Bull. Korean Math. Soc. **59** (2022), No. 6, pp. 1523–1537
https://doi.org/10.4134/BKMS.b210870
pISSN: 1015-8634 / eISSN: 2234-3016

# ON THE POCKLINGTON-PERALTA SQUARE ROOT ALGORITHM IN FINITE FIELDS

Chang Heon Kim, Namhun Koo, and Soonhak Kwon

Abstract. We present a new square root algorithm in finite fields which is a variant of the Pocklington-Peralta algorithm. We give the complexity of the proposed algorithm in terms of the number of operations (multiplications) in finite fields, and compare the result with other square root algorithms, the Tonelli-Shanks algorithm, the Cipolla-Lehmer algorithm, and the original Pocklington-Peralta square root algorithm. Both the theoretical estimation and the implementation result imply that our proposed algorithm performs favorably over other existing algorithms. In particular, for the NIST suggested field P-224, we show that our proposed algorithm is significantly faster than other proposed algorithms.

## 1. Introduction

Computing square roots in finite fields is a classical problem in computational number theory. For example, it can be applied in point compression in elliptic curves over finite fields. In this paper, we consider the problem of computing square roots in the finite field $\mathbb{F}_q$ of $q$ elements where $q$ is a power of an odd prime, and we write $q-1=2^s t$, where $t$ is odd and $s>0$. The Tonelli-Shanks algorithm [13, 14] is the most well known square root algorithm, and is efficient for small $s$. However, there are some cases that finite fields with large $s$ are necessary. For example, NIST P-224 elliptic curve [3] is defined over the prime finite field $\mathbb{F}_q$ with $q=2^{224}-2^{96}+1$, where $s=96$ is relatively large to apply Tonelli-Shanks algorithm. In this case, Cipolla-Lehmer algorithm [2, 6] can be an alternative because its complexity does not depend on $s$.

Received December 2, 2021; Revised February 20, 2022; Accepted April 1, 2022.
2020 *Mathematics Subject Classification.* 11T06, 11Y16, 68W40.
*Key words and phrases.* Square root algorithm, finite field, Pocklington-Peralta algorithm, Tonelli-Shanks algorithm, Cipolla-Lehmer algorithm.
This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. 2016R1A5A1008055). Namhun Koo was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1C1C2003888). Soonhak Kwon was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1F1A1058920 and No. 2021R1F1A1050721).

Pocklington [11] proposed another square root algorithm and it was rediscovered by Peralta [10] who was unaware of the work of Pocklington (see also [1]). The complexity of Pocklington-Peralta square root algorithm does not depend on $s$ nor the Hamming weight of $q$, like Cipolla-Lehmer algorithm. In this paper, we propose a refinement of Pocklington-Peralta square root algorithm. We give a complexity analysis about our proposed algorithm so that the complexity of our proposed algorithm does not depend on $s$ and the Hamming weight of $q$. We also compare with several known square root algorithms including the original Pocklington-Peralta square root algorithm by implementation results via SageMath. By implementation results, we can see that our proposed algorithm is at least 1.67 times faster than the original Pocklington-Peralta algorithm.

The remainder of this paper is organized as follows. In Section 2, we review Pocklington-Peralta square root algorithm and several known square root algorithms. In Section 3, we propose our refinement of Pocklington-Peralta square root algorithm. In Section 4, we analyze the theoretical complexity of our proposed algorithm and give the implementation results, using SageMath, of our algorithm and other existing algorithms. Finally, in Section 5, we give a concluding remark.

## 2. Some special cases and Pocklington-Peralta method

Though Pocklington [11] and Peralta considered the finite field $\mathbb{F}_q$ with odd prime $q$, their approaches are also good for the general finite field. Therefore we assume that $q$ is a power of a prime and let $\mathbb{F}_q$ be a finite field with $q$ elements. Let $c \neq 0 \in \mathbb{F}_q$ be a square in $\mathbb{F}_q$, i.e., there exists $x \in \mathbb{F}_q$ such that $x^2 = c$.

Note that if $q \equiv 3 \pmod 4$, a square root of $c \in \mathbb{F}_q$ is given as $c^{\frac{q+1}{4}}$. Also if $q \equiv 5 \pmod 8$, then a square root of $c$ is given as $c^{\frac{q+3}{8}}$ if $c^{\frac{q-1}{4}} = 1$, and is given as $c^{\frac{q+3}{8}}\xi$ if $c^{\frac{q-1}{4}} = -1$, where $\xi^2 = -1$ (i.e., a primitive 4-th root of unity which can be precomputed). In fact, if $q \equiv 2^s + 1 \pmod{2^{s+1}}$, a square root of $c$ is given as $c^{\frac{q+2^s-1}{2^{s+1}}}\xi^u$ for some suitable $0 \leq u < 2^{s-1}$, where $u$ is depending on the (multiplicative) order of $c^{\frac{q-1}{2^s}}$. Note that the condition $q \equiv 2^s + 1 \pmod{2^{s+1}}$ automatically implies that $s$ is the greatest integer satisfying $2^s \mid q - 1$. This approach of finding a square root is only effective when $s$ is small, since it involves a discrete logarithmic problem in the subgroup of $\mathbb{F}_q^\times$ of order $2^s$. For a detailed explanation of above arguments, see [4,5]. From now on, we assume $q \equiv 1 \pmod 4$ (i.e., $s \geq 2$), and will especially focus on the case where $s$ is slightly large, say $s \geq 5$. Since $s$ is the largest power satisfying $2^s \mid q - 1$, we write $q - 1 = 2^s t$ with $t$ odd.

For a given square $c \in \mathbb{F}_q$, $-c$ is also square in $\mathbb{F}_q$ since $q \equiv 1 \pmod 4$, and by letting $x \in \mathbb{F}_q$ be a square root of $-c$, we have the factorizaton $X^2 + c = (X - x)(X + x) \in \mathbb{F}_q[X]$. Now we briefly summarize the Pocklington-Peralta method [10, 11] on square root computation in $\mathbb{F}_q$. One chooses a random $a + X \in \mathbb{F}_q[X]/\langle X^2 + c\rangle$, where $a \in \mathbb{F}_q$ and the ring $\mathbb{F}_q[X]/\langle X^2 + c\rangle$ has

the structure of addition and multiplication modulo $X^2 + c$. Now compute $(a+X)^t = A + BX \in \mathbb{F}_q[X]/\langle X^2 + c\rangle$ and check whether $A$ or $B$ is zero. If so, then choose another $a$ until one has $AB \neq 0$. From the relation $(A + BX)^2 = (A^2 - B^2c) + 2ABX$, one has $(A+BX)^2 \in \mathbb{F}_q$ if and only if $AB = 0$. Therefore, assuming $AB \neq 0$, the condition $(A+BX)^{2^m} \in \mathbb{F}_q$ implies that there is a unique $1 \leq j < m$ such that $(A + BX)^{2^j} \in \mathbb{F}_q \cdot X$, where $\mathbb{F}_q \cdot X = \{bX \mid b \in \mathbb{F}_q\} \subset \mathbb{F}_q[X]/\langle X^2 + c\rangle$. Since $(a + X)^{q-1} = 1$, there exists the least positive integer $1 \leq j \leq s - 1$ such that

$$(a + X)^{t2^j} \in \mathbb{F}_q \cdot X. \tag{1}$$

Then letting $A_1 + B_1X = (a + X)^{t2^{j-1}}$, one has $(a + X)^{t2^j} = (A_1 + B_1X)^2 = (A_1^2 - B_1^2c) + 2A_1B_1X \in \mathbb{F}_q \cdot X$. Therefore one deduces $A_1^2 - B_1^2c = 0$ and gets a square root of $c$ as $c = \left(\frac{A_1}{B_1}\right)^2$.

**Algorithm 1** Pocklington-Peralta Square Root Algorithm (Algorithm 2 in [10])

**Input:** A prime $q \equiv 1 \pmod 4$ and a quadratic residue $c \pmod q$
**Output:** A square root of $c \pmod q$

1: Choose a random $a \pmod q$
2: **if** $a^2 = -c$ **then** go to STEP 1
3: Compute $(a + X)^t = A + BX \pmod{X^2 + c}$
4: **if** either $A = 0$ or $B = 0$ **then** go to STEP 1
5: Compute $(A + BX)^{2^i} \pmod{X^2 + c}$ for $i = 1, 2, \ldots$ by repeated squaring until $(A + BX)^{2^i} = 0 + kX \pmod{X^2 + c}$ for some $k$
6: Let $(A + BX)^{2^{i-1}} = A_1 + B_1X \pmod{X^2 + c}$ (Then $A_1^2 - B_1^2c = 0$)
7: **return** $A_1/B_1 \pmod q$

The complexity of the above Pocklington-Peralta algorithm is $O(\log^3 q)$ but it uses a heavy ring arithmetic in $\mathbb{F}_q[X]/\langle X^2 + c\rangle$. When compared with the Tonelli-Shanks square root algorithm which uses an arithmetic only in $\mathbb{F}_q$, the Pocklington-Peralta algorithm only achieves the computational advantage over the Tonelli-Shanks when $s$ is quite large where $q - 1 = 2^s t$ with $t$ odd. The complexity of the Tonelli-Shanks algorithm is $O(\log^3 q + s^2 \log^2 q)$. Therefore when $s \approx \log q$, it achieves the worst-case complexity $O(\log^4 q)$. More precisely, it is shown in [7,9] that the average number of $\mathbb{F}_q$-multiplications of the Tonelli-Shanks is $2\log q + 2H(q-1) + \frac{s^2}{4} + O(s)$, where $H(q-1)$ is the Hamming weight (i.e., the number of ones in the binary representation) of $q - 1$. To put it simply, based on the square and multiply method for an exponentiation in $\mathbb{F}_q$, the average cost of the Tonelli-Shanks is *two exponentiations plus* $\frac{s^2}{4}$ *multiplications in* $\mathbb{F}_q$, where two exponentiations come from $2(\log q + H(q-1))$.

## 3. New refined algorithm

Since the original version of the Pocklington-Peralta algorithm requires ring multiplications in $\mathbb{F}_q[X]/\langle X^2+c\rangle$ such as

$$\begin{aligned}(a_1+b_1X)(a_2+b_2X) &= (a_1a_2-b_1b_2c)+(a_1b_2+a_2b_1)X,\\ (a+bX)^2 &= a^2-b^2c+2abX,\end{aligned} \tag{2}$$

the cost of such single multiplication in $\mathbb{F}_q[X]/\langle X^2+c\rangle$ is very high compared with the cost of single multiplication in $\mathbb{F}_q$ even if we use some special techniques like Karatsuba type multiplications in the above equations in (2), i.e., $a_1b_2+a_2b_1=(a_1+b_1)(a_2+b_2)-a_1a_2-b_1b_2$.

Our new idea will exploit the properties of ring structure of $\mathbb{F}_q[X]/\langle X^2+c\rangle$ and its related linear recurrences having characteristic polynomials with roots in $\mathbb{F}_q[X]/\langle X^2+c\rangle$. Using linear recurrences, we transform exponentiations in $\mathbb{F}_q[X]/\langle X^2+c\rangle$ into compact $\mathbb{F}_q$-multiplications with reduced cost compared with naive methods in (2) or its Karatsuba type variants. Letting $x\in\mathbb{F}_q$ be a square root of $-c\neq 0$, one has the following isomorphism of rings

$$\mathbb{F}_q[X]/\langle X^2+c\rangle \cong \mathbb{F}_q\times\mathbb{F}_q,$$

where the isomorphism is given as

$$\begin{aligned}\varphi : \mathbb{F}_q[X]/\langle X^2+c\rangle &\longrightarrow \quad \mathbb{F}_q\times\mathbb{F}_q\\ a+bX &\mapsto (a+bx,a-bx).\end{aligned} \tag{3}$$

We define the conjugate of $\theta=a+bX\in\mathbb{F}_q[X]/\langle X^2+c\rangle$ as $\bar{\theta}=a-bX$, and we also define the trace and the norm of $\theta$ as

$$Tr(\theta)=\theta+\bar{\theta}=2a, \qquad N(\theta)=\theta\bar{\theta}=a^2+b^2c,$$

where $Tr(\theta),N(\theta)\in\mathbb{F}_q$. In fact, for a given $\theta=a+bX\in\mathbb{F}_q[X]/\langle X^2+c\rangle$, the trace and the norm of $\theta$ is the usual trace and determinant of the linear transformation $\ell_\theta:\mathbb{F}_q[X]/\langle X^2+c\rangle\longrightarrow\mathbb{F}_q[X]/\langle X^2+c\rangle$ defined by $\ell_\theta(w)=w\theta$.

We denote the set of invertible elements in each field and ring as $\mathbb{F}_q^*$ and $(\mathbb{F}_q[X]/\langle X^2+c\rangle)^*$. Then we have

$$N(\theta)\neq 0 \Longleftrightarrow \varphi(z)\in\mathbb{F}_q^*\times\mathbb{F}_q^*,$$

which implies that we also have the (multiplicative) isomorphism between the sets of invertible elements;

$$\varphi : (\mathbb{F}_q[X]/\langle X^2+c\rangle)^* \cong \mathbb{F}_q^*\times\mathbb{F}_q^*. \tag{4}$$

Recall, from the previous section, that one has $a^2-b^2c=0$ if and only if $\theta^2=(a+bX)^2\in\mathbb{F}_q\cdot X$, which is also equivalent to the condition $Tr(\theta^2)=0$. Therefore the condition of $\theta^{2^jt}\in\mathbb{F}_q\cdot X$ can be detected by checking $Tr(\theta^{2^jt})=0$.

### 3.1. Applying Lucas sequence over the ring $\mathbb{F}_q[X]/\langle X^2 + c\rangle$

Now we define two sequences $\{a_k\}$ and $\{b_k\}$ over $\mathbb{F}_q$ as

$$a_k + b_k X = \theta^k = (a + bX)^k \in \mathbb{F}_q[X]/\langle X^2 + c\rangle, \quad k = 0, 1, 2, \ldots. \tag{5}$$

Our purpose is to compute $a_k$ and $b_k$ as efficiently as possible, where the Lucas type recurrence relations are extensively used. Note that $\theta = a + bX \in \mathbb{F}_q[X]/\langle X^2+c\rangle$ is a root of the polynomial $f(Y) = Y^2 - Tr(\theta)Y + N(\theta) \in \mathbb{F}_q[Y]$. Thus $Tr(\theta^k) = \theta^k + \bar{\theta}^k$ can be computed via the second order linear relation determined by $f(Y)$. It is well known that [8], if $f(Y) = Y^2 - hY + 1$, the computation of the Lucas sequence $V_k$ defined by

$$V_0 = 2, V_1 = h, \qquad V_{k+2} - hV_{k+1} + V_k = 0 \quad (k \geq 0)$$

can be efficiently computed by the relation

$$V_{2k} = V_k^2 - 2, \quad V_{2k+1} = V_k V_{k+1} - h. \tag{6}$$

To use the above mentioned trick, we assume $N(\theta) = a^2 + b^2 c = 1$. We will explain how to choose such $\theta$ in the next subsection. Then, letting $f(Y) = Y^2 - 2aY + 1$, we get $f(\theta) = 0$ and thus we have $Tr(\theta^k) = V_k$ for all $k$, since $Tr(\theta^0) = 2 = V_0, Tr(\theta^1) = 2a = V_1$ and both $Tr(\theta^k)$ and $V_k$ satisfy the same recurrence relation determined by $f(Y)$.

From the expression $\theta^k = a_k + b_k X$ in the equation (5), we get $Tr(\theta^k) = 2a_k$ and thus the sequence $a_k = \frac{1}{2}Tr(\theta^k) = \frac{1}{2}V_k$ can be efficiently computed. Now the question is whether we can apply a similar method to the sequence $b_k$.

We answer this question affirmatively since all three sequences $\{a_k\}$, $\{b_k\}$ and $\{Tr(\theta^k)\}$ are determined by the same characteristic polynomial $f(Y) = Y^2 - 2aY + 1$ with different initial input values. This can be explained as follows. Again from $\theta^k = a_k + b_k X$, we get

$$\begin{aligned} 0 &= \theta^k(\theta^2 - 2a\theta + 1) \\ &= \theta^{k+2} - 2a\theta^{k+1} + \theta^k \\ &= (a_{k+2} - 2aa_{k+1} + a_k) + (b_{k+2} - 2ab_{k+1} + b_k)X, \end{aligned}$$

which implies

$$a_{k+2} - 2aa_{k+1} + a_k = 0, \quad b_{k+2} - 2ab_{k+1} + b_k = 0,$$

and thus $\{a_k\}$ and $\{b_k\}$ satisfy the same recurrence relation determined by $f(Y)$.

Once $V_k = Tr(\theta^k)$ is computed, no extra cost (multiplication or division) is required for $a_k = \frac{1}{2}Tr(\theta^k)$ (half of the Lucas sequence). For $b_k$, the situation is slightly complicated but we want the cost of computing $b_k$ negligible also.

One may use the same trick of using the formula (6) to compute $\{b_k\}$ because $b_k$ satisfies the same recurrence relation but different initial inputs $b_0 = 0$, $b_1 = b$. However, this method again requires the same number of multiplications as the case of $a_k$, which doubles the total computational cost. Our new idea is that the computation of $b_k$ can be accomplished almost freely from the information

of $a_k$ and $a_{k+1}$, so the total cost of computing both $a_k$ and $b_k$ is equal to the cost of evaluating $a_k$ only.

For the time being, assume that the sequence $\{b_k\}$ can be written as

$$b_k = Tr(\beta\theta^k) \quad \text{for some } \beta \in \mathbb{F}_q[X]/\langle X^2 + c\rangle \tag{7}$$

for the time being, and we want to find a (unique) $\beta \in \mathbb{F}_q[X]/\langle X^2+c\rangle$ satisfying the above relation for all $k \geq 0$. Since $\beta \in \mathbb{F}_q[X]/\langle X^2 + c\rangle$, we may also write $\beta = d_0 + d_1\theta$ with $d_0, d_1 \in \mathbb{F}_q$ and the equation (7) is written as

$$\begin{aligned} b_k = Tr(\beta\theta^k) &= Tr((d_0 + d_1\theta)\theta^k) \\ &= d_0 Tr(\theta^k) + d_1 Tr(\theta^{k+1}). \end{aligned} \tag{8}$$

If the above relation (8) holds for $k = 0, 1$, then the relation holds for all $k \geq 0$ since $\{b_k\}$ and $\{Tr(\theta^k)\}$ satisfy the same recurrence relation. Now writing the relation for $k = 0$ and $k = 1$,

$$\begin{aligned} 0 = b_0 &= d_0 Tr(\theta^0) + d_1 Tr(\theta^1) = 2d_0 + 2ad_1, \\ b = b_1 &= d_0 Tr(\theta^1) + d_1 Tr(\theta^2) = 2ad_0 + 2(a^2 - b^2c)d_1, \end{aligned}$$

and solving the above system of linear equations, the solution exists if and only if $bc \neq 0$, and we get

$$d_0 = \frac{a}{2bc}, \quad d_1 = -\frac{1}{2bc}. \tag{9}$$

Therefore, for our initial choice of such $d_0$ and $d_1$, $b_k$ is expressed as

$$b_k = d_0 Tr(\theta^k) + d_1 Tr(\theta^{k+1}) = 2d_0 a_k + 2d_1 a_{k+1} = \frac{aa_k - a_{k+1}}{bc}. \tag{10}$$

The detailed process of computation of $Tr(\theta^k)$ using the idea mentioned above is described in Algorithm 2. This algorithm requires $\log t$ multiplications and $\log t$ squarings for computing $Tr(\theta^t)$.

**Algorithm 2** Lucas Sequence Computation

**Input:** $V_0, V_1$ and $t = \sum_{j=0}^{l-1} t_j 2^j$
**Output:** $Tr(\theta^t), Tr(\theta^{t+1})$

1: $V \leftarrow 2,\ W \leftarrow 2a$ $(V = V_0, W = V_1)$
2: **for** $j$ from $l-1$ down to 0 **do**
3: **if** $t_j = 0$, **then** $W \leftarrow VW - 2a,\ V \leftarrow V^2 - 2$ (i.e., $W = V_{2k+1}, V = V_{2k}$)
4: **if** $t_j = 1$, **then** $V \leftarrow VW - 2a,\ W \leftarrow W^2 - 2$ (i.e., $V = V_{2k+1}, W = V_{2k+2}$)
5: **return** $V, W$

### 3.2. Selecting $\theta \in \mathbb{F}_q[X]/\langle X^2 + c\rangle$ with $N(\theta) = 1$

Define a subgroup $S$ of $(\mathbb{F}_q[X]/\langle X^2 + c\rangle)^*$ as

$$S = \{\theta \in (\mathbb{F}_q[X]/\langle X^2 + c\rangle)^* \mid N(\theta) = 1\}.$$

Then one has

$$S = \{\theta \in (\mathbb{F}_q[X]/\langle X^2 + c\rangle)^* \mid \theta = \frac{a + bX}{a - bX} \text{ with } a^2 + b^2 c \neq 0\}$$

because the condition $N(\theta)=1$ is equivalent to $\theta=\frac{\tau}{\bar{\tau}}$ for some $\tau\in(\mathbb{F}_q[X]/\langle X^2+c\rangle)^*$. This fact is just a simple analogue of Hilbert Theorem 90. More precisely, when $N(a + bX) = a^2 + b^2 c = 1$, then $a + bX$ can be expressed as

$$a + bX = \begin{cases} \dfrac{a + 1 + bX}{a + 1 - bX} & \text{if } a \neq -1, \\ \dfrac{X}{-X} & \text{if } a = -1, \end{cases}$$

since $a = -1$ implies $b = 0$.

We claim that $S$ is isomorphic to $\mathbb{F}_q^*$ via the map $\varphi_0 : S \to \mathbb{F}_q^*$ with $\varphi_0(a + bX) = a + bx$, where the isomorphism $\varphi_0$ is induced from $\varphi$ in the equation (3) restricted on $S$,

$$\begin{array}{ccc} S & \xrightarrow{\varphi} & \mathbb{F}_q^* \times \mathbb{F}_q^* \\ a + bX & \mapsto & (a + bx, a - bx) = (a + bx, \dfrac{1}{a + bx}), \end{array} \tag{11}$$

such that $\varphi(S) = \{(d, \frac{1}{d}) \mid d \in \mathbb{F}_q^*\} \subsetneq \mathbb{F}_q^* \times \mathbb{F}_q^*$. Injectivity of $\varphi_0$ is clear since $\varphi$ is injective. For the surjectivity of $\varphi_0$, given $d \in \mathbb{F}_q^*$, one can solve the system of the equations

$$a + bx = d, \quad a - bx = \frac{1}{d}$$

to find unique $a = \frac{1}{2}(d + \frac{1}{d})$ and $b = \frac{1}{2x}(d - \frac{1}{d})$ satisfying the above relations because $x \neq 0$ (i.e., $c \neq 0$).

### 3.3. Our proposed algorithm

Let $\theta \in S$ and suppose that $\theta^t = d \in \mathbb{F}_q$. Then from $1 = N(\theta)^t = N(\theta^t) = d^2$, one has $d = \pm 1$. Therefore the probability that a randomly chosen $\theta \in S$ satisfies $\theta^t \in \mathbb{F}_q$ is same to the probability that a randomly chosen $\theta \in S$ satisfies $\theta^t = \pm 1$. The condition $\theta^t = \pm 1$ is equivalent to the condition $\theta^{2t} = 1$ because $\theta \in S \cong \mathbb{F}_q^*$. Since there are $2t$ possible such $\theta \in S$ satisfying $\theta^t = \pm 1$, we find that the probability that a randomly chosen $\theta \in S$ satisfies $\theta^t \in \mathbb{F}_q$ is $\frac{2t}{q-1} = \frac{2t}{2^s t} = \frac{1}{2^{s-1}}$.

Now we need to derive the other probabilities in view of the equation (1).

**Proposition 3.1.** *Let $q$ be a prime power with $q - 1 = 2^s t$ and $\gcd(2, t) = 1$. Let $0 \leq m \leq s - 2$ and $P_m$ be the probability that a randomly chosen $\theta \in S$ satisfies $\theta^{2^m t} \in \mathbb{F}_q \cdot X$. Then $P_m = \frac{1}{2^{s-m-1}}$.*

*Proof.* We have to find the probability that $\theta^{2^m t} = hX$ for some $h \in \mathbb{F}_q$. Due to the isomorphism in the equation (4), we may assume $\varphi(\theta) = (d, \frac{1}{d}) \in \mathbb{F}_q^* \times \mathbb{F}_q^*$ and

$$(hx, -hx) = \varphi(\theta^{2^m t}) = \varphi(\theta)^{2^m t} = (d^{2^m t}, \frac{1}{d^{2^m t}}).$$

Thus we get $d^{2^{m+1}t} = -1$ and since there are exactly $2^{m+2}t$ possibilities of $d \in \mathbb{F}_q^* \cong S$ satisfying $d^{2^{m+2}t} = 1$ when $2^{m+2}t \mid q-1$, there are exactly $2^{m+1}t$ possibilities of $d$ satisfying $d^{2^{m+1}t} = -1$ for any $0 \le m \le s-2$. Consequently there are $2^{m+1}t$ possibilities of $\theta \in S$ satisfying $\theta^{2^m t} \in \mathbb{F}_q \cdot X$, and the desired probability $P_m$ is $\frac{2^{m+1}t}{q-1} = \frac{1}{2^{s-m-1}}$. □

*Remark* 3.2. Note that the condition $N(\theta) = 1$ implies that $\theta^{2^{s-1}t} = \pm 1 \in \mathbb{F}_q$. Therefore, in the above proposition, the case $m = s-1$ (i.e., the case $\theta^{2^{s-1}t} \in \mathbb{F}_q \cdot X$) does not happen.

*Remark* 3.3. As a special case, when $m = 0$, we get the probability that $\theta^t = b_t X$ as $\frac{1}{2^{s-1}}$. We also remark that one can find a square root of $c$ if $\theta^t \in \mathbb{F}_q \cdot X$. That is, if $\theta^t = b_t X$ for some $b_t \in \mathbb{F}_q$, then by taking the norm, one has $1 = N(\theta)^t = b_t X \cdot (-b_t X) = b_t^2 c$. Thus $c = (\frac{1}{b_t})^2$. In view of the equations (9), (10), we have $b_t = -\frac{Tr(\theta^{t+1})}{2bc}$ so that we get $c = \left(\frac{2bc}{Tr(\theta^{t+1})}\right)^2$.

Since we already showed that the probability that a randomly chosen $\theta \in S$ satisfies $\theta^t \in \mathbb{F}_q$ is $\frac{1}{2^{s-1}}$ in the statement just before Proposition 3.1, we have total $s$ possible cases ($\theta^t \in \mathbb{F}_q$ or $\theta^{2^m t} \in \mathbb{F}_q \cdot X$ for $m = 0, 1, 2, \ldots, s-2$) and the sum of all these probabilities is

$$\frac{1}{2^{s-1}} + \left(\frac{1}{2^{s-1}} + \frac{1}{2^{s-2}} + \frac{1}{2^{s-3}} + \cdots + \frac{1}{2^2} + \frac{1}{2}\right) = 1$$

as is expected.

Our observations in Proposition 3.1 and Remark 3.3 lead to a new square root algorithm shown in Algorithm 3, whose complexity is $O(\log^3 q)$ where the cost of the algorithm is one exponentiation in $\mathbb{F}_q$, which will be explained in detail in Section 4. In the algorithm, we try random $\theta = a + X \in \mathbb{F}_q[X]/\langle X^2 + c\rangle$ with $N(\theta) = 1$ until we find $\theta$ satisfying $\theta^t \neq \pm 1$, i.e., $V \neq \pm 2$ at the end of STEP 8. Then, we apply repeated squaring to $\theta^t$ until we have $\theta^{2^m t} \in \mathbb{F}_q \cdot X$ for some $0 \le m \le s-2$. The condition $\theta^{2^m t} \in \mathbb{F}_q \cdot X$ is equivalent to the condition $V = 0$ in the Algorithm 3 and is being checked in STEPS 10-14. STEP 10 is the case $\theta^t \in \mathbb{F}_q \cdot X$ (i.e., $m = 0$) where the square root $\frac{2bc}{W}$ is obtained from the Remark 3.3. The while loop (STEPS 11-14) is terminated after $m$-iterations when $\theta^{2^m t} \in \mathbb{F}_q \cdot X$. From the Remark 1, we know that such $m$ with $0 \le m \le s-2$ always exists. The final output $x$ in STEP 15 is obtained using the equations (9), (10).

**Algorithm 3** Proposed Pocklington-Peralta Algorithm

**Input:** A square $c$ in $\mathbb{F}_q$ where $q-1=2^s t$ with $q \equiv 1 \pmod 4$, $\gcd(2,t)=1$, and a binary representation of $t=\sum_{j=0}^{n-1} t_j 2^j$ with $t_{n-1}=1$

**Output:** $x$ satisfying $x^2=c$ in $\mathbb{F}_q$

1: Choose random $a \in \mathbb{F}_q$ and let $\theta = a+X \in \mathbb{F}_q[X]/\langle X^2+c\rangle$
2: **if** $N(\theta)=0$ **then** go to STEP 1 # if a square root of $-1$ is precomputed, then we are done
3: $a+bX \leftarrow \dfrac{a+X}{a-X}$ # $\dfrac{a+X}{a-X}=\dfrac{a^2-c+2aX}{a^2+c}$
4: $V \leftarrow 2a, W \leftarrow V^2-2$ # $V=Tr(\theta^1), W=Tr(\theta^2)$ where $\theta=a+bX$
5: **for** $j$ from $n-1$ down to 0 **do**
6: **if** $t_j=0$, **then** $W \leftarrow VW-2a, V \leftarrow V^2-2$ # $W=V_{2j+1}, V=V_{2j}$
7: **else then** $V \leftarrow VW-2a, W \leftarrow W^2-2$ # $V=V_{2j+1}, W=V_{2j+2}$
8: **end for** # $V=Tr(\theta^t), W=Tr(\theta^{t+1})$ where $\theta^t=a_t+b_tX$
9: **if** $V=\pm 2$, **then** go to STEP 1 # This is the case $\theta^t \in \mathbb{F}_q$
10: **else if** $V=0$, **then** $x \leftarrow \dfrac{2bc}{W}$ and **return** $x$ # This is the case $\theta^t \in \mathbb{F}_q \cdot X$
11: **while** $V \neq 0$ **do**
12: $W_0 \leftarrow W, V_0 \leftarrow V$
13: $W \leftarrow V_0W_0-2a, V \leftarrow V_0^2-2$ # $V=Tr(\theta^{t2^j}), W=Tr(\theta^{t2^j+1})$
14: **end while**
15: $x \leftarrow \dfrac{V_0bc}{aV_0-W_0}$ and **return** $x$

## 4. Complexity analysis and comparison

Because of Proposition 3.1 (see also Remark 3.3), the probability of having $\theta^t \in \mathbb{F}_q \cdot X$ in STEP 10 is $\frac{1}{2^{s-1}}$, and the probability of having $\theta^{2^m t} \in \mathbb{F}_q \cdot X$ exactly immediately after $m$-th iteration of the while-loop (STEPS 11-14) is $\frac{1}{2^{s-m-1}}$ for $1 \le m \le s-2$. Therefore the expected number of iterations of the while-loop is

$$\begin{aligned}\sum_{m=1}^{s-2} m \cdot \frac{1}{2^{s-m-1}} &= \frac{1}{2^{s-2}}+\frac{2}{2^{s-3}}+\cdots+\frac{s-3}{2^2}+\frac{s-2}{2} \\ &= s-3+\frac{1}{2^{s-2}} \qquad (s \ge 3)\end{aligned} \tag{12}$$

and the average cost of the STEPS 11-14 is

$$2(s-3+\frac{1}{2^{s-2}}) \approx 2(s-3) \tag{13}$$

$\mathbb{F}_q$-multiplications, i.e., at each iteration, one multiplication and one squaring is executed.

On the other hand, the cost of the for-loop (STEPS 5-8) is

$$2n \leq 2(\lfloor \log t \rfloor + 1) \approx 2 \log t = 2 \log \frac{q-1}{2^s} \approx 2 \log q - 2s \tag{14}$$

$\mathbb{F}_q$-multiplications. Since the probability that one has $V \neq \pm 2$ (i.e., $\theta^t \notin \mathbb{F}_q$) at the end of the for-loop is $1 - \frac{1}{2^{s-1}}$, the expected number of the iterations of the for-loop is $\frac{2^{s-1}}{2^{s-1}-1} \approx 1$.

For other steps, we require 4 multiplications and one inversion in $\mathbb{F}_q$ in STEP 3, and 3 multiplications and one inversion in $\mathbb{F}_q$ for STEP 15. (Note that we do not have to consider the cost of STEP 10 because it is vastly dominated by the cost of STEPS 11-15.)

Consequently, combining the expression (13) and (14), the total number of necessary $\mathbb{F}_q$-multiplications (except two inversions in $\mathbb{F}_q$) of the proposed Algorithm when $s$ is slightly large (say $s \geq 4$) is

$$2 \log q - 2s + 2(s-3) + 7 \approx 2 \log q,$$

which is the cost of just ONE exponentiation in $\mathbb{F}_q$ and independent of $s$.

In Section 2, we mentioned that the number of the necessary multiplications of the Tonelli-Shanks algorithm is $2 \log q + 2H(q-1) + \frac{s^2}{4} + O(s)$. Comparing with our proposed Pocklington-Peralta algorithm, the Tonelli-Shanks requires $2H(q-1) + \frac{s^2}{4}$ extra multiplications in $\mathbb{F}_q$ (if we ignore the cost of two $\mathbb{F}_q$-inversions of our proposed algorithm).

We compared our proposed algorithm with several well-known square root algorithms in the finite field $\mathbb{F}_q$; the Tonelli-Shanks algorithm [13,14], the Cipolla-Lehmer algorithm [2, 6] and original Pocklington-Peralta algorithm [10, 11].

For convenience, we used prime fields $\mathbb{F}_q$ with 2000 bits primes. Tables 1, 2 and 3 show the implementation results with SAGE of the above mentioned three algorithms and our proposed algorithm for the Hamming weight of $q(=H(q))$ by $< 10$, $\approx 300$, $\approx 1000$, respectively. The implementation was performed on Intel Core i7-4770 3.40GHz with 8GB memory. Average timings of the square root computations for 100 different inputs of quadratic residue $a \in \mathbb{F}_q$ are computed for those cases $s \approx 5, 10, 50, 100, 200, 300$ etc. When we choose each prime, we first choose a random integer $p_0$ such that $2^s \mid (p_0 - 1)$ and $p_0$ has Hamming weight satisfying given condition. Then we add $2^s$ to $p_0$ until it is a prime. If the Hamming weight of chosen prime is significantly different from given condition, we try again until satisfying it.

The second row of each table contains the results of revised Tonelli-Shanks algorithm when a quadratic nonresidue (that is randomly chosen in the original Tonelli-Shanks algorithm) is given and hence requires only one exponentiation. The fifth row of each Table gives the results to run *square_root_mod_prime*$(a, q)$ built in SAGE program [12]. As one can see in the tables, the timings of the Tonelli-Shanks algorithm increase drastically as $s$ becomes larger, while the timings of other existing algorithms and our algorithm are independent of $s$. Also, the timings of our proposed algorithm show that our proposed algorithm

TABLE 1. Running time (in ms) for square root computation with $q \approx 2^{2000}$ and $H(q) < 10$

| $s \approx$ | 5 | 10 | 50 | 100 | 200 | 300 |
|---|---|---|---|---|---|---|
| $H(q)$ | 7 | 9 | 9 | 7 | 7 | 9 |
| Tonelli-Shanks [13, 14] | 8.259 | 8.291 | 9.484 | 13.296 | 27.679 | 52.076 |
| Tonelli-Shanks (precom.) [13, 14] | 4.058 | 4.120 | 5.371 | 9.406 | 23.844 | 48.227 |
| SageMath [12] | 5.070 | 5.095 | 6.988 | 13.004 | 36.508 | 74.398 |
| Cipolla-Lehmer [2, 6] | 14.709 | 14.660 | 14.666 | 14.692 | 14.675 | 14.663 |
| original Pocklington-Peralta [10, 11] | 14.937 | 14.671 | 14.657 | 14.901 | 14.669 | 14.672 |
| Proposed Alg. | 8.963 | 8.803 | 8.782 | 8.879 | 8.799 | 8.822 |

TABLE 2. Running time (in ms) for square root computation with $q \approx 2^{2000}$ and $H(q) \approx 300$

| $s \approx$ | 5 | 10 | 50 | 100 | 200 | 300 |
|---|---|---|---|---|---|---|
| $H(q)$ | 299 | 305 | 299 | 300 | 305 | 303 |
| Tonelli-Shanks [13, 14] | 9.604 | 9.804 | 11.038 | 14.555 | 30.013 | 54.436 |
| Tonelli-Shanks (precom.) [13, 14] | 4.732 | 4.837 | 6.221 | 9.928 | 25.039 | 50.770 |
| SageMath [12] | 5.474 | 5.582 | 7.738 | 13.188 | 37.297 | 75.737 |
| Cipolla-Lehmer [2, 6] | 15.697 | 15.794 | 15.890 | 15.772 | 15.810 | 15.865 |
| original Pocklington-Peralta [10, 11] | 15.993 | 15.785 | 15.761 | 15.811 | 15.809 | 15.866 |
| Proposed Alg. | 9.103 | 8.989 | 8.991 | 9.012 | 8.989 | 9.040 |

TABLE 3. Running time (in ms) for square root computation with $q \approx 2^{2000}$ and $H(q) \approx 1000$

| $s \approx$ | 5 | 10 | 50 | 100 | 200 | 300 |
|---|---|---|---|---|---|---|
| $H(q)$ | 1002 | 998 | 1002 | 999 | 1000 | 1001 |
| Tonelli-Shanks [13, 14] | 12.157 | 12.638 | 13.706 | 17.498 | 32.029 | 56.054 |
| Tonelli-Shanks (precom.) [13, 14] | 6.045 | 6.295 | 7.526 | 11.355 | 26.640 | 49.815 |
| SageMath [12] | 5.779 | 5.883 | 7.691 | 13.470 | 36.371 | 74.815 |
| Cipolla-Lehmer [2, 6] | 17.244 | 17.630 | 17.697 | 17.745 | 17.715 | 17.652 |
| original Pocklington-Peralta [10, 11] | 17.779 | 17.759 | 17.628 | 17.752 | 17.758 | 17.631 |
| Proposed Alg. | 8.982 | 8.944 | 8.968 | 9.013 | 8.979 | 8.940 |

does not depend on the Hamming weight of $q$, but the timings of the other algorithms increase as the Hamming weight of $q$ higher. The tables also show that our proposed algorithm is consistently faster than Cipolla-Lehmer algorithm and the original Pocklington-Peralta algorithm. In particular, the average timings $8.84, 9.02, 8.97$(ms) of the proposed algorithm are about $1.67, 1.76, 1.98$ times faster than the average timing $14.75, 15.84, 17.72$(ms) of the original Pocklington-Peralta algorithm when $HW(q) < 10$ or when $HW(q) \approx 300, 1000$, respectively.

TABLE 4. Running time (in ms) for square root computation with NIST curves P-224 and P-521 over $\mathbb{F}_q$ with $q = 2^{224} - 2^{96} + 1$, $2^{521} - 1$ (*For P-521, since* $q \equiv 3 \pmod 4$ *and thus* $s = 1$, *a square root of* $c$ *is* $c^{\frac{q+1}{4}}$, *which is used in SageMath*).

| $q$ | P-224 | P-521 |
|---|---|---|
| Tonelli-Shanks [13, 14] | 1.212 | 1.130 |
| Tonelli-Shanks (precom.) [13, 14] | 1.031 | 0.511 |
| SageMath [12] | 0.695 | 0.071 |
| Cipolla-Lehmer [2, 6] | 0.333 | 1.393 |
| original Pocklington-Peralta [10, 11] | 0.319 | 2.282 |
| Proposed Alg. | 0.176 | 1.351 |

In Table 4, we show the implementation results of some practical cases for the NIST suggested fields $\mathbb{F}_q$ with $q = 2^{224} - 2^{96} + 1$ and $q = 2^{521} - 1$, i.e., for P-224 and P-521. When $q = 2^{224} - 2^{96} + 1$, we get $s = 96$ and $HW(q-1) = 128$ and hence both $s$ and $HW(q-1)$ are relatively large. As is already explained in our complexity estimation, we see that our proposed algorithm is indeed quite efficient than other existing algorithms in this case of P-224. When $q = 2^{521} - 1$, one has $s = 1$ and $HW(q-1) = 520$. This is the case of $q \equiv 3 \pmod 4$ and, as is mentioned in Section 2, a square root of $c$ is given as $c^{\frac{q+1}{4}} \in \mathbb{F}_q$. Since SageMath uses special methods (such as $c^{\frac{q+1}{4}}$ when $s = 1$) to compute a square root when $s$ is very small, the running time of SageMath is the shortest in Table 4 for the field P-521.

Figures 1, 2 and 3 are the graphs of timings of Tables 1, 2 and 3, respectively. In our implementation results, the timings of the original Pocklington-Peralta are roughly the same with those of the Cipolla-Lehmer, so the graphs of the two algorithms almost overlap with each other and one sees only 5 graphs (instead of 6) in each figures.

## 5. Conclusion

We proposed a new square root algorithm in $\mathbb{F}_q$ which is a refinement of Pocklington-Peralta algorithm [10,11]. By using linear recurrences arising from quadratic (but reducible over $\mathbb{F}_q$) polynomials, we showed that the cost of our algorithm is essentially one exponentiation in $\mathbb{F}_q$, that is, our algorithm needs $2 \log q$ multiplications in $\mathbb{F}_q$. Our algorithm does not depend on $s$ nor on the Hamming weight of $q$, while the Tonelli-Shanks algorithm depends on both $s$ and $HW(q-1)$.

By the implementation via SageMath, we showed that, even for the medium size NIST suggested field $\mathbb{F}_q$ with $q = 2^{224} - 2^{96} + 1$, our proposed algorithm is the fastest one when compared with other algorithms. Implementation over general finite fields imply that our proposed algorithm is at least 1.67 times

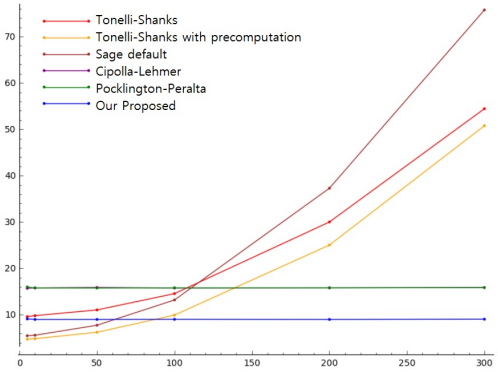


FIGURE 1. Running time (in ms) for square root computation $q \approx 2^{2000}$ and $H(q) < 10$

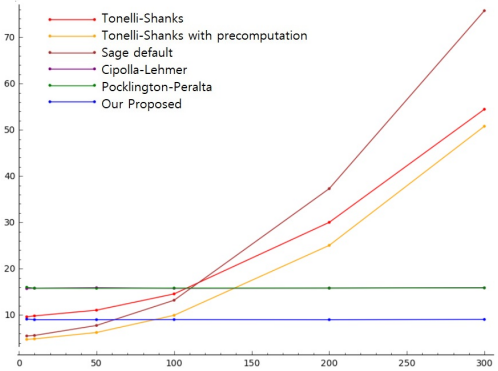


FIGURE 2. Running time (in ms) for square root computation $q \approx 2^{2000}$ and $H(q) \approx 300$

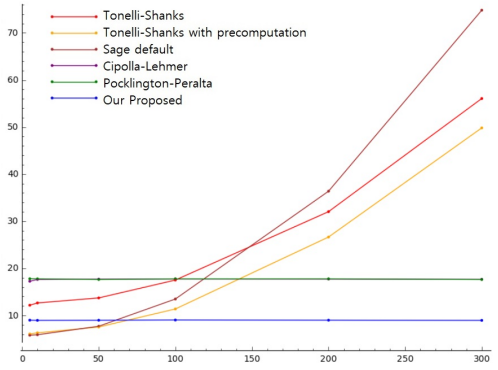


FIGURE 3. Running time (in ms) for square root computation $q \approx 2^{2000}$ and $H(q) \approx 1000$

faster than the original Pocklington-Peralta square root algorithm. Our implementation also shows that the proposed algorithm is superior to the Tonelli-Shanks algorithm when $s > 10$.

## References


[1] D. Bernstein, *Faster square roots in annoying finite fields*, preprint, available at http://cr.yp.to/papers/ sqroot.pdf

[2] M. Cipolla, *Un metodo per la risolutione della congruenza di secondo grado*, Rendiconto dell'Accademia Scienze Fisiche e Matematiche, Napoli, Ser. 3, vol. IX, pp. 154–163, 1903.

[3] Digital Signature Standard(DSS), *Federal information processing standards publication 186–4*, Information Technology Laboratory, National Institute of Standards and Technology, 2013. http://doi.org/10.6028/NIST.FIPS.186-4

[4] N. Koo, G. H. Cho, and S. Kwon, *Square root algorithm in* $\mathbb{F}_q$ *for* $q \equiv 2^s+1 \pmod{2^{s+1}}$, Electronics Letters **49** (2013), no. 7, 467–468. https://doi.org/10.1049/el.2012.4239

[5] N. Koo, G. H. Cho, and S. Kwon, *On r-th root extraction algorithm in* $\mathbb{F}_q$ *for* $q \equiv lr^s+1 \pmod{r^{s+1}}$ *with* $0 < \ell < r$ *and small* $s$, IEEE Trans. Comput. **65** (2016), no. 1, 322–325. https://doi.org/10.1109/TC.2015.2417562

[6] D. H. Lehmer, *Computer technology applied to the theory of numbers*, in Studies in Number Theory, 117–151, Math. Assoc. America, Buffalo, NY, 1969.

[7] S. Lindhurst, *An analysis of Shanks's algorithm for computing square roots in finite fields*, in Number theory (Ottawa, ON, 1996), 231–242, CRM Proc. Lecture Notes, 19, Amer. Math. Soc., Providence, RI, 1999. https://doi.org/10.1090/crmp/019/21

[8] S. Müller, *On the computation of square roots in finite fields*, Des. Codes Cryptogr. **31** (2004), no. 3, 301–312. https://doi.org/10.1023/B:DESI.0000015890.44831.e2

[9] I. Niven, H. S. Zuckerman, and H. L. Montgomery, *An Introduction to the Theory of Numbers*, fifth edition, John Wiley & Sons, Inc., New York, 1991.

[10] R. C. Peralta, *A simple and fast probabilistic algorithm for computing square roots modulo a prime number*, IEEE Trans. Inform. Theory **32** (1986), no. 6, 846–847. https://doi.org/10.1109/TIT.1986.1057236

[11] H. C. Pocklington, *The direct solution of the quadratic and cubic binomial congruences with prime moduli*, Proceedings of the Cambridge Philosophical Society, vol. 19, pp. 57–59, 1917.

[12] Sage Reference Manual, *Elements of* $\mathbb{Z}/n\mathbb{Z}$, available at http://doc.sagemath.org/html/en/reference/finite_rings/sage/rings/finite_rings/integer_mod.html.

[13] D. Shanks, *Five number-theoretic algorithms*, in Proceedings of the Second Manitoba Conference on Numerical Mathematics (Univ. Manitoba, Winnipeg, Man., 1972), 51–70. Congressus Numerantium, VII, Utilitas Math., Winnipeg, MB, 1973.

[14] A. Tonelli, *Bemerkung über die Auflösung Quadratischer Congruenzen*, Göttinger Nachrichten, pp. 344–346, 1891.



Chang Heon Kim
Applied Algebra & Optimization Research Center
Sungkyunkwan University
Suwon 16419, Korea
*Email address*: chhkim@skku.edu

Namhun Koo
Institute of Mathematical Sciences
Ewha Womans University
Seoul 03760, Korea
*Email address*: nhkoo@ewha.ac.kr

Soonhak Kwon
Applied Algebra & Optimization Research Center
Sungkyunkwan University
Suwon 16419, Korea
*Email address*: shkwon@skku.edu