有限体の上の開平演算

王鳳[†]　野上　保之[†]　森川　良孝[†]

† 岡山大学通信ネットワーク工学科 〒700-8530 岡山県岡山市津島中 3-1-1
E-mail: †{wangfeng,nogami,morikawa}@trans.cne.okayama-u.ac.jp

あらまし　拡大体における平方根計算は, その計算に先立って平方剰余 (QR) テストが必要である。今までに知られている平方根計算アルゴリズムはスマート法である。しかし、平方根計算と QR テストに多くの重複計算が存在していた。本論文は新たに QR テストと平方根計算アルゴリズムを提案して、すべての重複計算を取り除いて効率を上げること目的にしている。平方根計算アルゴリズムの計算途上で必要となるすべてのデータはすでに QR テストで計算したものを用いる手法を説明して、計算時間と計算量が大幅に減少することを示す。$p = 2^{16} + 1$ と $p = 2^{16} + 3$ の場合、Pentium4(2.6GHz) 上で C++言語を用いて, $GF(p)$ と $GF(p^2)$ における、スマート法と提案法のシミュレーションを行った。$p = 2^{16} + 1$ の場合、提案法は平均としてスマート法より $GF(p)$ で２倍速く、$GF(p^2)$ で１０倍速い。また $p = 2^{16} + 3$ の場合、提案法は平均としてスマート法より $GF(p)$ で２０倍速く、$GF(p^2)$ で６倍速い。

キーワード　開平演算, 有限体

A Fast Square Root Computation in Some Finite Fields

Feng WANG[†], Yasuyuki NOGAMI[†], and Yoshitaka MORIKAWA[†]

† Communication Network Engineering, Okayama University,
3-1-1, Tsushimanaka, Okayama 700-8530, Japan
E-mail: †{wangfeng,nogami,morikawa}@trans.cne.okayama-u.ac.jp

Abstract　It is well known that quadratic residue (QR) test should be implemented in advance of a square root (SQRT) computation. Smart algorithm, the previously known fastest algorithm for SQRT computation, only got the idea how to compute SQRT through QR test. However there is a lot of computation overlap in QR test and Smart algorithm. The essence of our proposition is thus to present a new QR test and SQRT algorithm to avoid all the overlapping computations. In this paper the authors devised a SQRT algorithm for which most of the data required have been computed in QR test. This yields many reductions in the computational time and amount. In $GF(p)$ and $GF(p^2)$, we implemented Smart algorithm and the proposed algorithm in C++ language on Pentium4 (2.6GHz), where $p = 2^{16} + 1$ ($4 \mid p - 1$) and $p = 2^{16} + 3$ ($4 \nmid p - 1$). The computer simulations showed that for $p = 2^{16} + 1$ the proposed algorithm on average accelerates the SQRT computation 2 times and 10 times faster than Smart algorithm over $GF(p)$ and $GF(p^2)$, respectively and for $p = 2^{16} + 3$ the proposed algorithm on average accelerates the SQRT computation 20 times and 6 times faster than Smart algorithm over $GF(p)$ and $GF(p^2)$, respectively.

Key words　Square root, Finite fields

1. Introduction

The importance for keeping communication secure is increasing due to the prevalence of the Internet and other forms of electronic communication. A hybrid cryptosystem is well known as a technology to provide a secure environment where communication can be conducted without fear. Specifically, it uses both secret-key and public-key cryptosystems; secret-key cryptosystems are used to encrypt information, and public-key cryptosystems are used to distribute secret keys of the secret-key cryptosystems. As public-key cryptosystems, both Rivest-Shamir-Adleman (RSA) cryptosystem and elliptic curve cryptosystem (ECC) can provide secure communication. However, ECC offers equivalent secu-

rity with smaller key sizes. For example, 160-bit ECC guarantees as secure as 1024-bit RSA cryptosystem, and it follows that ECC is more suitable for small devices such as smart cards and cellular telephones. Therefore ECC has tremendous potential to keep communication secure and much attention [1]- [3] is attracted to the realization of ECC.

In realizing ECC, not only the implementation of the fundamental operations such as multiplications and inversions, but also that of the square root (SQRT) computation must be studied. This is a big motivation to develop SQRT algorithm over extension fields $GF(p^m)$.

ECC provides the user a great of flexibility in the choice of system parameters. In this paper, we let p be a prime number greater than 3 and let m have the form of $m = 2^c$, where c is an nonnegative integer. However, ECC over $GF(p^{2^c})$ can provide secure communications only when $c = 0, 1, 2$ [4]. In fact, its security is weak when $c = 2$ [4]. So, we mainly consider the two cases of $c = 0, 1$ in this paper.

It is well known that if an arbitrary nonzero element x in $GF(p^m)$ is not a quadratic residue, i.e. the element has not its SQRT in the same field, then it is nonsensical to compute its SQRT. In advance of SQRT computation, we should thus identify whether x is a quadratic residue or not, which is called quadratic residue test. Therefore, if an algorithm for SQRT computation requires a lot of data computed in the quadratic residue test, then it will provide a means for efficient SQRT computation.

Smart algorithm [5], the previously known fastest algorithm for SQRT computation in $GF(p^m)$, had the idea how to compute SQRT through quadratic residue test. However there is a lot of computation overlap in QR test and Smart algorithm, which makes square root computation inefficiency. The essence of our proposition is thus to present a new quadratic residue test and SQRT algorithm to avoid all the overlapping computations. Based on the main idea of Smart algorithm, the authors devised an algorithm, named moving window-sign testing (MW-ST) algorithm, for which most of the data required have been computed in quadratic residue test. This yields many reductions in the computational time and amount. In this paper, MW-ST algorithm for SQRT computation in $GF(p^m)$ has been proved to be much faster than Smart algorithm.

The organization of the paper is as follows. In Section 2., we first present how to implement quadratic residue test over $GF(p^m)$, and then estimate its computational complexity. In Section 3., we first review Smart algorithm over $GF(p^m)$, and then estimate its computational complexity. In Section 4., the MW-ST algorithm are first proposed in $GF(p)$ and $GF(p^2)$ respectively, and then estimate its computational complexity in $GF(p)$ and $GF(p^2)$ respectively. In Section 5., the two algorithms over $GF(p)$ and $GF(p^2)$ were implemented in C++ language running on Pentium4 (2.6GHz), where $p = 2^{16} + 1$ $(4 \mid p - 1)$ and $p = 2^{16} + 3$ $(4 \nmid p - 1)$. The computer simulations showed that for $p = 2^{16} + 1$ the proposed algorithm on average accelerates the SQRT computation 2 times and 10 times faster than Smart algorithm over $GF(p)$ and $GF(p^2)$, respectively and for $p = 2^{16} + 3$ the proposed algorithm on average accelerates the SQRT computation 20 times and 6 times faster than Smart algorithm over $GF(p)$ and $GF(p^2)$, respectively.

Throughout the paper, A_m and M_m denote additions and multiplications in extension fields $GF(p^m)$, respectively. $\#A_m$ and $\#M_m$ denote the numbers of these operations respectively and $\#S$ denotes the number of shift computation.

2. Quadratic Residue Test

In this section, we first present how to implement quadratic residue test, and then estimate its computational complexity.

2.1 How to Implement Quadratic Residue Test

We usually use the Euler's criterion to identify whether or not a nonzero element x is a quadratic residue:

$$C(x) = x^{(p^m-1)/2} = \begin{cases} 1 & \text{QR} \\ -1 & \text{QNR} \end{cases}, \tag{1}$$

where *quadratic residue* and *quadratic non-residue* are abbreviated to QR and QNR, respectively. Conventionally, we directly compute $(p^m - 1)/2$-th power of x to implement QR test as shown in Fig.1.

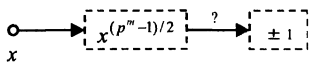

Fig. 1 Conventional quadratic residue test for $x \in GF(p^m)$

Since $(p^m - 1)/2$ can be factorized as

$$(p^m - 1)/2 = 2^T s, \ s \text{ is odd}, \tag{2}$$

it follows that

$$C(x) = x^{(p^m-1)/2} = (x^s)^{2^T} = x_0^{2^T} \text{ for } x_0 = x^s. \tag{3}$$

Based on (3), a more efficient QR test is proposed to first compute x_0, and then to repeatedly compute the square of x_0 T times. In fact, it is not necessary for some input elements to compute the power of $x^{(p^m-1)/2}$. If $x_0 = 1$, then we can assert the input element x is a QR. If not, we can repeatedly compute the square of x_0 until the product becomes -1. As shown in Fig.2, when $t < T$, x is a QR; when $t = T$, x is a QNR. Note that when $x_0 = -1$ we do not need to compute the square of x_0, it implies $t = 0$. For the convenience of the succeeding SQRT computation, we first

compute $x^{(s-1)/2}$, and then multiply $x^{(s-1)/2}$ by x to get $x^{(s+1)/2}$, finally multiply $x^{(s+1)/2}$ and $x^{(s-1)/2}$ together to attain x_0.

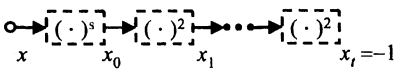

Fig. 2 Proposed quadratic residue test for $x \in GF(p^m)$

2.2 Computational Complexity of QR Test

As described in Section 2. 1, we first compute $x^{(s-1)/2}$, and then multiply $x^{(s-1)/2}$ by x to get $x^{(s+1)/2}$, finally multiply $x^{(s+1)/2}$ and $x^{(s-1)/2}$ together to attain x_0. To compute $x^{(s-1)/2}$ by using binary method, we require $\lfloor \log_2(\frac{s-1}{2}) \rfloor + w\left(\frac{s-1}{2}\right) - 1$ multiplications in $GF(p^m)$, where $\lfloor \cdot \rfloor$ and $w(\cdot)$ denote the maximum integer less than $\cdot$ and the Hamming weight of $\cdot$, respectively. Therefore, we can easily know that x_0 requires the following multiplications:

$$\#M_m = LW(\frac{s-1}{2}) + 1, \tag{4}$$

where the expression $\lfloor \log_2(\cdot) \rfloor + w(\cdot)$ is denoted as $LW(\cdot)$ for the convenience. Note that when $s=1$, we do not require any computation to get x_0, in other words $\#M_m = 0$.

If $x_0 = 1$, then we can assert that the input element x is a QR. If not, we need t multiplications to get $x_t = -1$ as shown in Fig.2. Note that the value of t depends on the input element x that can not be known in advance. Since $t \leqslant T$, we can know that QR test at most requires the following multiplications:

$$\#M_m = LW(\frac{s-1}{2}) + 1 + T, \tag{5}$$

where $\#M_m = T$ when $s = 1$.

3. Smart Algorithm over $GF(p^m)$

Smart algorithm is well known as a conventional method to compute square root. In this section, we first review Smart algorithm over $GF(p^m)$ and then estimate its complexity. Note that Smart algorithm over $GF(p^m)$ is considered under the following condition:

$$(p^m - 1)/2 = 2^T s, \ s \text{ is odd.} \tag{6}$$

3.1 Smart algorithm

Smart algorithm

Input: A nonzero QR $x \in GF(p^m)$.

Output: A square root $\sqrt{x} \in GF(p^m)$.

Preparation:

(1) Factorize $(p^m - 1)/2$ as

$$(p^m - 1)/2 = 2^T s, \ s \text{ is odd.} \tag{7}$$

(2) Find an appropriate QNR $\theta \in GF(p^m)$ and compute $a = \theta^s$.

Procedure:

- **Step1:** Compute $b = x^{(s-1)/2}$ and set $t_0 = 0, k = 0$.
- **Step2:** Iteratively compute the following expressions until k increases by 0 to $T-1$:

$$t_{k+1} = t_k + 2^k c_k, \tag{8a}$$

$$\text{where } c_k = \begin{cases} 0 & \text{if } \left((a^{t_k} b)^2 \cdot x\right)^{2^{T-1-k}} = 1 \\ 1 & \text{if } \left((a^{t_k} b)^2 \cdot x\right)^{2^{T-1-k}} = -1 \end{cases}. \tag{8b}$$

- **Step3:** Output the square root $\sqrt{x} = a^{t_T} b x$.

Since the finite fields $GF(p^m)$ are given before the SQRT computation, a, s and T can be prepared in advance. Note that when $T = 0$, **Step2** is skipped and Smart algorithm still remains valid.

3.2 Computational Complexity of Smart Algorithm

SQRT computation is performed in the given extension fields $GF(p^m)$, so computational complexity of the preparation for Smart algorithm becomes unimportant. In what follows, we only consider computational complexity of the main procedure. Note that Smart algorithm adopt the conventional QR test.

In **Step1**, we can use binary method to compute the $(s-1)/2$-th power of x, which requires the following multiplications:

$$\#M_m = LW(\frac{s-1}{2}) - 1, \tag{9}$$

where $\#M_m = 0$ when $s = 1$.

In **Step2**, $a^{t_k} b$ shown in (8b) have the form

$$b, a^{c_0} b, a^{c_0} a^{2c_1} b, a^{c_0} a^{2c_1} a^{2^2 c_2} b, \cdots, \left(\prod_{i=0}^{T-1} a^{2^i c_i}\right) b \tag{10}$$

for each $k = 0, 1, \cdots, T$. Note that we only need to increase k up to $T-1$ in **Step2**. The reason that k is increased by 0 to T in (10) is that the last expression in (10) i.e. $a^{t_T} b$ is required for **Step3**. The number of the multiplications required for (10) depends on the value of c_k for $k = 0, 1, 2, \cdots, T-1$. When every c_k is equal to 0, the number of the multiplications required for (10) becomes the minimum. When every c_k is equal to 1, the number of the multiplications required for (10) becomes the maximum and $a^{t_k} b$ have the form

$$b, \ ab, \ aa^2 b, \ aa^2 a^4 b, \ \cdots, \ \left(\prod_{i=0}^{T-1} a^{2^i}\right) b \tag{11}$$

for each $k = 0, 1, \cdots, T$. Because the value of c_k depends on the input element x that can not be known in advance, we

only consider the maximum for the convenience. In (11), ab (when $k = 1$) requires one multiplication in $GF(p^m)$. aa^2b (when $k = 2$) requires 2 multiplications in $GF(p^m)$ because aa^2b can be computed by multiplying ab and a^2 together, where ab has already been computed when $k = 1$. aa^2a^4b (when $k = 3$) requires 3 multiplications in $GF(p^m)$ because aa^2a^4b can be computed by multiplying aa^2b and a^4 together, where aa^2b has already been computed when $k = 2$. In this way, we can know that $\left(\prod_{i=0}^{T-1} a^{2^i}\right)b$ (when $k = T$) requires T multiplications in $GF(p^m)$. It follows that (11) requires

$$\#M_m = \frac{T^2+T}{2}. \tag{12}$$

Next, for each $k = 0, 1, \cdots, T-1$ we compute the square of $a^{t_k}b$, and then multiply by x as shown in (8b). This process requires

$$\#M_m = 2T. \tag{13}$$

In addition to the above, we require the exponentiation to 2^{T-1-k}-th power for each k as shown in (8b) and finally multiply $a^{t_T}b$ by x in **Step3**, then this process requires

$$\#M_m = \sum_{i=1}^{T-1} i + 1 = \frac{T(T-1)}{2} + 1. \tag{14}$$

In total, Smart algorithm needs the following multiplications:

$$\#M_m = LW(\frac{s-1}{2}) + T^2 + 2T, \tag{15}$$

where $\#M_m = T^2 + T + 1$ when $s = 1$.

4. MW-ST Algorithm over $GF(p^m)$

Based on the proposed QR test and the main idea of Smart algorithm, an efficient SQRT algorithm, named MW-ST algorithm, was devised. In this section, we first present the MW-ST algorithm over $GF(p^m)$ ($m = 1$ or 2), and then estimate its computational complexity in $GF(p)$ and $GF(p^2)$ respectively.

4.1 MW-ST Algorithm over the Finite Fields $GF(p^m)$

When $x_0 = 1$ i.e. $x^s = 1$, multiplying its both sides by x and taking SQRT, we have $\sqrt{x} = x^{(s+1)/2}$. When $x_0 \neq 1$, QR test will be implemented as shown in Fig.3. If $t < T$, as shown in the upper series of Fig.3, then x is a QR ; if not, as shown in the lower series of Fig.3, then x, denoted by c, is a QNR.

It is well known that the SQRT of 1 must be 1 or -1 in any finite field; however, we are not sure if the SQRT of -1 exists in $GF(p^m)$. Even if the SQRT of -1 exists in $GF(p^m)$, we also can not know what it is. Therefore, it is unfeasible to extract the SQRT of $x_t = -1$. From Fig.3, we can easily know that $c_T x_t = 1$ and the SQRT of $c_T x_t$ is equal to $c_{T-1}x_{t-1}$ defined as

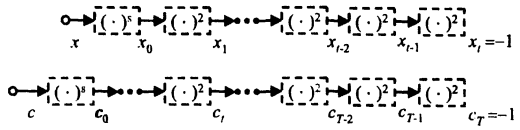

Fig. 3 A series of QR and QNR in $GF(p^m)$

$$\sigma(1) = c_{T-1}x_{t-1}. \tag{16}$$

Since the SQRT of 1 must be 1 or -1, we can assert that $\sigma(1) = \pm 1$. If $\sigma(1) = 1$, then we take the SQRT of $\sigma(1)$ to get $c_{T-2}x_{t-2}$ which is defined as $\sigma(2) = c_{T-2}x_{t-2}$. If not, then we first multiply $\sigma(1)$ by c_T to get $c_T c_{T-1}x_{t-1} = 1$, and then take its SQRT that is defined as $\sigma(2) = c_{T-1}c_{T-2}x_{t-2}$. It is easy to know that $\sigma(2) = \pm 1$.

As what described above, we first consider the windows of c_T and x_t, then take the SQRT of $c_T x_t$ to consider the windows of c_{T-1} and x_{t-1}, and then test the sign of $c_{T-1}x_{t-1}$. If the sign is $+$, we take its SQRT to get $c_{T-2}x_{t-2} = \pm 1$, which implies we will move to the windows of c_{T-2} and x_{t-2}. If the sign is $-$, we take the SQRT of $c_T c_{T-1}x_{t-1}$ to get $c_{T-1}c_{T-2}x_{t-2}$, which also implies we will move to the windows of c_{T-2} and x_{t-2}. Therefore, the proposed algorithm is named moving window-sign testing algorithm.

In this way, we have

$$\sigma(k) = c_{T-1}^{i_{k-1}} c_{T-2}^{i_{k-2}} \cdots c_{T-k-1}^{i_1} c_{T-k} x_{t-k} = \pm 1, \tag{17}$$

where for $1 \leqslant k \leqslant t$, if $\sigma(k) = 1$ then $i_k = 0$ and if $\sigma(k) = -1$ then $i_k = 1$. When $k = t$, we have

$$\sigma(t) = c_{T-1}^{i_{t-1}} c_{T-2}^{i_{t-2}} \cdots c_{T-t-1}^{i_1} c_{T-t} x_0 = \pm 1. \tag{18}$$

Since $x_0 = x^s$ and $c_T = -1$, (18) can be developed as

$$c_T^{i_t} c_{T-1}^{i_{t-1}} \cdots c_{T-t-1}^{i_1} c_{T-t} x^s = 1, \tag{19}$$

where for $1 \leqslant k \leqslant t$, i_k has the same definition as in (17). Multiplying the both sides of (19) by x and taking SQRT, we can easily get

$$\sqrt{x} = c_{T-1}^{i_t} c_{T-2}^{i_{t-1}} \cdots c_{T-t-2}^{i_1} c_{T-t-1} x^{(s+1)/2}. \tag{20}$$

Moving window-sign testing algorithm over $GF(p^m)$ ($m = 1$ or 2)

Input: A nonzero QR $x \in GF(p^m)$.

Output: A square root $\sqrt{x} \in GF(p^m)$.

Preparation:

(1) Factorize $(p^m-1)/2$ as

$$(p^m-1)/2 = 2^T s, \ s \text{ is odd.} \tag{21}$$

(2) Find an appropriate QNR $c \in GF(p^m)$ and compute $c_0, c_1, \cdots, c_T$ as shown in Fig.3.

Main Procedure:

(1) Check whether x_0 is equal to 1 or not. If $x_0 = 1$, then output $\sqrt{x} = x^{(s+1)/2}$ and input another element. If not, execute **Procedure** 2 and **Procedure** 3 in order.

(2) Let $\tau[0] = T-1$, $\mu = 1$ and $k = 1$, and then repeatedly execute the following steps in order until k becomes $t+1$.

- Step 1: Compute $\sigma = x[t-k]\prod_{i=0}^{\mu-1} c[\tau[i]]$. If $\sigma = -1$, then $\tau[\mu] = T-1$ and $\mu = \mu+1$. If not, the values of $\tau[\mu]$ and μ are not modified.
- Step 2: For $0 \leqslant i < \mu$, let $\tau[i] = \tau[i]-1$.
- Step 3: Let $k = k+1$.

(3) Compute the following equation:

$$\sqrt{x} = x^{(s+1)/2}\prod_{i=0}^{\mu-1} c[\tau[i]].$$

When $t=0$, **Procedure2** is skipped and the MW-ST algorithm remains valid. Note that $x_0, x_1, \cdots, x_t$ and $x^{(s+1)/2}$ required for the MW-ST algorithm have been computed in QR test and saved in the memory. So we only need to call those data in the MW-ST algorithm, which makes the SQRT computation terribly efficient. We should also note that it is not necessary to implement exponentiations to a i_k-th power in the MW-ST algorithm, because i_k is equal to 1 or -1 for $1 \leqslant k \leqslant t$. For example, in (20), we save the value of c_{T-t-1} into the memory when $i_1 = 1$; we ignore the value of c_{T-t-1} when $i_1 = 0$.

4.2 Computational Complexity of MW-ST Algorithm over Prime Fields $GF(p)$

From what described in Section 4.1, it is easy to know that the number of the multiplications required for the MW-ST algorithm depends on the values of i_k and t shown in (20), where $k = 1, 2, \cdots, t$. When $i_k \equiv 0$ and $t = 0$, the number of the multiplications required for the MW-ST algorithm becomes the minimum. When $i_k \equiv 1$ and $t = T-1$, the number of the multiplications required for the MW-ST algorithm becomes the maximum and $\sigma(k)$ and $\sqrt{x}$ have the form:

$$\sigma(k) = c_{T-1}c_{T-2}\cdots c_{T-k}x_{T-1-k}, \tag{22a}$$

$$\sqrt{x} = c_{T-1}c_{T-2}\cdots c_0 x^{(s+1)/2}. \tag{22b}$$

where $k = 1, 2, \cdots, T-1$.

Because the values of i_k and t depend on the input element x that can not be known in advance, we only consider the maximum for the convenience. Based on (22a), one multiplication is required for $\sigma(1) = c_{T-1}x_{T-2}$. 2 multiplications are required for $\sigma(2) = c_{T-1}c_{T-2}x_{T-3}$. 3 multiplications are required for $\sigma(3) = c_{T-1}c_{T-2}c_{T-3}x_{T-4}$. In this way, it is easy to know that k multiplications are required for $\sigma(k)$, where $k = 1, 2, \cdots, T-1$. It is also easy to know that T multiplications are required for (22b), where $x^{(s+1)/2}$ has been computed in QR test. In total, the MW-ST algorithm over $GF(p)$ at most needs the following multiplications:

$$\#M_1 = \frac{T^2+T}{2}. \tag{23}$$

4.3 Computational Complexity of MW-ST Algorithm over Extension Fields $GF(p^2)$

In $GF(p^2)$, $(p-1)/2$ and $p+1$ can be factorized as

$$(p-1)/2 = 2^{T_1}s_1, \quad p+1 = 2^{T_2}s_2, \tag{24}$$

from (21) it follows that $T=T_1+T_2$ and $s=s_1s_2$.

Since $x^{p+1} \in GF(p)$ for any $x \in GF(p^2)$, it follows that $(x^{p+1})^{s_1} \in GF(p)$ for any $x \in GF(p^2)$. From (24), we know $(x^{p+1})^{s_1} = (x_0)^{2^{T_2}}$, where $x_0 = x^s$. We thus have $(x_0)^{2^{T_2}} \in GF(p)$ for any $x \in GF(p^2)$.

As described in Section 1., p is a prime number, then we have $2 \mid p-1$, where there are only two possibilities: $4 \nmid p-1$ or $4 \mid p-1$. Since only one of the four continuous integers $p-1$, p, $p+1$, $p+2$ can be divided by 4, it implies that

(i) $T_1 \equiv 0$ when $4 \nmid p-1$;

(ii) $T_2 \equiv 1$ when $4 \mid p-1$.

In what follows, for $4 \nmid p-1$ and $4 \mid p-1$, we evaluate the computational complexity of the MW-ST algorithm over $GF(p^2)$ respectively.

When $4 \nmid p-1$, based on (i) and $(x_0)^{2^{T_2}} \in GF(p)$, we have $(x_0)^{2^T} \in GF(p)$ for any $x \in GF(p^2)$, which implies $x, x_0, x_1, \cdots x_{t-1}$ and $c, c_0, c_1, \cdots, c_{T-1}$ all belong to $GF(p^2)$. In this case, the computational complexity of MW-ST algorithm over $GF(p^2)$ is the same as described in Section 4.2 except that all the computations are implemented in $GF(p^2)$. Therefore, the MW-ST algorithm over $GF(p^2)$ at most needs the following multiplications:

$$\#M_2 = \frac{T^2+T}{2}. \tag{25}$$

When $4 \mid p-1$, based on (ii) and $(x_0)^{2^{T_2}} \in GF(p)$, we have $(x_0)^2 \in GF(p)$ for any $x \in GF(p^2)$. Same as above, the maximum complexity is evaluated. It follows that we only need to evaluate the computational complexity of Eqs.(22). Since $x_0, x_1, \cdots x_{t-1}$ and $c_1, \cdots, c_{T-1}$ all belong to $GF(p)$,

Table 1 Computational Complexity and Running Time (CPU: Pentium4, (2.6GHz))

		A.Computational Complexity			B. Running Time
		$\#M_1$	$\#A_1$	$\#S$	[μs]
$GF(p)$	Proposed QR Test	15	0	0	0.53
	Smart algorithm	256	0	0	3.75
	MW-ST algorithm	120	0	0	1.99
$GF(p^2)$	Proposed QR Test	93	186	31	4.69
	Smart algorithm	909	1818	303	2.38 x 10
	MW-ST algorithm	139	6	1	2.31

Remarks: $p = 2^{16} + 1$, where $4 \mid p - 1$.

Table 2 Computational Complexity and Running Time (CPU: Pentium4, (2.6GHz))

		A.Computational Complexity			B. Running Time
		$\#M_1$	$\#A_1$	$\#S$	[μs]
$GF(p)$	Proposed QR Test	15	0	0	0.75
	Smart algorithm	15	0	0	0.62
	MW-ST algorithm	0	0	0	0.032
$GF(p^2)$	Proposed QR Test	99	198	33	7.97
	Smart algorithm	117	234	39	8.72
	MW-ST algorithm	9	18	3	1.56

Remarks: $p = 2^{16} + 3$, where $4 \nmid p - 1$.

same as in Section 4. 2, we require k multiplications in $GF(p)$ to compute (22a) for $k = 1, 2, \cdots, T-1$.

In (22b), we first compute $c_{T-1} \cdots c_1$ and $c_0 x^{(s+1)/2}$ respectively, and then multiply them together, where $x^{(s+1)/2}$ has been computed in QR test. To compute $c_{T-1} \cdots c_1$, we require $T-2$ multiplications in $GF(p)$. To compute $c_0 x^{(s+1)/2}$, we require 1 multiplication in $GF(p^2)$ because $c_0, x^{(s+1)/2} \in GF(p^2)$. To compute the product of $c_{T-1} \cdots c_1$ and $c_0 x^{(s+1)/2}$, we require 2 multiplications in $GF(p)$ because $c_{T-1} \cdots c_1 \in GF(p)$ and $c_0 x^{(s+1)/2} \in GF(p^2)$. In total, the MW-ST algorithm over $GF(p^2)$ at most needs the following operations:

$$\#M_1 = \frac{T^2 + T}{2}, \tag{26a}$$

$$\#M_2 = 1. \tag{26b}$$

5. Simulation Results

In this section, we set the characteristic p and extension degree m of finite fields $GF(p^m)$ as follows:

$$p = 2^{16} + 3 = 65539, \quad \text{where } 4 \nmid p - 1, \tag{27a}$$

$$p = 2^{16} + 1 = 65537, \quad \text{where } 4 \mid p - 1, \tag{27b}$$

$$m = 1 \text{ or } 2. \tag{27c}$$

Smart and the MW-ST algorithms in $GF(p)$ and $GF(p^2)$ were implemented on a Pentium4 (2.6GHz) with C++ programing language, where $GF(p^2)$ are constructed as OEF by adopting the following binomial as the modular polynomial:

$x^2 - 3$. (28)

As described in 1., 1 multiplication in $GF(p^2)$ requires 3 multiplications in $GF(p)$, 6 additions in $GF(p)$ and a shift computation. Based on Eqs.(27), (5), (15), (23), (25) and Eqs.(26), in the column A of Table 1 and Table 2, we can thus evaluate the complexity of Smart and the MW-ST algorithms in $GF(p)$ and $GF(p^2)$ such as $\#A_1$, $\#M_1$. From the column A we see that $\#M_1$ required for the MW-ST algorithm is much less than that required for Smart algorithm. The main reason is that most data required for the MW-ST algorithm have been computed in QR test.

Inputting 500000 QRs randomly generated, the computation time for the two algorithms was measured on average in column B. When $p = 65537$, Table 1 clearly shows that the MW-ST algorithm reduces the computation amount 2 times and 7 times less than Smart algorithm over $GF(p)$ and $GF(p^2)$, respectively, which makes the MW-ST algorithm accelerate the SQRT computation 2 times and 10 times faster than Smart algorithm over $GF(p)$ and $GF(p^2)$, respectively. When $p = 65539$, the MW-ST algorithm accelerate the SQRT computation 20 times and 6 times faster than Smart algorithm over $GF(p)$ and $GF(p^2)$, respectively. Since the computation amounts for the two algorithms depend on the value of the input element x that can not be known in advance, we only considered the maximum computation amounts required for the two algorithms in the paper. So, there was a little difference between the ratio of running time for Smart algorithm to that for the MW-ST algorithm and the ratio of the computational amount for Smart algorithm to that for the MW-ST algorithm.

6. Conclusion

In $GF(p^m)$, we have proposed the MW-ST algorithm for SQRT computation which was proved to be much faster than the previously known fastest Smart algorithm. Taking example for $p = 65537$ and $p = 65539$, we implemented Smart and the MW-ST algorithms on a Pentium4 (2.6GHz) with C++ programing language. The computer simulations showed that for $p = 2^{16} + 1$ the proposed algorithm on average accelerates the SQRT computation 2 times and 10 times faster than Smart algorithm over $GF(p)$ and $GF(p^2)$, respectively and for $p = 2^{16} + 3$ the proposed algorithm on average accelerates the SQRT computation 20 times and 6 times faster than Smart algorithm over $GF(p)$ and $GF(p^2)$, respectively. Consequently we can conclude that the MW-ST algorithm is quite effective.

References

[1] Y. Nogami and Y. Morikawa, "Fast generation of elliptic curves with prime order over $F_{p^{2^c}}$," Proc. WCC 2003, pp. 347 – 356, Mar. 2003.

[2] Y. Nogami and Y. Morikawa, "A fast implementation of elliptic curve cryptosystem with prime order defined over F_{p^8}," Memories of the Faculty of Engineering Okayama University, vol. 37, no. 2, pp. 73 – 88, Mar. 2003.

[3] S. Kwon, C. H. Kim and C. P. Hong, "Efficient exponentiation for a class of finite fields $GF(2^n)$ determined by Gauss Periods," Proc. CHES 2003, LNCS 2779, pp. 228 – 242, Springer–Verlag, 2003.

[4] http://www. ieee. org/groups/1363

[5] I. Blake, G. Seroussi and N. Smart, Elliptic Curves in Cryptography, LMS 265, Cambridge University Press, 1999.

[6] D. E. Knuth, The Art of Computer Programming. 3rd ed. Addison-Wesley, 1997, vol. 2.

Appendix

1. Fundamental Operations in $GF(p^2)$

This appendix briefly shows how to implement the fundamental operations such as additions and multiplications in $GF(p^2)$ which is constructed as an OEF by adopting the modular polynomial $x^2 - 3$.

Let $\omega \in GF(p^2)$ be a root of the irreducible polynomial $x^2 - 3$, it follows [2] that $\{1, \omega\}$ is the basis of $GF(p^2)$ over $GF(p)$. Using the basis, the arbitrary elements $A, B \in GF(p^2)$ can be uniquely represented as follows:

$$A = a_0 + a_1\omega, \quad a_0, a_1 \in GF(p). \tag{A·1a}$$

$$B = b_0 + b_1\omega, \quad b_0, b_1 \in GF(p). \tag{A·1b}$$

The sum and difference of A and B are given by

$$A \pm B = (a_0 \pm b_0) + (a_1 \pm b_1)\omega. \tag{A·2}$$

Using Karatsuba method [6], the product AB is given by

$$AB = (C_0 + 3C_1) + (C_2 - C_1 - C_0)\omega, \tag{A·3}$$

where $C_0 = a_0b_0$, $C_1 = a_1b_1$ and $C_2 = (a_0 + a_1)(b_0 + b_1)$. Note that $3C_1 = 2C_1 + C_1$, where $2C_1$ can be implemented by a shift computation. Therefore, 3 multiplications in $GF(p)$ and 6 additions in $GF(p)$ and a shift computation are required for 1 multiplication in $GF(p^2)$.