \times \cdots \times \alpha^{2^{m-2}} \times \alpha^{2^{m-1}}. \tag{4}$$

It can be clearly seen from (4) that inversion requires $(m-2)$ multiplications and $(m-1)$ squarings. Given that squaring is a free runtime operation when using NB, inversion operation needs only $(m-2)$ NB multiplications in $GF(2^m)$.

2.2 Addition chains

An addition chain for a positive integer r, denoted as C_r, is a chain (*sequence*) of elements (*integers*) of length l, with the property that r (*the last chain-element*) is given by gradual addition of previous chain-elements. When r is small enough, such as the factors in $(m-1)$, the chain is referred to as the shortest addition chain (SAC). This because the chain is a priori known and has the minimal chain-length. Given r, then

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

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

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

Assume $r = 18$, then

$$C_{18} = (1,2,4,8,16,18),$$

associated with the sequence of integer-pairs

$$A_{18} = ((1,1),(2,2),(4,4),(8,,8),(16,2)),$$

following the governing rule $c_i = c_{i-1} + c_{i-1} = 2c_{i-1}$ for $i \in \{1,2,3,4\}$, except for $i = 5$, where $c_5 = c_4 + c_1$. Another chain for $r = 18$ is given by

$$C_{18} = (1,2,3,6,12,18),$$

associated with the sequence of integer-pairs

$$A_{18} = ((1,1),(2,1),(3,3),(6,,6),(12,6)),$$

following the governing rule $c_i = c_{i-1} + c_{i-1} = 2c_{i-1}$ for $i \in \{1,3,4\}$, and $c_i = c_{i-1} + c_{i-2}$ for $i \in \{2,5\}$. Note that both chains are the shortest addition chains (SACs) for $r = 18$. There are different SACs for r of same length.

Addition chains are useful in computing terms of the form (α^{2^r-1}) with fewest possible multiplications, given that $\alpha \in GF(2^m)$ and r is small positive integer. Such terms assist in inverse computation in $GF(2^m)$. When we get to introduce our decomposition method in section IV, r can be equal to either $(m-1)$ or any factor of it as long as $h \in C_r$. In that section, by means of an example we will also show how such computations are performed.

3 A Review of FLT based Inversion Algorithms in $GF(2^m)$ Using NB

With FLT based inversion algorithms given in (1), an inverse in $GF(2^m)$ can be generated in $(m-2)$ multiplication operations. Itoh and Tsujii proposed an algorithm that significantly reduces the number of required multiplications for inversion [8]. Their method can be described as follows.

Note that $2^m - 2 = 2(2^{m-1} - 1)$ and write $m-1$ as a q-bit binary number $(m_{q-1}m_{q-2}\dots m_1m_0)_2$ with the MSB $m_{q-1} = 1$, then

$$2^{m-1} - 1 = 2^{(m_{q-1}m_{q-2}\dots m_1m_0)_2} - 1. \tag{5}$$

Note that

$$2^{(m_j\dots m_0)_2} - 1 = (2^{m_j 2^j} - 1)2^{(m_{j-1}\dots m_0)_2} + 2^{(m_{j-1}\dots m_0)_2} - 1. \tag{6}$$

Applying (6) to (5) repeatedly with $j = q-1, q-2, \dots, 1$, it follows

$$2^{m-1} - 1 = (2^{m_{q-1}2^{q-1}} - 1)2^{(m_{q-2}\dots m_0)_2} + \cdots$$
$$(2^{m_{q-2}2^{q-2}} - 1)2^{(m_{q-3}\dots m_0)_2} + (2^{m_1 2^1} - 1)2^{m_0} + (2^{m_0} - 1). \tag{7}$$

Write

$$\begin{aligned}(2^{m_j 2^j} - 1) &= m_j(2^{2^{j-1}} + 1)(2^{2^{j-1}} - 1)\\ &= m_j(2^{2^{j-1}} + 1)(2^{2^{j-2}} + 1)\cdots(2^{2^0} + 1)(2^{2^0} - 1)\\ &= m_j(2^{2^{j-1}} + 1)(2^{2^{j-2}} + 1)\cdots(2^{2^0} + 1). \end{aligned} \tag{8}$$

Substitute $(2^{m_j 2^j} - 1)$, $j = q-1, q-2, \dots, 1$, in (7) with (8). Note that $(2^{m_0} - 1) = m_0$ and $m_{q-1} = 1$, thus we have

$$2^{m-1} - 1 = (\cdots((1 + 2^{2^{q-2}})2^{m_{q-2}2^{q-2}} + m_{q-2})\cdots$$
$$\cdots(1 + 2^{2^{q-3}})2^{m_{q-3}2^{q-3}} + \cdots + m_1)(1 + 2^{2^0})2^{m_0 2^0} + m_0.$$

Therefore, the inverse of $\alpha \in GF^{\times}(2^m)$ using ITA expression is given by

$$\alpha^{-1} =$$
$$\left(((\cdots(((\alpha^{1+2^{2^{q-2}}})^{2^{m_{q-2}2^{q-2}}} \times \alpha^{m_{q-2}})^{1+2^{2^{q-3}}})^{2^{m_{q-3}2^{q-3}}} \cdots \right.$$
$$\left. \cdots \times \alpha^{m_1})^{1+2^{2^0}})^{2^{m_0 2^0}} \times \alpha^{m_0} \right)^2. \tag{9}$$

The number of multiplication operations involved in (9) is the sum of the following two parts: i) the number of '$+$' signs in any exponent, and ii) the number of '$\times$' signs. The the number of '$+$' signs in (9) is $q-1$ or $\ell(m-1) - 1$. The number of '$\times$' signs depends on whether or not m_j is equal to one for $j = 0, 1, \dots, q-2$. This because $\alpha^{m_j} = 1$ if $m_j = 0$ and the sign '$\times$' immediately preceding α^{m_j} $(= 1)$ can be saved. Note that $m_{q-1} = 1$, thus part ii) is $hw(m-1) - 1$. Therefore, inversion cost in (9) is given by $[\ell(m-1) + hw(m-1) - 2]$. ITA inversion algorithm was derived based on previous discussion (**Algorithm 1**).

Feng [14] proposed an inversion algorithm with inversion cost similar to that of ITA algorithm, except that it relies on field multiplications and square-roots (left cyclic-shifts in NB) to computer inversion. The algorithm is highly regular and modular, thus, is suitable for VLSI implementation as claimed by the author.

Chang *et al.* improved ITA algorithm for cases in which $(m-1)$ can be factorized into two non-trivial divisors [9]. Their method can be described as follows. Given $m - 1 = x \times y$, we have

Algorithm 1 : ITA Inversion Algorithm in $GF(2^m)$ [8]

Input: $\alpha \in GF^{\times}(2^m)$, and $m-1 = (1m_{q-2}...m_1m_0)_2$.
Output: $\delta = \alpha^{2^m-2} = \alpha^{-1} \in GF(2^m)$
Initialization: $\delta := \alpha$;
for $i := q-2$ to 0 **do**
  $\delta := \delta \times \delta^{2^{2^i}}$;
  if $m_i = 1$ **then**
    $\delta := \alpha \times \delta^{2^{2^i}}$;
  end if
end for
$\delta := \delta^2$;
return δ

$$2^{m-1}-1 = 2^{xy}-1$$
$$= (2^x-1)((2^x)^{y-1}+(2^x)^{y-2}+\cdots+(2^x)^1+(2^x)^0).$$

Therefore, α^{-1} using CEA expression is given by

$$\alpha^{-1} = \alpha^{2(2^{m-1}-1)}$$
$$= \left[(\alpha^{2^x-1})^{(2^x)^{y-1}+(2^x)^{y-2}+\cdots+(2^x)^1+(2^x)^0}\right]^2.$$

Assume that $\delta = (\alpha^{2^x-1})$ has been computed with ITA algorithm by setting the inputs as α and $m-1 = x$. Thus it requires $[\ell(x)+hw(x)-2]$ multiplications. Let y be represented by r-bit binary number $y = (1y_{r-2}\ldots y_0)_2$. By following similar procedure to that in (9), the number of multiplication operations involved is given by $[\ell(y)+hw(y)-2]$. Therefore, inversion cost of this improved algorithm is given by

$$[(\ell(x)+hw(x)-2)+(\ell(y)+hw(y)-2)]. \quad (10)$$

Note that this improvement is not applicable when $(m-1)$ is prime.

TYT algorithm proposed by Takagi *et al.* is a further improvement on ITA and the method of Chang *et al* in two aspects [10]. First TYT allows $(m-1)$ to be a prime by factorizing $(m-1-h)$, where h restricted to small value, rather than $(m-1)$. Secondly, TYT permits more than two divisors, as long as an optimal decomposition [10] is obtained. Their method can be described as follows.

Assume $(m-1)$ can be decomposed into $(m-1) = \prod_{j=1}^{k} r_j + h$, *i.e.*, several factors plus a small remainder h. Write $2^m-2 = 2^{m-1}+2^{m-2}+\cdots+2^{m-h}+2^{m-h}-2$, then α^{-1} using TYT expression is given by

$$\alpha^{-1} = \underbrace{\alpha^{2^{m-1}} \times \alpha^{2^{m-2}} \times \cdots \times \alpha^{2^{m-h}} \times}_{h \text{ multiplications}} (\alpha^{2^{m-h-1}-1})^2. \quad (11)$$

The last term $(\alpha^{2^{m-h-1}-1}) = \alpha^{2^{r_1 \times r_2 \times \cdots r_k}-1}$ in (11) can be computed by applying Chang *et al*'s method recursively: First let $x = r_1 \times \cdots \times r_{k-1}$, $y = r_k$, and $\delta = \alpha^{2^x-1}$. Based on (9) it requires $[\ell(r_k)+hw(r_k)-2]$ multiplications. Secondly, let $x = r_1 \times \cdots \times r_{k-2}$, $y = r_{k-1}$, and $\delta = \alpha^{2^x-1}$ to compute $\alpha^{2^{r_1 \times \cdots \times r_{k-1}}-1}$, which requires $[\ell(r_{k-1})+hw(r_{k-1})-2]$ multiplications, *etc.* Finally, computation of $\alpha^{2^{r_1}-1}$ using ITA or (9) requires $[\ell(r_1)+hw(r_1)-2]$ multiplications. Therefore, inversion cost of TYT algorithm is given by

$$\left[\sum_{j=1}^{k}[\ell(r_j)+hw(r_j)-2]+h\right].$$

Note that h is always restricted to value of 1 as claimed by the authors.

LCA algorithm proposed by Li *et al.* is a further improvement on TYT algorithm in one aspect [11]. It reuses some intermediate results to save $\ell(h)$ multiplications while keeping $hw(h)$ small in $(m-1) = \prod_{j=1}^{k} r_j + h$. Compared to TYT method, Li *et al*'s method allows h to be larger than 1 as long as reuse is applicable and $hw(h)$ can be kept minimal. As a result, Li's method gives more flexibility in decomposing $(m-1)$. Instead of (11), α^{-1} using LCA expression is given by

$$\alpha^{-1} = \alpha^{\sum_{i=m-h}^{m-1} 2^i} \times (\alpha^{2^{m-h-1}-1})^2$$
$$= \alpha^{2^m-2^{m-h}} \times (\alpha^{2^{m-h-1}-1})^2$$
$$= \underbrace{(\alpha^{2^h-1})^{2^{m-h}} \times}_{hw(h) \text{ multiplications}} (\alpha^{2^{m-h-1}-1})^2. \quad (12)$$

It is shown in LCA algorithm that it takes only $hw(h)$ multiplications to compute the underbraced term in (12), provided that h and r_1 are properly chosen, and certain intermediate results from other terms can be reused in computing the term $(\alpha^{2^h-1})^{2^{m-h}}$. Therefore, inversion cost of LCA algorithm is given by

$$\left[\sum_{j=1}^{k}[\ell(r_j)+hw(r_j)-2]+hw(h)\right].$$

Note that h is always restricted to minimum $hw(h)$ as claimed by the authors.

4 Proposed Inversion Algorithm

While inversion cost of TYT algorithm depends on value of h, inversion cost of LCA algorithm depends on Hamming weight of h. In the rest of this section, we will propose a new method to compute $\alpha^{-1} \in GF(2^m)$. In this method, inversion cost is neither dependent on h nor Hamming weight of h. Thus, more flexible choices for h value are available.

Theorem 1*Given* $(m-1)=\prod_{j=1}^{k} r_j + h$ *with* h *properly chosen to belong to* C_{r_1} *(shortest addition chain of factor* r_1*),* α^{-1} *for a nonzero* $\alpha \in GF(2^m)$ *can be computed with inversion cost bounded above by*

$$\sum_{j=1}^{k} [\ell(r_j)+hw(r_j)-2]+1. \quad (13)$$

*Proof.*Given $(m-1)$ as in *Theorem 1*, then

$$\alpha^{-1}=\alpha^{2^m-2}=\left(\alpha^{2^{m-1}-1}\right)^2$$
$$=\left(\alpha^{2^{\prod_{j=1}^{k} r_j+h}-1}\right)^2=\left((\alpha^{2^{r_1\times\cdots\times r_k}-1})^{2^h}\times(\alpha^{2^h-1})\right)^2$$
$$=\left((\alpha^{2^{r_1}-1})^{e2^h}\times(\alpha^{2^h-1})\right)^2,$$

given that

$$e=\left(((2^{r_1})^{r_2-1}+\cdots+1)\cdots((2^{r_1\times\cdots\times r_{k-1}})^{r_k-1}+\cdots+1)\right). \quad (14)$$

Knowing that α^{2^i} powers are simply i-bit right cyclic-shifts, i.e., free runtime operations. Thus, inverse expression in (14) can be reduced to the following

$$\alpha^{-1}=\left[(\alpha^{2^{r_1}-1})^e\times(\alpha^{2^h-1})\right]. \quad (15)$$

Let us now forget about the computational cost of the term $(\alpha^{2^{r_1}-1})$ (we return to it before ending the proof), the computational cost required for using exponent e in (15) is given by

$$\sum_{j=2}^{k}[\ell(r_j)+hw(r_j)-2], \quad (16)$$

and its detailed steps are given as follows. Firstly, computational cost of

$$(\alpha^{2^{r_1}-1})^{(2^{r_1})^{r_2-1}+\cdots+(2^{r_1})^0}=(\alpha^{2^{r_1\times r_2}-1})$$

is $[\ell(r_2)+hw(r_2)-2]$ multiplications. Secondly, computational cost of

$$(\alpha^{2^{r_1\times r_2}-1})^{(2^{r_1\times r_2})^{r_3-1}+\cdots+(2^{r_1\times r_2})^0}=(\alpha^{2^{r_1\times r_2\times r_3}-1})$$

is $[\ell(r_3)+hw(r_3)-2]$ multiplications. Finally, computational cost of

$$(\alpha^{2^{r_1\times\cdots\times r_{k-1}}-1})^{(2^{r_1\times\cdots\times r_{k-1}})^{r_k-1}+\cdots+(2^{r_1\times\cdots\times r_{k-1}})^0}$$
$$=(\alpha^{2^{r_1\times\cdots\times r_k}-1})=(\alpha^{2^{r_1}-1})^e,$$

is $[\ell(r_k)+hw(r_k)-2]$ multiplications. Thus, computational cost relevant to all factors in $(m-1)$, except for r_1, is exactly as given in (16) above. By returning to the term $(\alpha^{2^{r_1}-1})$, its computational cost can be given as follows. Let the SAC for r_1 be 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 associated sequence of integer pairs of C_{r_1} be given by

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

using the addition rule $c_i=c_{i-1}+c_{i-1}$ for $i\in\{1,2,3\}$. Then, it follows that $(\alpha^{2^{r_1}-1})=(\alpha^{2^{c_3}-1})$ is computed as follows

$$(\alpha^{2^{c_0}-1})^{2^{c_0}}\times(\alpha^{2^{c_0}-1})=(\alpha^{2^{c_0+c_0}-1})=(\alpha^{2^{c_1}-1})$$
$$(\alpha^{2^{c_1}-1})^{2^{c_1}}\times(\alpha^{2^{c_1}-1})=(\alpha^{2^{c_1+c_1}-1})=(\alpha^{2^{c_2}-1})$$
$$(\alpha^{2^{c_2}-1})^{2^{c_2}}\times(\alpha^{2^{c_2}-1})=(\alpha^{2^{c_2+c_2}-1})=(\alpha^{2^{c_3}-1}) \quad (17)$$

Let us assign integer values for C_{r_1} elements while satisfying the addition rule of A_{r_1}, i.e., let $c_0=1, c_1=2, c_2=4, c_3=r_1=8$ such that $C_{r_1}=C_8=\{1,2,4,8\}$, then $[\ell(r_1)+hw(r_1)-2]=[\ell(8)+hw(8)-2]=3$. This result exactly equal the number of multiplications necessary to compute the term $(\alpha^{2^{r_1}-1})=(\alpha^{2^8-1})$, as evident from (17) above. In other words, 3 is equal to the length of $C_{r_1}=C_8$, which represents the number of commas separating its elements. Note that previous discussion is valid for any choice of r_1 and its associated SAC. Thus, computational cost of the term $(\alpha^{2^{r_1}-1})$ is given by

$$[\ell(r_1)+w(r_1)-2], \quad (18)$$

multiplications. Notice that if $h=c_i$ for $i\in\{0,1,2,3\}$, then $(\alpha^{2^h-1})=(\alpha^{2^{c_i}-1})$ is an intermediate result, as evident from (17) above. Thus, all computational cost of the term (α^{2^h-1}) is saved. Extra multiplication is required to combine the terms in (15) above. When added to other costs given in (16) and (18), the inversion cost of our proposed algorithm is exactly as given in *Theorem 1* above.

In the following, by means of an example we introduce our proposed algorithms. Consider $GF(2^{216})$ and assume that $m=216$ is already passed to our proposed decomposition algorithm (**Algorithm 2**). Given that r_1 is a small factor, the SAC and the sequence of integer pairs of r_1 are assumed already available and accessible. Given that $m=216$ and a nonzero field element $\alpha\in GF(2^{216})$ in which its inverse α^{-1} is required, then

$$DecomExtn(216)=(13,4,7) \text{ and } (m-1)=13\times16+7.$$

Algorithm 2 *DecomExtn*(m) Algorithm in $GF(2^m)$

Input: extension degree m
Output: $(m-1) = r_1 \times n + h$ given as (r_1, n, h)
Initial: $t := (m-1)$, $l := \sqrt{t}$, $j := 0$;
if t is odd: $S \leftarrow S_o = \{1,3,\cdots,l := \lceil l \rceil\}$. $\lfloor l \rfloor$ if l odd;
if t is even: $S \leftarrow S_e = \{2,4,\cdots,l := \lfloor l \rfloor\}$. $\lceil l \rceil$ if l odd;
for all i in the selected S **do**
  $R := t - i$;
  while ($R \neq 2^k$ **and** $2|R$ **and** $\lceil R/2 \rceil \geq i$) **do**
    $R := (\frac{R}{2})$, $j := j+1$;
  end while
  save the resulted 3-tuple $(R, j, S(i))$ in array (memory)
  $j := 0$;
  next i;
end for
find $(R, j, S(i))$ with $i \in C_R$ and largest j;
return $(r_1 := R, n := 2^j, h := S(i))$

Thus $r_1 = 13$, $n = 16$ and $h = 7$. The SAC for $r_1 = 13$, $C_{13} = \{1,2,3,6,7,13\}$. Given that α, C_{r_1}, A_{r_1}, and h are passed as inputs to our proposed *chain-inverse* algorithm (**Algorithm 3**) , the resulted output is $(\alpha^{2^{13}-1})$, which requires $[\ell(r_1) + hw(r_1) - 2] = [\ell(13) + hw(13) - 2] = 5$ $GF(2^{216})$-multiplications. In order to understand how *chain-inverse* algorithm works, calculation steps for the term $(\alpha^{2^{r_1}-1})$ are shown in (19) for $r_1 = 13$ and C_{13} as given above.

Algorithm 3 *chain-inverse* Algorithm in $GF(2^m)$

Input: $e \in GF^*(2^m)$, C_v *and* A_v precomputed,
  $\kappa := 0$ if $(m-1)$ not-decomposed, otherwise $\kappa := h$
Output: $\delta_{c_l}^2 = e^{-1} \in GF(2^m)$, $f = (\alpha^{2^\kappa - 1})$
Given: $\delta_{c_i}(e) = e^{2^{c_i}-1}$, $\delta_{c_{i_1}+c_{i_2}}(e) = [\delta c_{i_1}(e)]^{2^{c_{i_2}}} \times \delta_{c_{i_2}}(e)$
Initial: $l := length(C_v)$, $\delta_{c_0} := e$;
for $i := 1$ to l **do**
  $\delta_{c_i}(e) := [\delta c_{i_1}(e)]^{2^{c_{i_2}}} \times \delta_{c_{i_2}}(e)$;
  if $\kappa = c_i$ **then**
    $f := \delta_{c_i}(e)$;
  end if
end for
if $\kappa \neq 0$ **then**
  return δ_{c_l}, f
end if
return $\delta_{c_l}^2, f$

$$
\begin{aligned}
(\alpha)^{2^1} \times (\alpha) &= (\alpha^3) = (\alpha^{2^2-1}) \\
(\alpha^3)^{2^1} \times (\alpha) &= (\alpha^7) = (\alpha^{2^3-1}) \\
(\alpha^7)^{2^3} \times (\alpha^7) &= (\alpha^{63}) = (\alpha^{2^6-1}) \\
(\alpha^{63})^{2^1} \times (\alpha) &= (\alpha^{127}) = (\alpha^{2^7-1}) \\
(\alpha^{127})^{2^6} \times (\alpha^{63}) &= (\alpha^{8191}) = (\alpha^{2^{13}-1})
\end{aligned} \tag{19}
$$

Given that $h = 7 \in C_{13}$, then the term $(\alpha^{2^h-1}) = (\alpha^{2^7-1})$ is available when computing the term $(\alpha^{2^{13}-1})$, as evident from (19) above. Notice how the outputs in (19) at each computation step have the form (α^{2^x-1}), where x is the elements of C_{13} in their appropriate order with $(\alpha^{2^1-1}) = \alpha$.

Computations relevant to other factors in $(m-1)$ are performed using our proposed *Factors* algorithm (**Algorithm 4**). The algorithm requires $\sum_{j=2}^{k}[\ell(r_j) + hw(r_j) - 2] = [\ell(16) + hw(16) - 2] = 4$ $GF(2^{216})$-multiplications.

Algorithm 4 *Factors* Algorithm in $GF(2^m)$

Input: $\lambda \in GF(2^m), v = \prod_{j=2}^{k} r_j : r_j = (1m_{q_j-2}^{(j)} \cdots m_0^{(j)})_2$
Output: $\mu = \lambda^e = (\alpha^{2^{r_1}-1})^e$
Initial: $r := 1$;
for $j := 2$ to k **do**
  $\mu := \lambda$;
  $r := r \times r_{j-1}$;
  for $i := q_j - 2$ to 0 **do**
    $\mu := \mu \times \mu^{2^{r2^i}}$;
    if $m_i^{(j)} = 1$ **then**
      $\mu := \lambda \times \mu^{2^{r2^i}}$;
    end if
  end for
  $\lambda := \mu$;
end for
return μ

When adding the required multiplications in (19) to those required by *Factors* algorithm, the total number of required multiplications relevant to all factors in $(m-1)$ is given by $\sum_{j=1}^{k}[\ell(r_j) + hw(r_j) - 2] = 5 + 4 = 9$ $GF(2^{216})$-multiplications. Knowing that one extra multiplication is required to combine the term (α^{2^h-1}), the inverse of α is computed with $\sum_{j=1}^{k}[\ell(r_j) + hw(r_j) - 2] + 1 = 9 + 1 = 10$ $GF(2^{216})$-multiplications, which is the inversion cost using our proposed algorithm.

Inverse(α, m) is our proposed main inversion algorithm (**Algorithm 5**). It consists of all previously mentioned helper algorithms which are ready for call when appropriate and depending on the case under consideration.

In what follows, we present a set of propositions helpful to understand the way in which *Inverse*(α, m) algorithm works, and to clarify the best possible techniques to decompose $(m-1)$. Here are some useful notations. The required number of multiplications necessary to compute the term (α^{2^r-1}), $NRMs(r)$, binary length of r, $l(r)$, Hamming weight of r, $w(r)$, r is a full-weight integer, $l(r) = w(r)$, shortest addition chain of

Algorithm 5 *Inverse* Algorithm in $GF(2^m)$

Input: $\alpha \in GF^*(2^m)$, extension degree m
Output: $\delta = \alpha^{-1} \in GF(2^m)$
Initial: $(m-1)$;
Case when $(m-1)$ is not-decomposed:
  if $((m-1) = 2^k$ **or** $w(m-1) = 2)$ **then**
    return $\delta := chain\text{-}inverse(\alpha, C_{m-1}, A_{m-1}, 0)$
Case when $(m-1)$ is decomposed:
  else
    fetch (r_1, n, h) from the array (memory);
    $[\eta, \rho] := chain\text{-}inverse(\alpha, C_{r_1}, A_{r_1}, h)$;
    $\gamma := Factors(\eta, n)$;
    if $(h = 0)$ **then**
      return $\delta := \gamma$
    else
      return $\delta := ((\gamma)^{2^h} \times \rho)^2$
  end if

r, C_r, length of C_r, l_{C_r}. Note that if $r = (m-1)$, then $NRMs(m-1)$ gives the required number of multiplications necessary to compute the inverse. Also, if $r = h$, then $NRMs(h) = 1$ (using our decomposition method). Furthermore, given $DecomExtn(m) = r_1 \times n + h$, then $NRMs(DecomExtn(m)) = NRMs(r_1 \times n + h) = NRMs(r_1) + NRMs(n) + 1$.

Prop. **1** *Given $m = (2^k + 1)$ for any positive integer k, no decomposition is required for m. In calling chain-inverse algorithm, the inverse is obtained with $NRMs(2^k) = k$ multiplications.*

Prop. **2** *Given $m = (2^k + 2^j + 1)$ for any positive integers $k > j$ (i.e., $w(m-1) = 2$), no decomposition is required for m. In calling chain-inverse algorithm, the inverse is obtained with $NRMs(2^k + 2^j) = k + 1$ multiplications.*

Prop. **3** *Given $m = (p+1)$, for prime p with $w(p) = 3$, then $DecomExtn(p+1)$ is expected to produce more reductions, and the inverse is obtained with $NRMs(r_1 \times n + h)$ multiplications.*

Prop. **4** *Given $m = (p \times q + 1)$, for prime p and q with $w(p \times q) = 3$, then $DecomExtn(p \times q + 1)$ is expected to produce more reductions, and the inverse is obtained with $NRMs(r_1 \times n + h)$ multiplications.*

Prop. **5** *Given $m = (2^k \times p + 1)$ for any positive integer k and prime p. If $w(p) < 3$, no decomposition is required for m. In calling chain-inverse algorithm, the inverse is obtained with $NRMs(2^k \times p) = k + NRMs(p)$ multiplications. If $w(p) \geq 3$, let $r_1 = 2^k$ and $r_2 = DecomExtn(p+1)$, thus the inverse is obtained with $NRMs(r_1 \times r_2) = k + NRMs(r_2)$ multiplications.*

Prop. **6** *For the set of extension degrees m those decomposed as $r_1 \times n + h$, if $h \in C_{r_1}$ is a full-weight remainder, $w(r_1) = 2$ and $n = 2^k$ for any positive integer k. When increasing k while fixing r_1 and h values, those m values are expected to have the minimum $NRMs(r_1 \times n + h)$ in inverse computation relative to others.*

Prop. **7** *For the set of m values those decomposed as $r_1 \times n + h$ with $h = 1$ (this h always $\in C_{r_1}$), those m values in inverse computation have $NRMs(r_1 \times n + h)$ same as that required by using TYT algorithm. With $h = 2^k$ for any positive integer k (this h often $\in C_{r_1}$), those m values in inverse computation have $NRMs(r_1 \times n + h)$ same as that required by using LCA algorithm. Thus, such m values are also associated with lowest possible inversion cost using our inversion algorithm.*

Prop. **8** *For the set of m values those decomposed as $r_1 \times n + h$ with $h = r_1$ (this h always $\in C_{r_1}$), those m values can be factorized into prime factors. In such a case, $DecomExtn(m)$ algorithm may or may not produce the minimum $NRMs(r_1 \times n + h)$ in inverse calculation compared to prime factors decomposition. Select first decomposition if associated with lower inversion cost, otherwise, select the latter using the prime factors r_1 and r_2, since $r_1 \times n + r_1 = r_1 \times (n+1) = r_1 \times r_2$. Factorize r_2 if possible, to achieve more reductions.*

The following propositions are helpful in finding the shortest addition chains (SACs) for an integer, which may be applicable in some cases. In searching for the SAC for a positive integer x (i.e., C_x), then:

Prop. **1** *If x is large, there are many possible SACs for x. Thus, the set of values taken by h is maximized given that $h \in C_x$.*

Prop. **2** *If $x = r_1 \times r_2 \times \cdots \times r_k$ (multiplication of several factors), given that h is equal to any factor r_i for $1 \leq i \leq k$, then h must belong to the SAC of x, i.e., $h \in C_x$.*

Prop. **3** *If $x = r_1 \times r_2 \times \cdots \times r_k$ (multiplication of several factors), given that $r_1 > r_i$ for $2 \leq i \leq k$ and $x \neq 2^k$ for any positive integer k, there must be a SAC of x of length $l_{C_x} = l_{C_{r_1}} + \lceil \log_2 r_i \rceil$ for $2 \leq i \leq k$.*

Prop. **4** *If $x = 2^k$ for any positive integer k, then x has only one SAC of length $l_{C_x} = k$, which is the Power of Two SAC, i.e., $(1, 2, 4, \cdots, 2^k)$. Notice that the elements of the General SAC can have any value, not only power of two values.*

Prop. **5** *If $x = 2^k + 2^j$ for any positive integers $k > j$ (i.e., $w(x) = 2$), then x has at least two SACs of length $l_{C_x} = k + 1$, where the Power of Two SAC is a possible choice.*

Prop. **6** *If $x = k + n$, given that k is the largest power of two 2^i value in x for $i = 1, 2, 3, \cdots$, if $w(n) < 3$, then C_x is obtained by using either the General or the Power of Two SAC. Otherwise, if $w(n) \geq 3$, then the General SAC is expected to produce the shortest addition chain C_x of x.*

5 Analysis and Results

Here we consider a selected set of m values taken from the range $(100 \leq m < 571)$ to represent binary extension fields under consideration. Some of the values are recommended for use in ECC by NIST and SECG. The rest achieves better results in using our algorithm relative to other algorithms, which probably of interest to other cryptographic or code-theoretic applications. Given that such values are applied as input to our decomposition algorithm, their associated decompositions (AD) and the corresponding inversion costs (IC) using our proposed inversion algorithm are listed in Table 1. For comparison purposes, the AD and IC of other inversion algorithms are also listed in the table.

Table 1: Proposed *vs* Other Inversion Algorithms $[GF(2^m): 100 \leq m < 571]$

$GF(2^m)$		**ITA** [8]	**TYT** [10]		**LCA** [11]		**Proposed** [Algo. 5]	
m	$(m-1)$	IC	AD	IC	AD	IC	AD	IC
100	99	9	11×9	9	11×9	9	3×32+3	8
108	107	10	2(13×4+1)+1	10	11×9+8	10	13×8+3	9
116	115	10	23×5	10	23×5	10	7×16+3	9
150	149	10	37×4+1	10	37×4+1	10	9×16+5	9
155	154	10	17×9+1	10	17×9+1	10	18×8+10	9
163	162	9	18×9	9	18×9	9	81×2	9
164	163	10	18×9+1	10	18×9+1	10	5×32+3	9
168	167	11	83×2+1	11	41×4+3	11	10×16+7	10
174	173	11	43×4+1	11	43×4+1	11	21×8+5	10
180	179	11	2(11×8+1)+1	11	11×16+3	11	11×16+3	10
184	183	12	14×13+1	11	14×13+1	11	11×16+7	10
185	184	10	23×8	10	23×8	10	23×8	9
208	207	12	23×9	11	23×9	11	24×8+15	10
215	214	11	107×2	11	53×4+2	11	13×16+6	10
216	215	12	43×5	11	43×5	11	13×16+7	10
228	227	11	113×2+1	11	25×9+2	11	7×32+3	10
231	230	11	23×10	11	23×10	11	7×32+6	10
233	232	10	29×8	10	29×8	10	29×8	10
239	238	12	17×14	10	17×14	10	17×14	10
280	279	12	31×9	12	31×9	12	17×16+7	11
283	282	11	141×2	11	20×14+2	11	17×16+10	11
294	293	11	73×4+1	11	73×4+1	11	9×32+5	10
299	298	11	2(37×4+1)	11	37×8+2	11	18×16+10	10
312	311	13	31×10+1	13	18×17+5	12	19×16+7	11
320	319	14	29×11	12	29×11	12	19×16+15	11
324	323	11	19×17	11	19×17	11	5×64+3	10
350	349	13	29×12+1	12	29×12+1	12	21×16+13	11
360	359	13	179×2+1	13	97×3+68	12	11×32+7	11
392	391	12	23×17	12	23×17	12	12×32+7	11
404	403	12	67×6+1	12	67×6+1	12	25×16+3	11
409	408	11	24×17	10	24×17	10	51×8	10
424	423	13	47×9	13	47×9	13	13×32+7	11
436	435	13	29×5×3	12	29×5×3	12	27×16+3	11
448	447	15	149×3	12	149×3	12	27×16+15	11
571	570	13	19×6×5	12	19×6×5	12	35×16+10	12

From Table 1, it is apparent that our proposed inversion algorithm has as few as or fewer inversion cost relative to other similar inversion algorithms. In some binary extension fields $GF(2^m)$, the reductions in inversion cost are up to 4 $GF(2^m)$-multiplications. Although the set of listed degrees of m is not comprehensive, the results shown in Table 1 reflect the applicability of our proposed decomposition method in accelerating field inversion in $GF(2^m)$.

Another set of degrees m of $GF(2^m)$, selected from [11] is shown in Table 2. Such a set is associated with the lowest possible inversion cost using decomposition method of LCA algorithm. Through this comparison, we aim to show the effectiveness of our decomposition method relative to the one of LCA algorithm. Therefore, the AD and IC using our algorithm are also listed in the table.

Table 2: Proposed *vs* LCA Algorithm

$GF(2^m)$		**LCA** [11]		**Proposed** [Algo. 5]	
m	$(m-1)$	AD	IC	AD	IC
123	122	40×3+2	9	14×8+10	9
187	186	34×5+16	10	11×16+10	10
189	188	36×5+8	10	22×8+12	10
238	237	68×3+33	11	14×16+13	11
384	383	25×5×3+8	12	23×16+15	12
428	427	25×17+2	12	13×32+11	12

From Table 2, it is apparent that inversion cost of both algorithms is identical. Therefore, our proposed inversion algorithm, namely $Inverse(\alpha, m)$, can be a substitute for LCA algorithm to calculate inversion in such binary extension fields.

Another set of degrees m of $GF(2^m)$, selected from [10] is shown in Table 3. Such a set is associated with lowest possible inversion cost using decomposition method of TYT algorithm. Through this comparison, we aim to show the effectiveness of our decomposition method relative to the one of TYT algorithm. Therefore, the AD and IC using our algorithm are also listed in the table.

From Table 3, it is apparent that inversion cost of both algorithms is the same in some $GF(2^m)$, except for the last three entries in the table. In these entries our algorithm achieves lower inversion costs in comparison with TYT inversion algorithm. Therefore, our proposed inversion algorithm, namely $Inverse(\alpha, m)$, can be a substitute for TYT algorithm to calculate inversion in such binary extension fields.

In the following, we show a table which includes some extension degrees m used in ECC when defined over Silverman fields [16]. We compare the AD and IC of inversion algorithms presented in this paper over such type of extension fields, as shown in Table 4.

Table 3: Proposed *vs* TYT Algorithm

$GF(2^m)$		**TYT** [10]		**Proposed** [Algo. 5]	
m	$(m-1)$	AD	IC	AD	IC
128	127	18×7+1	10	15×8+7	10
192	191	38×5+1	11	23×8+7	11
256	255	17×5×3	10	17×15	10
320	319	29×11	12	19×16+15	11
384	383	2(38×5+1)+1	13	23×16+15	12
416	415	83×5	12	25×16+15	11

Table 4: Proposed *vs* Other Inversion Algorithms [Subset of Silverman Fields]

$GF(2^m)$		**ITA** [8]	**TYT** [10]		**LCA** [11]		**Proposed** [Algo. 5]	
m	$(m-1)$	IC	AD	IC	AD	IC	AD	IC
106	105	9	8×13+1	9	8×13+1	9	12×8+9	9
178	177	10	11×16+1	10	11×16+1	10	11×16+1	10
226	225	10	14×16+1	10	14×16+1	10	7×32+1	10
964	963	14	37×26+1	14	32×6×5+3	13	15×64+3	12
1018	1017	16	8(18×7+1)+1	14	36×7×4+9	14	31×32+25	13
1186	1185	13	32×37+1	13	32×37+1	13	37×32+1	13

From Table 4, it is apparent that inversion cost of our algorithm is as few as or fewer than in other algorithms. Therefore, our inversion algorithm can be used to compute inversion in Silverman fields, thus improving the runtime of cryptographic applications defined over such fields.

In the following, we provide an approximate comparison on memory storage requirements. Since in practical applications m is frequently selected as a power of 2, which is a suitable choice w.r.t memories, assume $m = 128$. Note that decomposition of $(m-1)$ has no effect on memory requirements for same algorithm and extension degree under consideration.

In using our inversion algorithm, the required memory is given by $\ell(r_1) + hw(r_1)$. However, in using LCA algorithm it is given by $q + 4$, where $q = max\ \{q_1, q_2, \cdots, q_k\}$, is the binary length of largest factor in $(m-1)$ when decomposed into k factors. Using TYT and ITA algorithms, the required memory is given by a constant value. Although it slightly higher than in other algorithms, the required memory in our proposed algorithm is a function of r_1. Knowing that r_1 is small integer, the obtained results still promising and conforms with space-time tradeoffs.

In general, the achieved results reflect the applicability of our proposed method for accelerating field inversion in $GF(2^m)$. Binary extension fields recommended for use in ECC are also associated with minimal inversion cost using our algorithm. Therefore, when our algorithm is employed to compute inversion in scalar multiplication algorithm, the runtime of such algorithm becomes faster. Scalar multiplication algorithm is the core of most modern ECC-based cryptographic applications, where it dominates their execution time. Such applications are: elliptic curve digital signature algorithm (ECDSA), elliptic curve Diffie-Hellman (ECDH) key-agreement algorithm, elliptic curve ElGamal (EC-ElGamal) encryption algorithm, etc.

6 Closing Remarks

In this paper, we have proposed a fast field inversion algorithm in binary extension fields $GF(2^m)$ using normal basis representation. It is based on Fermat's approach for inversion. By appropriately decomposing m of the concerned $GF(2^m)$ into several factors and a remainder h, with h belongs to the short addition chain of any of such factors, the inversion cost of our algorithm is as few as or fewer than in other similar inversion algorithms. The suitability of our algorithm for use in elliptic curve cryptography, in addition to its reliance on a set of factors which definitely have shortest addition chains, renders our method more attractive in hardware implementation and for future consideration with other finite extension fields.

Acknowledgment

The authors would like to warmly thank the anonymous referees for their help in reviewing this manuscript.

References

[1] Standard specifications for public-key cryptography, "http://grouper.ieee.org/groups/1363/D1", draft-version number 1. IEEE standards documents, November 2009.

[2] D. Hankerson, J. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography*, New York, USA: Springer-Verlag Inc., 2004.

[3] S. Fenn, M. Benaissa and D. Taylor, "Fast Normal Basis Inversion in $GF(2^m)$," *IET Electronic Letters*, vol. 32, issue 17, pp. 1566-1567, Aug. 2002.

[4] Q. Deng, X. Bai, L. Guo and Y. Wang, "A fast hardware implementation of multiplicative inversion in $GF(2^m)$," Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics, pp. 472-475, Jan. 2009.

[5] R. Lidl and H. Niederreiter, *Finite Fields*, 2nd Ed., Cambridge, UK: Cambridge University Press, 1997.

[6] S. Fenn, M. Benaissa and D. Taylor, "Finite field inversion over the dual basis," *IEEE Transactions on VLSI Systems*, vol. 4, issue 1, pp. 134-137, Jan. 1996.

[7] C. Wang, T. Truong, H. Shao, L. Deutsch, J. Omura and I. Reed, "VLSI Architecture for Computing Multiplications and Inverses in $GF(2^m)$," *IEEE Transactions on Computers*, vol. 34, issue 8, pp. 709-716, Aug. 1985.

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

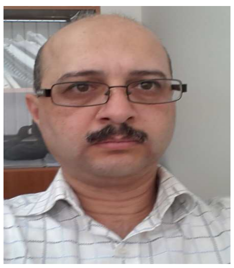
[9] T. Chang, E. Lu, Y. Lee, Y. Leu and H. Shyu, "Two Algorithms for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Basis," accepted by *Information Processing Letters Journal*.

[10] 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, May 2001.

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

[12] N. Cruz-Cortes, F. Rodriuez-Henriquez and C. Coello, "An Artificial Immune System Heuristic for Generating Short Addition Chains," *IEEE Transactions on Evolutionary Computation*, vol. 12, issue 1, pp. 1-24, Jan. 2008.

[13] D. Pei, C. Wang and J. Omura, "Normal Basis of Finite Field $GF(2^m)$," *IEEE Transactions on Information Theory*, vol. 32, issue 2, pp. 285-287, Feb. 1986.
[14] G. Feng, "A VLSI Architecture for Fast Inversion in $GF(2^m)$," *IEEE Transactions on Computers*, vol. 38, issue. 10, pp. 1383-1386, Oct. 1989.
[15] F. Rodrıguez, N. Saqib and N. Cruz, "A fast implementation of multiplicative inversion over $GF(2^m)$," *International Symposium on Info. Technology*, Las Vegas, Nevada, U.S.A., vol. 1, pp. 574-579, Apr. 2005.
[16] M. Ciet, J. Quisquater and F. Sica, "A Secure Family of Composite Finite Fields Suitable for Fast Implementation of Elliptic Curve Cryptography," *Progress in Cryptology INDOCRYPT*, vol. LNCS 2247, pp. 108-116, 2001.



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.