AAECC (2019) 30:135–145
https://doi.org/10.1007/s00200-018-0362-2

ORIGINAL PAPER

On the Cipolla–Lehmer type algorithms in finite fields

Gook Hwa Cho[1] · Byeonghwan Go[2] · Chang Heon Kim[3] · Namhun Koo[3] · Soonhak Kwon[3]

Received: 12 September 2017 / Revised: 17 April 2018 / Accepted: 29 May 2018 /
Published online: 25 June 2018

Abstract In this paper, we present a refinement of the Cipolla–Lehmer type algorithm given by H. C. Williams in 1972, and later improved by K. S. Williams and K. Hardy in 1993. For a given r-th power residue $c \in \mathbb{F}_q$ where r is an odd prime, the algorithm of H. C. Williams determines a solution of $X^r = c$ in $O(r^3 \log q)$ multiplications in $\mathbb{F}_q$, and the algorithm of K. S. Williams and K. Hardy finds a solution in $O(r^4 + r^2 \log q)$ multiplications in $\mathbb{F}_q$. Our refinement finds a solution in $O(r^3 + r^2 \log q)$ multiplications in $\mathbb{F}_q$. Therefore our new method is better than the previously proposed algorithms independent of the size of r, and the implementation result via SageMath shows a substantial speed-up compared with the existing algorithms. It should be mentioned that our method also works for a composite r.

Keywords Finite field · r-th root · Cipolla–Lehmer algorithm · Adleman–Manders–Miller algorithm · Primitive root

Mathematical Subject Classification 11T06 · 11Y16 · 68W40

1 Introduction

Let $r > 1$ be an integer and q be a power of a prime. Finding r-th root (or finding a root of $X^r = c$) in finite field $\mathbb{F}_q$ has many applications in computational number

✉ Soonhak Kwon
shkwon@skku.edu

[1] Institute for Mathematical Sciences, Ewha Womans University, Seoul 03765, Republic of Korea

[2] Department of Mathematics, Sungkyunkwan University, Suwon 16419, Republic of Korea

[3] Applied Algebra and Optimization Research Center, Sungkyunkwan University, Suwon 16419, Republic of Korea

theory and in many other related topics. Some such examples include point halving and point compression on elliptic curves [1], where square root computations are needed. Similar applications for high genus curves require r-th root computations also. Note that, for a prime r, r-th root finding is easy when $q \not\equiv 1 \pmod r$ because one has c^u as an r-th root of c, where $ru \equiv 1 \pmod{q-1}$ and such u can be found using Extended Euclidean algorithm. Therefore we are mainly interested in the case $q \equiv 1 \pmod r$ in this paper.

Among several available root extraction methods of the equation $X^r - c = 0$, there are two well known algorithms applicable for arbitrary integer $r > 1$; the Adleman–Manders–Miller algorithm [2,3], a straightforward generalization of the Tonelli–Shanks square root algorithm [4,5] to the case of r-th root extraction, and the Cipolla–Lehmer algorithms [6,7]. Due to the cumbersome extension field arithmetic needed for the Cipolla–Lehmer algorithm, one usually prefers the Tonelli–Shanks or the Adleman–Manders–Miller, and related researches [8,9] are reported which improve the Tonelli–Shanks by accelerating the discrete logarithm computation in $\mathbb{F}_q$.

The efficiency of the Adleman–Manders–Miller algorithm heavily depends on the exponent ν of a prime r satisfying $r^\nu | q-1$ and $r^{\nu+1} \nmid q-1$, which becomes quite slow if $\nu \approx \log q$. For example, in the case of $r = 2$, it had been observed in [10] that, for a prime $p = 9\times 2^{3354}+1$, running the Tonelli–Shanks algorithm using various softwares such as Magma, Mathematica and Maple cost roughly 5, 45, 390 min, respectively while the Cipolla–Lehmer costs under 1 min in any of the above softwares. It should be mentioned that such extreme cases (of p with $p-1$ divisible by high powers of 2) may happen in some cryptographic applications. For example, one of the NIST suggested curves [1] P-224 : $y^2 = x^3 - 3x + b$ over $\mathbb{F}_p$ uses a prime $p = 2^{224} - 2^{96} + 1$.

When q is of special type (such as high powers of small primes), very effective r-th root algorithms are reported in [11,12]. Also, root finding algorithms with fewer exponentiations by exploiting special properties of q are dealt in [10,13–17]. Another algorithm, so called the Pocklington algorithm [18], which is different from the Adleman–Manders–Miller or the Cipolla–Lehmer in the sense that the Pocklington algorithm uses the splitting property $\mathbb{F}_q[X]/\langle X^r - c\rangle \cong \mathbb{F}_{q^r}$ where c is an r-th power in $\mathbb{F}_q$, are studied in [19–22]. Note that, unlike the case of the Adleman–Manders–Miller and the Cipolla–Lehmer, there is no generalized version (applicable for any r) of the Pocklington algorithm except the cases $r = 2, 3, 5$. The case $r = 5$ in [22] is already complicated and the techniques are quite different from the cases $r = 2, 3$.

A generalization to r-th root extraction of the Cipolla–Lehmer square root algorithm was proposed by H. C. Williams [22] and the complexity of the proposed algorithm is $O(r^3 \log q)$ multiplications in $\mathbb{F}_q$. A refinement of the algorithm in [22] was given by K. S. Williams and K. Hardy [23] where the complexity is reduced to $O(r^4 + r^2 \log q)$ multiplications in $\mathbb{F}_q$. For the case of the square root, a new Cipolla–Lehmer type algorithm based on the Lucas sequence was given by Müller [10], and a similar result for the case $r = 3$ was also obtained in [24].

In this paper, we present a new Cipolla–Lehmer type algorithm for r-th root extractions in $\mathbb{F}_q$ whose complexity is $O(r^3 + r^2 \log q)$ multiplications in $\mathbb{F}_q$, which improves previously proposed results. The improvement is made by replacing a large exponentiation in $\mathbb{F}_{q^r}$ by products mainly in $\mathbb{F}_q$ and a smaller exponentiation. In our complexity estimation, we followed the standard approach in [22,23] using $\mathbb{F}_q$ operations (or mul-

tiplications) as our basic counting units. However one may also follow the Boolean model approach [25,26] to derive a more tight complexity bound. We also compare our algorithm with those in [22,23] using the software SageMath, and show that our algorithm performs consistently better than those in [22,23] as is expected from the theoretical complexity estimation. Only the case where r is an odd prime was considered in [22,23] but we will give the general arguments (i.e., no restriction on r) here. It should be mentioned that almost all algorithms referred in this paper including ours are randomized algorithms of Las Vegas type. A new result on the deterministic algorithm is recently appeared in [27].

The remainder of this paper is organized as follows: In Sect. 2, we briefly summarize the Cipolla–Lehmer algorithm, and introduce the works of H. C. Williams [22] and K. S. Williams and K. Hardy [23]. In Sect. 3, we present our refinement of the Cipolla–Lehmer algorithm. In Sect. 4, we give the complexity analysis of our algorithm and show the result of SageMath implementations of the three algorithms (in [22,23], and ours). Finally, in Sect. 5, we give the concluding remarks.

2 Cipolla–Lehmer algorithm in $\mathbb{F}_q$

Let q be a prime power and $\mathbb{F}_q$ be a finite field with q elements. Let $c \neq 0 \in \mathbb{F}_q$ be an r-th power residue in $\mathbb{F}_q$ for an integer $r > 1$ with $q \equiv 1 \pmod{r}$. *We restrict r as an odd prime in this section*, since the results of H. C. Williams [22] and K. S. Williams and K. Hardy [23] are only valid when r is an odd prime.

2.1 H. C. Williams' algorithm

Let $b \in \mathbb{F}_q$ be an element such that $b^r - c$ is not an r-th power residue in $\mathbb{F}_q$. Such b can be found after r random trials of b. (See pp. 479–480 in [23] for further explanation.) Then the polynomial $X^r - (b^r - c)$ is irreducible over $\mathbb{F}_q$ and there exists $\theta \in \mathbb{F}_{q^r} - \mathbb{F}_q$ such that $\theta^r = b^r - c$. Let $\omega = \theta^{q-1} = (b^r - c)^{\frac{q-1}{r}}$. Then we have $\omega^r = 1$ where ω is a primitive r-th root because $b^r - c$ is not an r-th power in $\mathbb{F}_q$.

For all $0 \leq i \leq r-1$, using $q \equiv 1 \pmod{r}$, one has $\theta^{q^i} = \theta \cdot \theta^{q^i - 1} = \theta \cdot \left(\theta^{q-1}\right)^{1+q+\cdots+q^{i-1}} = \theta\omega^i$, which implies $(b-\theta)^{q^i} = b - \theta^{q^i} = b - \omega^i\theta$. Letting $\alpha = b - \theta$, one has

$$\alpha^{\sum_{j=0}^{r-1} q^j} = (b-\theta)^{1+q+q^2+\cdots+q^{r-1}} = \prod_{i=0}^{r-1}(b - \omega^i\theta) = b^r - \theta^r = c. \tag{1}$$

Thus one may find an r-th root of c by computing $\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}} \in \mathbb{F}_q[\theta] = \mathbb{F}_q[X]/\langle X^r - (b^r - c)\rangle$.

Proposition 1 (H. C. Williams)
Suppose that $c \neq 0$ is an r-th power in $\mathbb{F}_q$. Let $\theta^r = b^r - c$ with $\theta \in \mathbb{F}_{q^r}$ and $b \in \mathbb{F}_q$ such that $b^r - c$ is not an r-th power in $\mathbb{F}_q$. Then letting $\alpha = b - \theta$,

Algorithm 1 H. C. Williams' r-th root algorithm [22]

Input : An r-th power residue c in $\mathbb{F}_q$
Output : $x \in \mathbb{F}_q$ satisfying $x^r = c$

1: **do** Choose a random $b \in \mathbb{F}_q$ **until** $b^r - c$ is not an r-th power residue.
2: $M \leftarrow \frac{1+q+\cdots+q^{r-1}}{r}$
3: $A \leftarrow (b, -1, 0, ..., 0)$ // A is a coefficient vector of $\alpha = b - \theta$. //
4: $x \leftarrow A^M$ // $x = \alpha^M$. Use recurrence relation (3). //
5: **return** x

$$\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}} \in \mathbb{F}_q$$

is an r-th root of c.

The usual 'square and multiply method' (or 'double and add method' if one uses a linear recurrence relation) requires roughly $\log \frac{\sum_{j=0}^{r-1} q^j}{r} \approx r \log q$ steps for the evaluation of $\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}}$, and therefore the complexity of the algorithm of H. C. Williams is $O(r^3 \log q)$ multiplications in $\mathbb{F}_q$. H. C. Williams' result can be expressed in Algorithm 1 using the recurrence relation technique of Sect. 2.2.

Note that $\alpha = b + \theta$ is used in the original paper [22], while our presentation is based on [23] where it uses $\alpha = b - \theta$. We followed [23] because it is more convenient to deal with general r which is not necessarily odd prime. For example, if one uses $\alpha = b + \theta$ as in [22], then the case of even r (such as $r = 2$) cannot be covered. Detailed explanations will be given in Sect. 3.

2.2 Recurrence relation

Given $\sum_{i=0}^{r-1} a_i \theta^i \in \mathbb{F}_q[\theta]$, define $a_i(j) \in \mathbb{F}_q$ $(0 \le i \le r-1,\ 1 \le j)$ as

$$\sum_{i=0}^{r-1} a_i(j)\theta^i = \left(\sum_{i=0}^{r-1} a_i \theta^i\right)^j. \tag{2}$$

In particular, one has $a_i(1) = a_i$ for all $0 \le i \le r-1$. Then one has

$$\begin{aligned}\sum_{i=0}^{r-1} a_i(m+n)\theta^i &= \left(\sum_{i=0}^{r-1} a_i(m)\theta^i\right)\left(\sum_{j=0}^{r-1} a_j(n)\theta^j\right) \\ &= \sum_{l=0}^{r-1}\left(\sum_{j=0}^{l} a_j(m)a_{l-j}(n)\right)\theta^l \\ &\quad + (b^r - c)\sum_{l=0}^{r-2}\left(\sum_{j=l+1}^{r-1} a_j(m)a_{l+r-j}(n)\right)\theta^l,\end{aligned}$$

which implies

$$a_l(m+n) = \sum_{j=0}^{l} a_j(m)a_{l-j}(n) + (b^r - c)\sum_{j=l+1}^{r-1} a_j(m)a_{l+r-j}(n) \tag{3}$$

for all $0 \le l \le r-1$. When $l = r-1$, the second summation in the Eq. (3) does not happen so that one has $a_{r-1}(m+n) = \sum_{j=0}^{r-1} a_j(m)a_{r-1-j}(n)$.

2.3 An improvement of K. S. Williams and K. Hardy

Williams and Hardy [23] improved the algorithm of H. C. Williams by reducing the loop length to $\log q$ as follows. Write $\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}}$ (where $\alpha = b - \theta$) as

$$\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}} = E_1^{\frac{q-1}{r}} \cdot E_2, \tag{4}$$

where

$$E_1 = \alpha^{(q-1)^{r-2}}, \qquad E_2 = \alpha^{\frac{q^r-1}{r(q-1)} - \frac{(q-1)^{r-1}}{r}}.$$

By noticing that the exponent $\frac{q^r-1}{r(q-1)} - \frac{(q-1)^{r-1}}{r}$ of E_2 is a polynomial of q with integer coefficients and using the binomial theorem, one has the following expression of E_1 and E_2 as

$$E_1 = \prod_{i=0}^{r-2} X_i \quad \text{with } X_i = (b - \omega^i\theta)^{(-1)^{r-i}\binom{r-2}{i}}, \tag{5}$$

$$E_2 = \prod_{i=1}^{r-1} Y_i \quad \text{with } Y_i = (b - \omega^{r-i-1}\theta)^{\frac{1-(-1)^i\binom{r-1}{i}}{r}}. \tag{6}$$

Thus we have the following result of Williams and Hardy.

Proposition 2 (Williams-Hardy)

(1) *Under same assumption as in Proposition 1,* $E_1^{\frac{q-1}{r}} \cdot E_2$ *is an* r*-th root of* c*, where*

$$E_1 = \alpha^{(q-1)^{r-2}}, \qquad E_2 = \alpha^{\frac{q^r-1}{r(q-1)} - \frac{(q-1)^{r-1}}{r}}.$$

(2) E_1 *and* E_2 *can be efficiently computed using the relations*

$$E_1 = \prod_{i=0}^{r-2}(b - \omega^i\theta)^{(-1)^{r-i}\binom{r-2}{i}}, \qquad E_2 = \prod_{i=1}^{r-1}(b - \omega^{r-i-1}\theta)^{\frac{1-(-1)^i\binom{r-1}{i}}{r}}.$$

Algorithm 2 Williams-Hardy r-th root algorithm [23]

Input : An r-th power residue c in $\mathbb{F}_q$
Output : $x \in \mathbb{F}_q$ satisfying $x^r = c$

1: **do** Choose a random $b \in \mathbb{F}_q$ **until** $b^r - c$ is not an r-th power residue.
2: $\omega \leftarrow (b^r - c)^{\frac{q-1}{r}}$, where $\theta^r = b^r - c$.
3: $E_1 \leftarrow 1,\ E_2 \leftarrow 1$
4: **for** i from 1 to $r-1$ **do**
5: $\quad X_i \leftarrow (b - \omega^{i-1}\theta)^{(-1)^{r-i+1}\binom{r-2}{i-1}},\ Y_i \leftarrow (b - \omega^{r-i-1}\theta)^{\frac{1-(-1)^i\binom{r-1}{i}}{r}}$
6: $\quad E_1 \leftarrow E_1 \cdot X_i,\ E_2 \leftarrow E_2 \cdot Y_i$
7: $E_1' \leftarrow E_1^{\frac{q-1}{r}}$ // $E_1' = E_1^{\frac{q-1}{r}}$. Use recurrence relation (3). //
8: $x \leftarrow E_1' \cdot E_2$
9: **return** x

The complexity of computing each of X_i in the Eq. (5) is of $O(\log q) + O(r) + O\left(r^2 \log \binom{r-2}{i}\right)$ multiplications in $\mathbb{F}_q$. Hence all X_i can be computed in $O(r \log q + r^4)$ $\mathbb{F}_q$-multiplications. Since the $O(r)$ multiplications of all X_i $(0 \le i \le r-2)$ in $\mathbb{F}_{q^r}$ need $O(r^3)$ multiplications in $\mathbb{F}_q$, the total complexity of computing E_1 (as a polynomial of θ degree at most $r-1$) is $O(r \log q + r^4)$ $\mathbb{F}_q$-multiplications. Similarly the complexity of computing E_2 is also $O(r \log q + r^4)$ $\mathbb{F}_q$-multiplications. For a detailed explanation, see [23]. Since the exponentiation $E_1^{\frac{q-1}{r}}$ (using the recurrence relation) needs $O(r^2 \log \frac{q-1}{r}) = O(r^2 \log q)$ multiplications in $\mathbb{F}_q$ and since the multiplication of two elements $E_1^{\frac{q-1}{r}}$ and E_2 needs $O(r)$ multiplications in $\mathbb{F}_q$ (because only the constant term of the θ expansion is needed), the total cost of computing an r-th root of c using the algorithm of K. S. Williams and K. Hardy [23] is $O(r^2 \log q + r^4)$.

3 Our new r-th root algorithm

In this section, we give an improved version of the Cipolla–Lehmer type algorithm by generalizing the method of [23]. Our new algorithm is applicable for all (either prime or composite) $r > 1$ with $q \equiv 1 \pmod r$. *Therefore, throughout this section, we assume that r is not necessarily a prime*. Thus $\omega = \theta^{q-1} = (b^r - c)^{\frac{q-1}{r}}$ may not be a primitive r-th root of unity even if $b^r - c$ is not an r-th power in $\mathbb{F}_q$. Consequently a more stronger condition is needed for the primitivity of ω. That is, ω is a primitive r-th root of unity if and only if $\omega^{\frac{r}{p}} \neq 1$ for every prime $p|r$, which holds if and only if $(b^r - c)^{\frac{q-1}{p}} \neq 1$ for every prime $p|r$. From now on, we will assume that $(b^r - c)^{\frac{q-1}{p}} \neq 1$ for every prime $p|r$ and therefore ω is a primitive r-th root of unity.

Let $\alpha \in \mathbb{F}_{q^r}$. Then, by extracting r-th roots from the following simple identity

$$\alpha^r \left(1 \cdot \alpha \cdot \alpha^{1+q} \cdots \alpha^{1+q+q^2+\cdots+q^{r-2}}\right)^q = \left(1 \cdot \alpha \cdot \alpha^{1+q} \cdots \alpha^{1+q+q^2+\cdots+q^{r-2}}\right) \alpha^{1+q+\cdots+q^{r-1}},$$

one may expect that $\alpha\left(1\cdot\alpha\cdot\alpha^{1+q}\cdots\alpha^{1+q+q^2+\cdots+q^{r-2}}\right)^{\frac{q-1}{r}}$ equals $\alpha^{\frac{1+q+\cdots+q^{r-1}}{r}}$ up to r-th roots of unity. In fact, they are exactly the same element in $\mathbb{F}_q$ and can be verified as follows;

$$\alpha^{\frac{1+q+\cdots+q^{r-1}}{r}} = \alpha^{\frac{\sum_{i=0}^{r-1}q^i}{r}} = \alpha\cdot\alpha^{\frac{(\sum_{i=0}^{r-1}q^i)-r}{r}} \tag{7}$$

$$= \alpha\cdot\alpha^{\frac{\sum_{i=0}^{r-1}(q^i-1)}{r}} = \alpha\cdot\alpha^{\frac{(q-1)\sum_{i=1}^{r-1}\sum_{j=0}^{i-1}q^j}{r}} \tag{8}$$

$$= \alpha\cdot\left(\alpha^{\sum_{i=1}^{r-1}\sum_{j=0}^{i-1}q^j}\right)^{\frac{q-1}{r}} \tag{9}$$

$$= \alpha\cdot\left(1\cdot\alpha\cdot\alpha^{1+q}\cdots\alpha^{1+q+q^2+\cdots+q^{r-2}}\right)^{\frac{q-1}{r}}. \tag{10}$$

Proposition 3 (Main Theorem)
Let $q\equiv 1 \pmod r$ with $r>1$ and let $(b^r-c)^{\frac{q-1}{p}}\neq 1$ for all prime divisors p of r. Then letting $\alpha=b-\theta$ where $\theta^r=b^r-c$,

$$\alpha\cdot\left(1\cdot\alpha\cdot\alpha^{1+q}\cdots\alpha^{1+q+q^2+\cdots+q^{r-2}}\right)^{\frac{q-1}{r}}$$

is an r-th root of c.

Based on the above simple result, we may present a new r-th root algorithm (Algorithm 3) of complexity $O(r^2\log q+r^3)$ with given information of the prime factors of r. It should be mentioned that our proposed algorithm is general in the sense that r can be any (composite) integer >1 satisfying $q\equiv 1 \pmod r$, while r was assumed to be an odd prime both in [22] and [23].

Both in [22] and [23], b was chosen so that $\omega=(b^r-c)^{\frac{q-1}{r}}\neq 1$, and since r is prime, ω is automatically a primitive r-th root. This property guarantees the validity of the Eq. (1), namely

$$(b-\theta)(b-\omega\theta)(b-\omega^2\theta)\cdots(b-\omega^{r-1}\theta)=b^r-\theta^r=c. \tag{11}$$

However if r is composite, then $\omega=(b^r-c)^{\frac{q-1}{r}}$ is not a primitive r-th root in general. In fact, letting $s>1$ be the least positive integer satisfying $\omega^s=1$, the degree of the irreducible polynomial of θ (where $\theta^r=b^r-c$) is s because

$$\theta^{q^s-1}=(\theta^{q-1})^{q^{s-1}+q^{s-2}+\cdots+q+1}=\omega^{q^{s-1}+q^{s-2}+\cdots+q+1}=\omega^s, \tag{12}$$

and one has

$$\begin{aligned}(b-\theta)(b-\omega\theta)\cdots(b-\omega^{r-1}\theta) &= \{(b-\theta)(b-\omega\theta)\cdots(b-\omega^{s-1}\theta)\}^{\frac{r}{s}}\\ &= (b^s-\theta^s)^{\frac{r}{s}}\neq c\end{aligned} \tag{13}$$

Algorithm 3 Our new r-th root algorithm

Input : An r-th power residue c in $\mathbb{F}_q$
Output : $x \in \mathbb{F}_q$ satisfying $x^r = c$

1: **do** Choose a random $b \in \mathbb{F}_q$ **until** $(b^r - c)^{\frac{q-1}{r}}$ is a primitive r-th root of unity.
2: $\omega \leftarrow (b^r - c)^{\frac{q-1}{r}}$, $\alpha \leftarrow b - \theta$ where $\theta^r = b^r - c$.
3: $P \leftarrow \alpha$, $A \leftarrow \alpha$, $W \leftarrow 1$
4: **for** $i = 1$ **to** $r - 2$ **do** //$A, P \in \mathbb{F}_q[\theta]$ and $W \in \mathbb{F}_q$//
5: $W \leftarrow W\omega$, $V \leftarrow b - W\theta$ //$W = \omega^i$, $V = b - \omega^i\theta = \alpha^{q^i}$//
6: $A \leftarrow AV$, $P \leftarrow PA$ //$A = \alpha^{1+q+\cdots+q^i}$, $P = \alpha \cdot \alpha^{1+q} \cdots \alpha^{1+q+\cdots+q^i}$//
7: $B \leftarrow P^{\frac{q-1}{r}}$ // Use recurrence relation (3). //
8: $x \leftarrow \alpha \cdot B$ // $x \in \mathbb{F}_q$ //
9: **return** x

if $s < r$. Therefore the methods of [22] and [23] do not work for a composite r unless one assumes the primitivity of ω.

Also, even if one assumes the primitivity of $\omega = (b^r - c)^{\frac{q-1}{r}}$, one still has some problems both in [22] and [23], which will be explained in the following remarks.

Remark 1 In [22], $\alpha = b + \theta$ was used (instead of $b - \theta$) under the assumption of $\theta^r = c - b^r$ with $(c - b^r)^{\frac{q-1}{r}} \neq 1$. If we choose $\alpha = b + \theta$ following [22], then we get

$$(b+\theta)(b+\omega\theta)\cdots(b+\omega^{r-1}\theta) = b^r - (-\theta)^r = b^r + (-1)^{r+1}\theta^r. \tag{14}$$

Therefore if r is an odd prime (as was originally assumed in [22]), one has $b^r + \theta^r = c$ and the r-th root algorithm is essentially same to the case $\alpha = b - \theta$. However when r is even (for example, when $r = 2$), the original method in [22] cannot be used because $b^r + (-1)^{r+1}\theta^r = b^r - \theta^r \neq c$.

Remark 2 The algorithm in [23] needs E_1 and E_2 satisfying $\alpha^{\frac{\sum_{j=0}^{r-1} q^j}{r}} = E_1^{\frac{q-1}{r}} \cdot E_2$. However for composite r, E_2 cannot be well-defined in some cases, since the exponent $\frac{1-(-1)^i\binom{r-1}{i}}{r}$ in the Eq. (6) is not an integer in general. That is, the property $(-1)^i\binom{r-1}{i} \equiv 1 \pmod{r}$ only holds when r is prime. Therefore the algorithm in [23] fails to give the answer when r is composite such as $r = 4, 6, 9, \ldots$. (i.e., when $r = 4$, one has $E_2 = \alpha^{q^2 - \frac{1}{2}q}$ so the coefficient $\frac{1}{2}$ of q in the exponent is not an integer and one cannot compute E_2.) The problem of E_2 being undefined is unavoidable even if one assumes the primitivity of ω.

4 Complexity analysis and comparison

4.1 Complexity analysis

An initial step of the proposed algorithm requires one to find a primitive r-th root ω in $\mathbb{F}_q$. When r is prime, one only needs to find b satisfying $\omega = (b^r - c)^{\frac{q-1}{r}} \neq 0, 1$

and the probability that a random b satisfies the required property is $\frac{1}{r} + O(q^{-\frac{1}{4}})$ ([23] p. 480) under the assumption of $r \leq q^{\frac{1}{4}}$. When r is composite, one further needs to check whether $\omega^{\frac{r}{p}} \neq 1$ for every prime divisor p of r. Since the complexity estimation $O(r^3 \log q)$ in [22] and $O(r^2 \log q + r^4)$ in [23] still hold if one assumes that a primitive root $\omega = (b^r - c)^{\frac{q-1}{r}}$ is already given, we will also assume that a primitive root ω is given in our estimation for a fair comparison.

At each i-th step of the for-loop of our proposed algorithm, step 5 needs 1 $\mathbb{F}_q$ multiplication. In step 6, the computation AV needs 1 $\mathbb{F}_{q^r}$ multiplication which, in fact, can be executed with $2r$ $\mathbb{F}_q$ multiplications because $V = b - \omega^i \theta$ is linear in θ. The computation PA needs 1 $\mathbb{F}_{q^r}$ multiplication which can be executed with r^2 $\mathbb{F}_q$ multiplications. Therefore, at the end of the for-loop, one needs at most $(r-2)(1+2r+r^2) < (r+1)^3$ $\mathbb{F}_q$ multiplications (of order $O(r^3)$). Since the exponentiation $P^{\frac{q-1}{r}}$ (in steps 7–9) needs $O(r^2 \log q)$ $\mathbb{F}_q$ multiplications, the total cost of our proposed algorithm is $O(r^3 + r^2 \log q)$ multiplications in $\mathbb{F}_q$. On the other hand, the cost of Algorithm 1 [22] is $O(r^2 \log \frac{q^r-1}{r}) = O(r^3 \log q)$, and the cost of Algorithm 2 [23] is $O(r^4 + r^2 \log q)$ where $O(r^4)$ comes from the cost of computing E_1 and E_2 in steps 4-6 of Algorithm 2. The theoretical estimation shows that our proposed algorithm is better than Algorithm 2 as r gets larger.

Finally, when $r = 2$, the for-loop can be omitted in our algorithm so that one only needs to compute $P \cdot P^{\frac{q-1}{2}}$ which is exactly same to the original Cipolla–Lehmer algorithm.

4.2 Numerical examples

Table 1 shows the implementation results using SageMath of the above mentioned two algorithms and our proposed one. The implementation was performed on Intel Core i7-4770 3.40 GHz with 8 GB memory.

For convenience, we used prime fields $\mathbb{F}_p$ with size about 2000 bits. To choose 2000-bit prime p, we first choose $r = 3, 4, 43, 101, 211$ and a fixed value of s, and considered an integer $p = r^s \lfloor 2^{2000}/(1.1 \cdot r^s) \rfloor + 1$ which is $\approx 2^{2000}$. We did a primality testing of the numbers $p + k \cdot r^s (k = 1, 2, 3, \ldots)$ until we get a prime. That is, the primes p in the Table 1 are the smallest primes $p > 2^{2000}$ satisfying $p \equiv 1 \pmod{r^s}$. Average timings of the r-th root computations for 5 different inputs of r-th power residue $c \in \mathbb{F}_p$ are computed for the primes $r = 3, 4, 43, 101, 211$, and the timings are independent of the choice of the exponents s as is expected from the

Table 1 Running time (in seconds) for r-th root algorithms with $p \approx 2^{2000}$

r	3	4	43	101	211
Algorithm 1 [22]	0.467	Fail	2026.962	Inter.	Inter.
Algorithm 2 [23]	0.254	Fail	53.849	535.043	3956.433
Our proposed algorithm	0.253	0.355	48.359	256.601	1098.401

Table 2 Running time (in seconds) for 11-th root algorithms

p	2^{200}	2^{300}	2^{400}	2^{500}	2^{600}	2^{700}
Algorithm 1 [22]	0.2075	0.4081	0.6719	0.8789	1.1769	1.5097
Algorithm 2 [23]	0.0267	0.0463	0.0750	0.0948	0.1272	0.1623
Our proposed algorithm	0.0225	0.0414	0.0695	0.0900	0.1202	0.1553
Default SageMath algorithm [28]	0.0019	0.0028	0.0086	39.3626	Inter.	Inter.

theoretical complexity. As one can see in the table, our proposed algorithm performs better than the algorithms in [22] and [23]. The table also shows that our algorithm gets dramatically faster than other algorithms as r gets larger. For example, when $r = 101$, our algorithm is roughly 2 times faster than Algorithm 2, and when $r = 211$, our algorithm is 4 times faster than Algorithm 2. For $r = 101, 211$, the computations for Algorithm 1 were interrupted after 3 h.

We also tried to compare our algorithm with the default SageMath algorithm for r-th root finding which is based on [28]. However for our choice of large $p \approx 2^{2000}$, the default algorithm is extremely slow and never terminate. So we tried smaller finite field $\mathbb{F}_p$ with $p \approx 2^{200}, 2^{300}, 2^{400}, 2^{500}, 2^{600}, 2^{700}$ for a fixed $r = 11$. Average timings of the 11-th root computations for 10 different inputs of 11-th power residue $c \in \mathbb{F}_p$ are computed. The results are shown in Table 2, where the computations for the default SageMath algorithm were interrupted after 3 h for $p \approx 2^{600}, 2^{700}$. The comparison table implies that the default SageMath algorithm is quite inefficient than other algorithms including ours when $q > 2^{500}$.

5 Conclusions

We proposed a new Cipolla–Lehmer type algorithm for r-th root extractions in $\mathbb{F}_q$. Our algorithm has the complexity of $O(r^3 + r^2 \log q)$ multiplications in $\mathbb{F}_q$, which improves the previous results of $O(r^3 \log q)$ in [22] and of $O(r^4 + r^2 \log q)$ in [23]. Our algorithm is applicable for any integer $r > 1$, whereas the previous algorithms are effective only for odd prime r. Software implementations via SageMath also show that our proposed algorithm is consistently faster than the previously proposed algorithms, and becomes much more effective as r gets larger.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable suggestions and kind comments. This research was supported by the National Research Foundation of Korea (KRF) Grant funded by the Korea government (MSIP) (No. 2016R1A5A1008055). The work of Soonhak Kwon was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2016R1D1A1B03931912).

References

1. NIST, Digital Signature Standard, Federal Information Processing Standard 186-3, National Institute of Standards and Technology (2000) http://csrc.nist.gov/publications/fips/

2. Adleman, L., Manders, K., Miller, G.: On taking roots in finite fields. In: Proceeding of 18th IEEE Symposium on Foundations on Computer Science (FOCS), pp. 175–177 (1977)
3. Cao, Z., Sha, Q., Fan, X.: Adleman–Manders–Miller root extraction method revisited, (2011) arxiv:1111.4877 (**preprint**)
4. Shanks, D.: Five number-theoretic algorithms, In: Proceeding of 2nd Manitoba Conference on Numerical Mathematics, Manitoba, Canada, pp. 51–70 (1972)
5. Tonelli, A.: Bemerkung über die auflösung quadratischer congruenzen, Göttinger Nachrichten, pp. 344–346 (1891)
6. Cipolla, M.: Un metodo per la risoluzione della congruenza di secondo grado, Rendiconto dell'Accademia Scienze Fisiche e Matematiche, Napoli, Ser. 3, Vol. IX, pp. 154–163 (1903)
7. Lehmer, D.H.: Computer Technology Applied to the Theory of Numbers, Studies in Number Theory, pp. 117–151. Prentice-Hall, Englewood Cliffs (1969)
8. Bernstein, D.: Faster square root in annoying finite field (2001) http://cr.yp.to/papers/sqroot.pdf (**preprint**)
9. Sutherland, A.V.: Structure computation and discrete logarithms in finite abelian p-groups. Math. Comput. **80**, 477–500 (2011)
10. Müller, S.: On the computation of square roots in finite fields. Des. Codes Cryptogr. **31**, 301–312 (2004)
11. Doliskani, J., Schost, E.: Taking roots over high extensions of finite fields. Math. Comput. **83**, 435–446 (2014)
12. Kaltofen, E., Shoup, V.: Fast polynomial factorization over high algebraic extensions of finite fields, In: ISSAC 97, ACM, pp. 184–188 (1997)
13. Atkin, A.O.L.: Probabilistic primality testing, summary by F. Morain, Inria research report, vol. 1779, pp. 159–163 (1992)
14. Barreto, P., Voloch, J.: Efficient computation of roots in finite fields. Des. Codes Cryptogr. **39**, 275–280 (2006)
15. Kong, F., Cai, Z., Yu, J., Li, D.: Improved generalized Atkin algorithm for computing square roots in finite fields. Inf. Process. Lett. **98**(1), 1–5 (2006)
16. Koo, N., Cho, G.H., Kwon, S.: Square root algorithm in $\mathbb{F}_q$ for $q \equiv 2^s + 1 \pmod{2^{s+1}}$. Electron. Lett. **49**, 467–469 (2013)
17. Rotaru, A.S., Iftene, S.: A complete generalization of Atkin's square root algorithm. Fundamenta Informaticae **125**, 71–94 (2013)
18. Pocklington, H.C.: The direct solution of the quadratic and cubic binomial congruences with prime moduli. Proc. Camb. Philos. Soc. **19**, 57–59 (1917)
19. Heo, G., Choi, S., Lee, K.H., Koo, N., Kwon, S.: Remarks on the Pocklington and Padró-Sáez cube root algorithm in $\mathbb{F}_q$. Electron. Lett. **50**, 1002–1003 (2014)
20. Padró, C., Sáez, G.: Taking cube roots in $\mathbb{Z}_m$. Appl. Math. Lett. **15**, 703–708 (2002)
21. Peralta, R.C.: A simple and fast probabilistic algorithm for computing square roots modulo a prime number. IEEE Trans. Inf. Theory **32**, 846–847 (1986)
22. Williams, H. C.: Some algorithm for solving $x^q \equiv N \pmod{p}$, In: Proceedings of 3rd Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic University), pp. 451–462 (1972)
23. Williams, K.S., Hardy, K.: A refinement of H. C. Williams' qth root algorithm. Math. Comput. **61**, 475–483 (1993)
24. Cho, G.H., Koo, N., Ha, E., Kwon, S.: New cube root algorithm based on third order linear recurrence relation in finite field. Des. Codes Cryptogr. **75**(3), 483–495 (2015)
25. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, Cambridge (2003)
26. Kedlaya, K.S., Umans, C.: Fast polynomial factorization and modular composition. SIAM J. Comput. **40**, 1767–1802 (2011)
27. Sze, T.-W.: On taking square roots without quadratic nonresidues over finite fields. Math. Comput. **80**, 1797–1811 (2011)
28. Johnston, A. M.: A Generalized qth Root Algorithm, In: Proceeding of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms. Baltimore, pp. 929–930 (1999)