Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 13, Number 4 (2017), pp. 1223–1239

http://www.ripublication.com/gjpam.htm

An Algorithm to Find Square Roots of Quadratic Residues Modulo p (p being an odd prime), $p \equiv 1 \pmod 4$

A. Uma Maheswari
Associate Professor & Head, Department of Mathematics,
Quaid-E-Millath Government College for Women (Autonomous)
Chennai - 600 002, India.

Prabha Durairaj
Assistant Professor, Department of Mathematics,
Quaid-E-Millath Government College for Women (Autonomous)
Chennai - 600 002, India.

Abstract

This paper proposes an algorithm to find square roots of quadratic residues modulo p, where p is an odd prime, $p \equiv 1 \pmod 4$. Quadratic residues in the finite field F_p^* are classified into three categories. This square root algorithm gives a deterministic formula for the first category, an explicit formula which requires the use of a non residue for the second category and a formula, which requires a pre-computed table for the third category. Numerical illustrations for various cases presented in the algorithm are given. A comparative study of this newly developed algorithm with the existing standard Tonelli and Shanks algorithm is done. It is interesting to note that this newly developed algorithm is more efficient than the existing algorithms.

AMS subject classification: 11Z05, 11T71.
Keywords: Quadratic residue modulo p, non-residue, Legendre symbol, square roots modulo p, Tonelli and Shanks Algorithm.

1. Introduction

The square root computation plays a pivotal role in public key cryptosystems like the Rabin Cryptosystem [1], Elliptic curve and Hyper elliptic curve cryptosystems. Algorithms for finding points on an elliptic curve E defined over a finite field F_p use the square root computation. If a is a quadratic residue in the finite field F_p^*, p prime, the square root of a is the solution of the quadratic congruence $x^2 \equiv a \pmod{p}$. This problem has an easy solution when $p \equiv 3 \pmod{4}$ namely $x \equiv \pm\, a^{\left(\frac{p+1}{4}\right)} \pmod{p}$. For the remaining primes $p \equiv 1 \pmod{4}$ there are explicit solutions when $p \equiv 5 \pmod{8}$ [2].

But the case $p \equiv 1 \pmod{8}$ is non-trivial. There are many probabilistic algorithms to solve this problem. In 1891, Tonelli [3] published an algorithm to find square roots molulo p, p prime. This was followed by Cipolla's algorithm [4] in 1903. This is a randomized algorithm and produces a special quadratic polynomial that allows square roots to be calculated in F_{p^2} and later evolved, in 1969 as the Cipolla-Lehmer [5] algorithm. In 1972, Daniel Shanks improved upon Tonelli's algorithm and came forth with the efficient Tonelli-Shanks [6] algorithm. This popular algorithm uses the group structure of F_p^* to inductively find better and better approximation to the square root. The algorithm of Adleman-Manders-Miller [7] was published in 1977. In 1980, M.O. Rabin, using Berlekamp's [8] work on polynomial factoring created the Berlekamp-Rabin [9] algorithm. A similar work of Peralta [10] followed in 1986.

These algorithms are efficient but probabilistic since they require a quadratic non-residue for their implementation. Other related works are found in [11, 12, 13, 14, 15].

In 1985, Schoof [16] used elliptic curves to propose a deterministic algorithm to find square roots modulo p, p prime. This algorithm is efficient (polynomial time) for some residues but not in general. In 2011, Tsz-Wo Sze [17] proposed a deterministic algorithm to find square roots over finite fields without being given any quadratic non-residue.

In this paper, a new algorithm to find square roots of quadratic residues modulo p (where p is an odd prime, $p \equiv 1 \pmod{4}$) is presented. This algorithm gives 3 formulae to find square roots of residues under three mutually exclusive categories i.e., a deterministic formula, which does not need any pre-computation, an explicit formula, which needs the use of a non-residue in F_p^* and a formula which requires the use of a non-residue and also a pre-computed table.

In section 2, we review the Tonelli and Shanks algorithm. Section 3 presents the theorem that forms the basis for the new square root computation together with numerical examples. Section 4 presents the new square root algorithm based on the formula derived in section 3. Section 5 gives the performance of the new algorithm using estimation of the computational complexity and a comparison with the existing algorithms.

2. Preliminary Concepts

In this section, we present the widely used Tonelli and Shanks Algorithm. This well-known algorithm computes the square root of a quadratic residue a modulo p, where p is any odd prime.

2.1. Tonelli and Shanks Algorithm [6, 18]

If $(p-1) = 2^e r$, r odd, this algorithm finds a generator z of the 2-sylow subgroup G (of order 2^e) of F_p^* as $z = n^r \mod p$, where n is any non-residue mod p. Observing that $b = a^r \bmod p$ is a square in G so that its inverse z^k (where k is an even integer, $0 \le k \le 2^e$) satisfies $a^r z^k = 1$ in G, this algorithm finds the square root x of a as $x \equiv a^{\frac{r+1}{2}} z^{\frac{k}{2}} \pmod p$ (so that $x^2 \equiv a (\mod p)$).

Finding the exponent k is more difficult (only $a^{\frac{r+1}{2}} z^{\frac{k}{2}}$ is needed). This is explained in the following algorithm, due to Shanks.

Input: A prime p and an integer a such that $\left(\frac{a}{p}\right) = 1$

Output: An integer x such that $x^2 \equiv a \pmod p$

1. Write $p-1 = 2^e r$ with r odd
2. Choose n at random such that $\left(\frac{n}{p}\right) = -1$
3. $z \leftarrow n^r \bmod p$, $y \leftarrow z$, $s \leftarrow e$ and $x \leftarrow a^{\left(\frac{r-1}{2}\right)} \bmod p$
4. $b \leftarrow (ax^2) \bmod p$ and $x \leftarrow (ax) \bmod p$
5. While $b \not\equiv 1 \pmod p$
6. $\quad m \leftarrow 1$
7. $\quad$ While $b^{2^m} \not\equiv 1 \pmod p$ do $m \leftarrow m+1$
8. $\quad t \leftarrow y^{2^{(s-m-1)}} \bmod p$, $y \leftarrow t^2 \bmod p$ and $s \leftarrow m$
9. $\quad x \leftarrow (tx) \bmod p$ and $b \leftarrow (yb) \bmod p$
10. return x

We note that at the beginning of step 5 we always have the congruences modulo p: $ab \equiv x^2$, $y^{2^{(s-1)}} \equiv -1$, $b^{2^{(s-1)}} \equiv 1$.

If G_s is the subgroup G whose elements have an order dividing 2^s, then this means that y is a generator of G_s and that b is in G_{s-1}. (i.e) b is a square in G_s. Since s is strictly decreasing at each loop of the algorithm, the number of loops is atmost e. When $s \le 1$ we have b = 1 and hence the algorithm terminates.

A variant of the Tonelli and Shanks Algorithm due to Koblitz [19], finds $\frac{k}{2}$ by a bit by bit approach.

3. The New Square Root Computation Theorem

This section presents a theorem that forms a basis for the new square root computation. The computational procedure is illustrated with examples.

Let p be an odd prime, $p \equiv 1 (\mod 4)$.

Let $(p-1) = 2^e r$, r odd. Let $a \in F_p^*$ be a quadratic residue. Then a satisfies exactly one of the following 3 mutually exclusive conditions:

(i) $a^{\left(\frac{p-1}{2^e}\right)} \equiv 1 \pmod p$

(ii) $a^{\left(\frac{p-1}{2^e}\right)} \equiv -1 \pmod{p}$

(iii) $a^{\frac{p-1}{2^{(e-k)}}} \equiv -1 \pmod{p}$ for some k, $1 \le k \le (e-2)$

Under these conditions, we prove the following theorem, which gives the formulae to find square roots of quadratic residues modulo p.

Theorem 3.1. Let p be an odd prime, $p \equiv 1(mod 4)$. Let a be a quadratic residue in the finite field F_p^*. Let $(p-1) = 2^e\, r$, r odd. Then the solution of the quadratic congruence $x^2 \equiv a \pmod{p}$ can be expressed as follows:

(i) $x \equiv \pm\, a^{\left(\frac{p+2^e-1}{2^{e+1}}\right)} \pmod{p}$, if $a^{\left(\frac{p-1}{2^e}\right)} \equiv 1 \pmod{p}$

(ii) $x \equiv \pm \left(n^{\left(\frac{p-1}{4}\right)} a^{\left(\frac{p+2^e-1}{2^{e+1}}\right)}\right) \pmod{p}$, if $a^{\left(\frac{p-1}{2^e}\right)} \equiv -1 \pmod{p}$ where n is any non-residue in F_p^*.

(iii) $x \equiv \pm\, b^{\left[\frac{(2^k-1)(p-1)}{2^{k+2}}\right]} a^{\left(\frac{p+2^e-1}{2^{e+1}}\right)} \pmod{p}$, if $a^{\left(\frac{p-1}{2^{e-k}}\right)} \equiv -1 \pmod{p}$ for some integer k, $1 \le k \le (e-2)$, where b is a non-residue in F_p^* such that $a^{\left(\frac{p-1}{2^e}\right)} \equiv -b^{\left(\frac{p-1}{2^{k+1}}\right)} \pmod{p}$.

Proof. Let a be a quadratic residue in F_p^*.
Then $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$ or briefly, $a^{\frac{p-1}{2}} = 1$ in F_p^*
$a^{\frac{p-1}{2}} = 1 \Rightarrow a^{\frac{p-1}{4}} = \pm 1$ in F_p^*
$a^{\frac{p-1}{4}} = 1 \Rightarrow a^{\frac{p-1}{8}} = \pm 1$ and so on.
In general $a^{\frac{p-1}{2^m}} = 1 \Rightarrow a^{\frac{p-1}{2^{m+1}}} = \pm 1$ in F_p^*, m = 1, 2, ..., (e−1)

Case (i)
Suppose $a^{\frac{p-1}{2^e}} = 1$ in F_p^*
This implies, $a^{\left(\frac{p-1}{2^e}\right)2^j} = 1$, j = 1, 2, …, (e−2)
i.e., $a^{\left(\frac{p-1}{2^{(e-j)}}\right)} = 1$, j = 1, 2, …, (e−2)
Hence case (i) covers those residues $a \in F_p^*$ simultaneously satisfying all the conditions,
$a^{\frac{p-1}{4}} = 1, a^{\frac{p-1}{8}} = 1, \ldots, a^{\frac{p-1}{2^e}} = 1$ **in** F_p^*.

We claim that

$$x \equiv \pm\left(a^{\frac{p+2^e-1}{2^{e+1}}}\right) \bmod p \tag{3.1}$$

is the square root of a, because in F_p^*,

$x^2 = a^{\left(\frac{p+2^e-1}{2^e}\right)} = a^{\frac{p-1}{2^e}}.\ a = a$ (since $a^{\frac{p-1}{2^e}} = 1$ in F_p^*)

Case (ii)

Suppose $a^{\left(\frac{p-1}{2^e}\right)} \equiv -1 \bmod p$

We claim that

$$x \equiv \pm\left[n^{\left(\frac{p-1}{4}\right)} a^{\left(\frac{p+2^e-1}{2^{e+1}}\right)}\right] \pmod p \tag{3.2}$$

is the square root of a, because, in F_p^*

$$\begin{aligned} x^2 &= n^{\left(\frac{p-1}{2}\right)} a^{\left(\frac{p+2^e-1}{2^e}\right)} = (-1)a^{\left(\frac{p-1}{2^e}\right)}.a \quad (\text{since n is a non}-\text{residue},\ \ n^{\frac{p-1}{2}} = -1) \\ &= (-1)(-1)a = a \quad (\text{since } \ a^{\left(\frac{p-1}{2^e}\right)} = -1) \end{aligned}$$

Case (iii)

Suppose $a^{\left(\frac{p-1}{2^e}\right)} \not\equiv \pm 1 \pmod p$. Then, $a^{\left(\frac{p-1}{2^{e-k}}\right)} \equiv -1 \bmod p$, for some integer k, $1 \le k \le (e-2)$. (i.e) $a^{\left(\frac{p-1}{2^{e-k}}\right)} = -1$ in F_p^*.

This implies that $a^{\left(\frac{p-1}{2^{e-k}}\right)} = b^{\frac{p-1}{2}}$ where b is any non-residue in F_p^*. Then successively taking square root, we have,

$a^{\left(\frac{p-1}{2^{e-k+1}}\right)} = \pm b^{\frac{p-1}{4}}$, where $\pm b^{\frac{p-1}{4}}$ are primitive fourth roots of unity in F_p^*

$a^{\left(\frac{p-1}{2^{e-k+2}}\right)} = \pm b^{\frac{p-1}{8}}$, where $\pm b^{\frac{p-1}{8}}$ are primitive 8-th roots of unity and so on.

In general,

$a^{\left(\frac{p-1}{2^{e-k+m}}\right)} = \pm b^{\left(\frac{p-1}{2^{m+1}}\right)}$, where $\pm b^{\left(\frac{p-1}{2^{m+1}}\right)}$ are primitive $2^{(m+1)}$-th roots of unity in F_p^*, $m = 1, 2, \ldots, k$.

We are assured of the existence of these primitive 2^k-th roots of unity in F_p^* (for $k \le e$) since 2^k divides $(p-1)$ when $k \le e$ (If n divides $(q-1)$ then F_q^* has a primitive n-th root of unity).

Choosing a non-residue b such that

$a^{\left(\frac{p-1}{2^e}\right)} \equiv -b^{\left(\frac{p-1}{2^{k+1}}\right)} \bmod p$, we claim that

$$x \equiv \pm\, b^{\left(\frac{(2^k-1)(p-1)}{2^{k+2}}\right)} a^{\left(\frac{p+2^e-1}{2^{e+1}}\right)} \bmod p \tag{3.3}$$

is the square root of a, because in F_p^*

$$\begin{aligned} x^2 &= b^{\left(\frac{(2^k-1)(p-1)}{2^{k+1}}\right)} a^{\left(\frac{p+2^e-1}{2^e}\right)} = b^{\left(\frac{(2^k-1)(p-1)}{2^{k+1}}\right)} a^{\left(\frac{p-1}{2^e}\right)}.a \\ &= b^{\left(\frac{(2^k-1)(p-1)}{2^{k+1}}\right)} \left(-b^{\left(\frac{p-1}{2^{k+1}}\right)}\right) a \quad (\text{since } \ a^{\left(\frac{p-1}{2^e}\right)} = -b^{\left(\frac{p-1}{2^{k+1}}\right)}) \\ &= -b^{\frac{p-1}{2}}.a = a \qquad (\text{since } b^{\frac{p-1}{2}} = -1) \end{aligned}$$

Hence the theorem. ■

Remark 3.2. The statements (i), (ii) and (iii) of Theorem 3.1 ((i.e) equations (3.1), (3.2), (3.3)) actually give the formula for solving the quadratic congruence $x^2 \equiv a(\text{mod p})$(i.e) for computation of square roots in different cases. Let us denote the statements by

Formula (1)

$$x \equiv \pm\, a^{\left(\frac{p+2^e-1}{2^{e+1}}\right)} \text{ (mod p)}, \quad \text{if } a^{\left(\frac{p-1}{2^e}\right)} \equiv 1 \text{ (mod p)}$$

Formula (2)

$x \equiv \pm \left(n^{\left(\frac{p-1}{4}\right)} a^{\left(\frac{p+2^e-1}{2^{e+1}}\right)} \right)$ (mod p), if $a^{\left(\frac{p-1}{2^e}\right)} \equiv -1$(mod p), where n is any non-residue in F_p^*.

Formula (3)

$x \equiv \pm\, b^{\left[\frac{(2^k-1)(p-1)}{2^{k+2}}\right]} a^{\left(\frac{p+2^e-1}{2^{e+1}}\right)}$ (mod p), if $a^{\left(\frac{p-1}{2^{e-k}}\right)} \equiv -1$ (mod p) for some integer k, $1 \le k \le (e-2)$, where b is a non-residue in F_p^* such that $a^{\left(\frac{p-1}{2^e}\right)} \equiv -b^{\left(\frac{p-1}{2^{k+1}}\right)}$ (mod p).

Computational procedure to find square root.
We note that any quadratic residue $a \in F_p^*$ belongs to Case (i) or Case (ii) or Case (iii).

Steps to find square root of a:

1. First check if $a^{\left(\frac{p-1}{2^e}\right)} \equiv 1$ (mod p).

   If so, square root can be calculated using formula (1). This is a deterministic formula.

2. If $a^{\left(\frac{p-1}{2^e}\right)} \equiv -1$ (mod p), square root is calculated using formula (2).

   This algorithm is probabilistic because by trial, a non-residue n has to be chosen and used in formula (2).

3. If $a^{\frac{p-1}{2^e}} \not\equiv \pm\, 1$ (mod p), then successively square the value $a^{\left(\frac{p-1}{2^e}\right)}$ and stop when $a^{\left(\frac{p-1}{2^{e-k}}\right)} \equiv -1$ (mod p), for some k, $1 \le k \le (e-2)$.

Then square root of a can be computed using formula (3). But formula (3) involves the use of a suitable non-residue $b \in F_p^*$ such that,

$$a^{\left(\frac{p-1}{2^e}\right)} = -b^{\left(\frac{p-1}{2^{k+1}}\right)} \tag{3.4}$$

To find such a non-residue one can proceed as follows:

Prepare a pre-computation table consisting of all primitive 2^e-th roots of unity in F_p^* and their higher powers. Using the table one can fix b as a suitable primitive 2^e-th root of unity in F_p^* satisfying eq (3.4). This is done as follows:

(i) Choose a non-residue n (the same non-residue n used in Case (ii) can be taken)

(ii) Using n obtain a primitive 2^e-th root of unity in F_p^* as z = n^r mod p [6]

(iii) The other primitive 2^e-th roots of unity can be generated by considering the odd powers of z. These are all non-residues in F_p^*. Hence $z, z^3, z^5, ..., z^{(2^e-1)}$ are the $2^{(e-1)}$ distinct primitive 2^e-th roots of unity in F_p^*. We can label them as $b_1, b_2, \ldots, b_{2^{(e-1)}}$

(iv) For each b_i, compute the powers $b_i^{\frac{p-1}{2^e}}, b_i^{\left(\frac{p-1}{2^{(e-1)}}\right)}, \ldots, b_i^{\left(\frac{p-1}{4}\right)}$. This can be done by first computing $b_i^{\left(\frac{p-1}{2^e}\right)}$ and successively squaring this value. When this is done for all b_i, $i = 1, 2, \ldots, 2^{(e-1)}$, these values can be stored and used as a pre-computation table consisting of $2^{(e-1)}$rows and (e−1) columns as depicted below:

General form of the pre-computation table

Table 1:

	$b^{\frac{p-1}{2^e}}$	$b^{\left(\frac{p-1}{2^{e-1}}\right)}$	...	...	...	$b^{\left(\frac{p-1}{8}\right)}$	$b^{\left(\frac{p-1}{4}\right)}$
b_1							
b_2							
⋮							
$b_{2^{e-1}}$							

Note that the size of the table depends only on e (where (p−1) = 2^e r). For instance consider the primes p = 89 (for which (p−1) = 2^3.11) and p = 99961 (for which p−1 = 2^3.12495). In both cases e = 3 and the corresponding pre-computation tables are of the same size (4 rows and 2 columns).

(v) One has to use the table to first find a b_i such that $a^{\frac{p-1}{2^e}} = -b_i^{\frac{p-1}{2^{k+1}}}$. In the column containing powers $b^{\left(\frac{p-1}{2^{k+1}}\right)}$, locate the cell containing the value $(p - a^{\frac{p-1}{2^e}})$. If this value is found in more than one cell, then any one of the cells may be chosen and the corresponding row number i may be noted.

Setting b = b_i and using formula (3.3), one can compute the square root of a. Thus it is seen that the square roots of thousands of residues (falling under case (iii)) in F_p^* can be computed using a single pre-computation table and formula(3). The number of entries in the table is (e−1) $2^{(e-1)}$. The computation is easier if e is small (where $p - 1 = 2^e r$).

Example 3.3. This is an example to illustrate formula (1).
Let p = 97, then (p−1) = 2^5.3

To find the square root of a modulo p, where $a = 35$.

It can be verified that $\left(\frac{a}{p}\right) = 1$. $a^{\frac{p-1}{32}} = 35^3 = 1$.

By formula (3.1), square root of a is $\pm\, a^{\left(\frac{p+31}{64}\right)}$ mod 97 = $\pm 35^2$ mod 97 = 61, 36.

Example 3.4. This is an example to illustrate formula (2).
Let p = 99961, then (p−1) = 2^3.12495. Here e = 3.
To find the square root of $a = 86094$.

It can be verified that $\left(\frac{a}{p}\right) = 1$. Hence a is a residue modulo p.

$a^{\frac{p-1}{2^e}}$ mod p = 86094^{12495} mod 99961.

To minimize the number of exponentiations, we calculate $a^{\frac{p-1}{2^e}}$ as,

$$a^{\frac{p-1}{2^e}} = a^{\left(\frac{p+2^e-1}{2^{e+1}}-1\right)}\left[a^{\left(\frac{p+2^e-1}{2^{e+1}}-1\right)}.a\right]$$

86094^{12495} mod p = 86094^{6247} [(86094^{6247}) (86094)] mod p = (65608) (58886) mod p = −1.

Since $a^{\frac{p-1}{2^e}} \equiv -1$ mod p, using formula (2),

Square root of a is $x \equiv \pm\left[n^{\frac{p-1}{4}} a^{\frac{p+2^e-1}{2^{e+1}}}\right] mod\ p$

Let us choose the non-residue n=19. Then

$$\begin{aligned} x &\equiv \pm\,(19^{24990})(86094)^{6248} \text{ mod } 99961 = (37804)(58,886) \text{ mod } 99961 \\ &= 94835, 5126 \end{aligned}$$

Example 3.5. This is an example to illustrate formula (3)
(i) Let p = 99961, $a = 40799$. Then (p−1) = 2^3.12495.
$a^{\frac{p-1}{8}}$ mod p = $(40{,}799)^{12495}$ mod p=40799 ^{6247} 40799^{6248} mod p = (12445)(41636)mod p=62157.

$$\left(a^{\frac{p-1}{8}}\right)^2 = 62157^2 = -1.$$

Since $a^{\frac{p-1}{4}} = -1$ using formula (3) square root of a is given by
$x \equiv \pm\left(b^{\frac{p-1}{8}} a^{\frac{p+7}{16}}\right)$ where b is a suitable non-residue such that $a^{\frac{p-1}{8}} = -b^{\frac{p-1}{4}}$.
To find b, we prepare a pre-computation table.
Let us use the non-residue n = 19 to find a primitive 2^e-th root of unity.
Set z = n^r mod p = 19^{12495} mod 99961 = 57236.
Then z, z^3, z^5, z^7 are the 4 different primitive 8^{th} roots of unity in F_p^*.

Relabelling them as b_1, b_2, b_3, b_4, it is seen that
$b_1 = 57236$, $b_2 = 93899$, $b_3 = 42725$, $b_4 = 6062$

(Observe that b_1 and b_3 are additive inverses; similarly b_2 and b_4 are additive inverses in F_p^*)

Next to prepare the pre-computation table, we have to calculate for each b_i, the powers $b_i^{\frac{p-1}{8}}$, $b_i^{\frac{p-1}{4}}$. Since $b_i^8 = 1$ and $b_i^4 = -1$, reducing $\left(\frac{p-1}{8}\right)$ modulo 8, we get,

$$b_i^{\frac{p-1}{8}} = b_i^{12495} = -b_i^3 \mod p; \quad b_i^{\frac{p-1}{4}} = b_i^6 = -b_i^2 \mod p$$

Table 2: Pre-computation table

i	b_i	$b_i^{\frac{p-1}{8}} = -b_i^3$	$b_i^{\frac{p-1}{4}} = -b_i^2$
1.	57236	6062	62157
2.	93899	42725	37804
3.	42725	93899	62157
4.	6062	57236	37804

From the table, we must choose a non-residue b such that $a^{\frac{p-1}{8}} = -b^{\frac{p-1}{4}}$. In the third column, we locate the cell containing the value $\left(p - a^{\frac{p-1}{8}}\right)$ = (99961−62157) = 37804.

This value is found in Row 2 and Row 4. Any one of the values b_2 or b_4 can be taken as the non-residue b. Taking b = b_4 = 6062, and using formula (3), square root of a is,

$$x \equiv \pm(6062^{12495})(40799)^{6248} = \pm[-(6062)^3 41636] \qquad \{\because b_4^8 = 1, b_4^4 = -1\}$$
$$= \pm(57236)(41636) = 7856, 92105$$

(ii) For the same prime p = 99961, consider a = 62157.

$\left(\frac{a}{p}\right)$=1, so a is a residue in F_p^*.

Here (p−1) = 2^3.12495.

$a^{\frac{p-1}{8}}$=$62157^{12495} \equiv 37804 \not\equiv \pm 1 \mod p$; $a^{\frac{p-1}{4}}$=$62157^{24990} \equiv -1 \pmod p$.

We have to choose a non-residue b such that $a^{\frac{p-1}{8}} = -b^{\frac{p-1}{4}}$

$\left(p - a^{\frac{p-1}{8}}\right)$ = 99961−37804 = 62157. In the third column we locate the cell containing 62157. Taking b_3 = 42725, square root of a is given by

$$x \equiv \pm\left(b_3^{\frac{p-1}{8}} a^{\frac{p+7}{16}}\right) = \pm(42725^{12495})(62157^{6248}) = \pm(-42725^3)(1)$$
$$= \pm(93899)(1) = 6062, 93899$$

Thus it is seen that square root of all residues $a \in F_p^*$ such that $a^{\frac{p-1}{4}} \equiv -1 \mod p$ can easily be computed using a single pre-computation table and formula (3).

4. New Square Root Algorithm

In this section two algorithms are presented. The first algorithm computes the primitive 2^eth roots of unity in F_p^* and their higher powers and stores these values as a pre-computation table.

Algorithm 4.1: Preparation of a pre-computation table which stores the values of primitive 2^e-th roots of unity in F_p^* (where p is prime, $p \equiv 1 \pmod 4$) and their higher powers.

Input: A prime $p \equiv 1 \pmod 4$ Output: Primitive 2^e-th roots of unity in F_p^* and their powers

1. Write $p-1 = 2^e r$, with r odd
2. [*Find non-residue*] Choose an integer n at random such that $\left(\frac{n}{p}\right) = -1$
3. [*Initialize*] Set $s \leftarrow e$, $z \leftarrow n^r \bmod p$, $b_1 \leftarrow z$ and
   $c \leftarrow z^2 \bmod p$
4. [*Find the $2^{(e-1)}$ distinct primitive 2^e-th roots of unity*]
   For i = 2 to $2^{(s-1)}$ do $b_i \leftarrow b_{i-1}\, c \bmod p$
5. [*For each b_i find the exponent* $b_i^{\left(\frac{p-1}{2^e}\right)}$. *By successively squaring this value, find* $b_i^{\left(\frac{p-1}{2^{e-1}}\right)}, \ldots, b_i^{\left(\frac{p-1}{4}\right)}$.
In finding $b_i^{\left(\frac{p-1}{2^e}\right)}$ *make use of the fact that* $b_i^{2^e} = 1$ *and* $b_i^{2^{(e-1)}} = -1$. *Then set* $a_{ij} = b_i^{\left(\frac{(p-1)}{2^{e-(j-1)}}\right)}$]
$k \leftarrow \left(\frac{p-1}{2^s}\right) \bmod 2^s$
6. For i = 1 to $2^{(s-1)}$ do
7. if $k = 2^{(s-1)}$ then $u \leftarrow (p-1)$
8. else if $k < 2^{(s-1)}$ then $u \leftarrow b_i^k \bmod p$
9. else $w \leftarrow (k - 2^{(s-1)})$, $d \leftarrow b_i^w \bmod p$ and $u \leftarrow (p-d)$
10. For j = 1 to (s−1) do
11. $a_{ij} \leftarrow u$
12. While $j \neq (s-1)$ do $u \leftarrow u^2 \bmod p$
13. [*Output b_i, i = 1, 2, ..., $2^{(e-1)}$ and a_{ij}, i = 1, 2, ..., $2^{(e-1)}$, j = 1, 2, ..., (e−1) and terminate the algorithm*]
   For i = 1 to $2^{(s-1)}$ do return b_i
      For j = 1 to (s−1) do return a_{ij}

Note:

In step 5 of the above algorithm we observe that the computation $b_i^{\left(\frac{p-1}{2^e}\right)}$ requires atmost (e-1) squarings and (e-1) multiplications. (This is because $b_i^{2^e} = 1$ and $b_i^{2^{(e-1)}} =$

-1; hence the exponent $\left(\frac{p-1}{2^e}\right)$ can be reduced to a value less than $2^{(e-1)}$).

The output from the above algorithm namely, the primitive 2^e-th roots of unity, $b_1, b_2, \ldots, b_{2^{(e-1)}}$ and their powers $a_{ij} = b_i^{\left(\frac{(p-1)}{2^{e-(j-1)}}\right)}$, $i = 1, 2, \ldots, 2^{(e-1)}$, $j = 1, 2, \ldots, (e-1)$ serves as input for the main Algorithm 4.2. As an illustration, the pre-computation table of Example 3.3 would be output as follows:

Table 3:

$b_1 = 57236$	$a_{11} = 6062$	$a_{12} = 62157$
$b_2 = 93899$	$a_{21} = 42725$	$a_{22} = 37804$
$b_3 = 42725$	$a_{31} = 93899$	$a_{32} = 62157$
$b_4 = 6062$	$a_{41} = 57236$	$a_{42} = 37804$

The second algorithm computes the square root modulo p (p prime, $p \equiv 1 \bmod 4$) of any quadratic residue $a \in F_p^*$, by using the output from Algorithm 4.1

Algorithm 4.2: Square root computation

Input: 1. A prime p such that $p \equiv 1 \pmod 4$
2. An integer a such that $\left(\frac{a}{p}\right) = 1$
3. An integer n such that $\left(\frac{n}{p}\right) = -1$
4. The pre-computed values of the primitive 2^e-th roots of unity in F_p^* namely b_i,
$i = 1, 2, \ldots, 2^{(e-1)}$
5. The pre-computed values of the powers of these primitive roots $a_{ij} = b_i^{\left(\frac{(p-1)}{2^{e-(j-1)}}\right)}$,
$i = 1, 2, \ldots, 2^{(e-1)}$, $j = 1, 2, \ldots, (e-1)$
Output: Square root of a (i.e) an integer x such that $x^2 \equiv a$

1. Write $(p-1) = 2^e r$, with r odd.
2. Choose an integer n at random such that $\left(\frac{n}{p}\right) = -1$
3. [*Initialize*] $s \leftarrow e$, $c \leftarrow \left(\frac{p+2^s-1}{2^{s+1}}\right)$, $d \leftarrow (c-1)$
4. $g \leftarrow a^d \bmod p$, $h \leftarrow (a^d \cdot a) \pmod p$, $u \leftarrow gh \bmod p$ and $w \leftarrow u$

[*If* $a^{\frac{p-1}{2^e}} = 1$*, use formula (1) to find square root*]

5. if $u = 1$ then $x \leftarrow h$ and $y \leftarrow (p-h)$
6. return x, y

[*If* $a^{\frac{p-1}{2^e}} = -1$ *use formula (2) to find square root*]

7. else if u = (p−1) then
8. $\quad$ t ← $n^{\left(\frac{p-1}{4}\right)}$ mod p, x ← th mod p and y ← (p−x)
9. return x, y

[If $a^{\frac{p-1}{2^e}} \neq \pm 1$, find a k such that $a^{\frac{p-1}{2^{(e-k)}}} \equiv -1$ mod p. This is done by repeated squarings. Once k is fixed, the pre-computed values a_{ij} are scanned to locate a suitable bi. Then formula (3) has to be used to find square root. For this, the value of $b_i^{\left(\frac{p-1}{2^{k+2}}\right)(2^k-1)}$ has to be computed. Since $b_i^{\frac{p-1}{2^{k+2}}}$ is a pre-computed value, we can obtain this from the table. The column m is found as m = (e−1−k) and the corresponding $a_{\mathrm{im}} = b_i^{\frac{p-1}{2^{k+2}}}$. Hence $b_i^{\left[\left(\frac{p-1}{2^{k+2}}\right)(2^k-1)\right]} \equiv a_{\mathrm{im}}^{(2^k-1)}$ mod p. Instead of evaluating this as an exponentiation, we can do it as repeated squaring and multiplication using the formula $(2^k-1) = 1 + 2 + 2^2 + 2^3 + \ldots + 2^{(k-1)}$. Hence $x^{(2^k-1)} = x.x^2.x^{2^2}.x^{2^3}.\cdots.x^{2^{(k-2)}}.x^{2^{(k-1)}}$]

10. else
11. $\quad$ j ← (s−1)
12. $\quad$ k ← 1
13. $\quad$ u ← u^2
14. $\quad$ while u ≠ (p−1) do
15. $\quad\quad$ u ← u^2
16. $\quad\quad$ k ← (k+1)
17. $\quad\quad$ j ← (j−1)
18. $\quad$ q ← (p−w)
19. $\quad$ i ← 1
20. $\quad$ while $a_{ij} \neq$ q do i ← (i+1)
21. $\quad$ b ← b_i
22. $\quad$ m ← (s−1−k)
23. $\quad$ t ← a_{im}
24. $\quad$ v ← a_{im}^2 mod p
25. $\quad$ while k > 1 do
26. $\quad\quad\quad$ for ℓ = 1 to (k−1) do
27. $\quad\quad\quad$ t ← tv mod p
28. $\quad\quad\quad$ v ← v^2 mod p
29. $\quad$ x ← th mod p
30. $\quad$ y ← (p−x)
31. $\quad$ return x, y.

5. Computational Complexity of the New Square Root Algorithm

In this section, the computational complexity of the new square root algorithm is calculated. Additions and subtractions are not counted and the computational complexity is considered only for multiplications and divisions.

We first discuss the time estimate for formula (1). This is a direct formula which uses one exponentiation and two multiplications and this takes $O(\log^3 p + \log^2 p)$ bit operations. The proportion of residues in F_p^* falling under case (i) is $\frac{(p-1)}{2^e} \div \frac{(p-1)}{2} = \frac{1}{2^{(e-1)}}$. For example, in the prime field F_p^*, p = 99961, we have e=3 (since $(p-1) = 2^3.12495$) and 25% of the residues fall under case (i), whose square roots can be calculated using formula (1).

Next, the time estimate for formula (2) is discussed. Suppose that a non residue n had already been chosen. Then formula (2) uses 2 exponentiations and 3 multiplications and this takes $O(\log^3 p + \log^2 p)$ bit operations. The proportion of residues in F_p^* covered by formula (3.2) is $\frac{1}{2^{(e-1)}}$. Again when e = 3, this works out to 25% of the residues in F_p^*.

The time estimate for formula (3) is calculated as follows. Formula (3) uses the pre-computation table. Preparation of this table by Algorithm 4.1 requires 1 exponentiation, $e2^{(e-1)}$ multiplications and (1+(e-1) 2^e) squarings. This takes $O(\log^3 p + e2^{(e-1)}\log^2 p)$ bit operations. Preparation of this table is a one-time computation and can be used to find square roots of thousand of residues in F_p^* falling under case (iii).

Using the pre-computed table, formula (3) requires 1 exponentiation, 4 multiplications and atmost e(e-2) squarings to find the square root of a residue falling under case (iii). This takes $O(\log^3 p + e^2\log^2 p)$ bit operations. The proportion of residues in F_p^* falling under case (iii) is $\left(\frac{2^{(e-2)} - 1}{2^{(e-2)}}\right)$.

Comparison of the New Method with the Existing Methods

We present some numerical examples to compare the efficiency of the new algorithm with existing algorithms. The results are tabulated below.

Example 5.1. An example to illustrate formula (2)
To find the square root of a = 86094 modulo p where p = 99961. Here $(p-1) = 2^3.12495 = 2^e r$. This example falls under case (ii). One sees that $e = 3$, $r = 12495$; Choose a non-residue $n = 19$. Refer to Table 4 given below.

Example 5.2. An example to illustrate formula (3)
To find the square root of a = 40799 modulo p where p = 99961. Here $(p-1) = 2^3.12495 = 2^e r$. Hence $e = 3$, $r = 12495$. This example falls under case (iii). Choose a non-residue $n = 19$. Refer to Table 5 given below.
Let us denote the arithmetic operations as follows:
E = exponentiation, S = squaring, M = multiplication, D = division.

We now compare the efficiency of the New Square Root Algorithm with the Tonelli and Shanks Algorithm by calculating the number of arithmetic operations used by each. Let us denote the arithmetic operations as follows:
M - Multiplication, E - Exponentiation, S - Squaring, D - Division

Table 4:

Method used	$z = n^r \ mod\ p$	$y \leftarrow z$	s	$x = a^{\frac{r-1}{2}} \ mod\ p$	$b = ax^2$	$x \leftarrow ax$	m	b^{2^m}	No. of operat. used
Tonelli & Shanks Algorithm	$19^{12395} = 57236$	57236	3	$86094^{6247} = 65608$	(89094) $(65608)^2 = -1$	(86094) 65608 = 58886	1	1	2E 3S 3M
	$t \leftarrow y^{2^{(s-m-1)}}$ $mod\ p$	$y = t^2 \ mod\ p$	s	$x \leftarrow tx \ mod\ p$	$b \leftarrow yb \ mod\ p$			square root	
	$57236^2 = 37804$	$37804^2 = -1$	1	(37804) (58886) = 94835	1			94835	
Our New Square Root Algorithm	$a^{\frac{p-1}{8}} \ mod\ p = a^{\left(\frac{p+7}{16}-1\right)} a^{\left(\frac{p+7}{16}-1\right)} a$ $mod\ p$			$n^{\frac{p-1}{4}} \ mod\ p$		$sq.root = \pm n^{\frac{p-1}{4}} a^{\left(\frac{p+7}{16}\right)}$			2E
	86094^{12495} $= (86094^{6247})(86094^{6247} 86094)$ = (65608) (58886) - 99960 = -1			$19^{24990} = 37804$		$\pm(37804)(58886)$ $= 94835, 5126$			3M

Table 5:

Method used	$z = n^r$ $mod\ p$	$y \leftarrow z$	s $mod\ p$	$x = a^{\frac{r-1}{2}}$	$b = ax^2$	$x \leftarrow ax$	m	b^{2^m}	No. of operat. used
Tonelli & Shanks Algorithm	19^{12495} 57236	57236	3	40799^{6247} = 12445	(40799) $(12445)^2$ = 62157	(40799) (12445) = 41636	1 2	-1 1	2E 3S 4M
	$t \leftarrow y^{2^{(s-m-1)}}$ $mod\ p$	$y = t^2$ $mod\ p$	s	$x \leftarrow tx$ $mod\ p$	$b \leftarrow yb$ $mod\ p$			square root	
	57236	57236^2 = 37804	2	(57236) (41636) = 7856	(37804) 62157 = 1			7856	
Our New Square Root Algorithm	Pre-Computation Table (Refer to Example 3.2). This is a one-time computation and can be used for thousands of residues falling under case (iii)								5E, 3M 5S, 1D
	$a^{\frac{p-1}{8}} = a^{\left(\frac{p+7}{16}-1\right)} a^{\left(\frac{p+7}{16}-1\right)} a\ mod\ p$		$a^{\frac{p-1}{4}}$ $mod\ p$	b	$b^{\frac{p-1}{8}} = -b^3$ $mod\ p$	$sq.root = \pm b^{\frac{p-1}{8}} a^{\left(\frac{p+7}{16}\right)}$ $mod\ p$			1E 4M 2S
	$(40799)^{12495}$ $= 40799^{6247}.40799^{6248}$ $= (12455)(41636)$ $= 62157$		62157^2 = -1	6062	$-(6062^3)$ = 57236	$(57236)(40799^{6248})$ = (57236)41636 = 7856			

	No. of operations used by Tonelli and Shanks Algorithm	No. of operations used by our New Square Root Algorithm	Efficiency of the New Algorithm
To find square root of a residue falling under case (i)	2E + 4M + atmost e(2e-1)S	Formula 1: 1E + 2M	100% more efficient
To find square root of a residue falling under case (ii)	2E + 4M + atmost e(2e-1)S	Formula 2: 2E + 3M	About 10% more efficient
To find square root of 1000 residue falling under case (iii)	1001E 4000M 1000 e(2e-1)S	**Pre computation Table:** 1E + $e2^{(e-1)}M$ + $(1 + (e - 1)2^e)S$ + **Formula 3:** 1000E + 4000M + 1000 e(e-2)S atmost	More efficient when e is small ($e \leq 13$)

6. Conclusion

In setting up cryptosystems, there has always been a need for a simple formula to compute square roots of quadratic residues modulo p where $p \equiv 1 \pmod 4$. Many a times, in the absence of a simple formula, a restriction on the choice of primes p is imposed, to primes $p \equiv 3 \bmod 4$. The algorithm proposed in this paper to find square roots of quadratic residues modulo p, $p \equiv 1 \pmod 4$ is efficient and has computational ease. This will facilitate the setting up of elliptic and hyper-elliptic curve cryptosystems over a broader class of finite fields F_p, for a prime p.

References

[1] Rabin, M.O, 1979, "Digitized signatures and public-key functions as intractable as factorization," MIT Laboratory for Computer Science, Technical Report, LCS/TR-212.

[2] Henri Cohen and Gerhard Frey, 2006, Handbook of elliptic and hyperelliptic curve cryptography, Chapman & Hall/CRC, pp. 211.

[3] Alberto Tonelli, 1891, "Bemerkung über die Auflösung quadratischar Congruenzen," Nachrichten der Akademie der Wissenschaften in Göttingen, pp. 344–346.

[4] Michele Cipolla, 1903, "Un metodo per la risoluzione della congruenza di secondo grado," Napoli Rend., 9, pp. 154–163.

[5] Derrick H. Lehmer, 1969, "Computer Technology applied to the theory of numbers," Studies in number theory (Englewood Cliffs, New Jersey) (William J. Leyeque, ed.), MAA studies in Mathematics, vol. 6, Prentice Hall, pp. 117–151. MR0246815 (40:84).

[6] Daniel Shanks, 1972, "Five number theoretical algorithms," Proceedings, 2^{nd} Manitoba Conference on Numerical Mathematics, pp. 51–70, MR0371855(51:8072).

[7] Leonard M. Adleman, Kenneth L. Manders and Gary L. Miller, 1977, "On taking roots in finite fields," Proceedings of the 18^{th} IEEE Symposium on Foundations of Computer Science, IEEE, pp. 175–178. MR0502224 (58:19339).

[8] Elwyn R. Berlekamp, 1970, "Factoring polynomials over large finite fields," Math. Comp., 24, no. 111, pp. 713–735. MR0276200 (43:948).

[9] Michael O. Rabin, 1980, "Probabilistic algorithms in finite fields," SIAM J. Comput. 9, no. 2, pp. 273–280, MR568814 (81g:12002).

[10] René C. Peralta, 1986, "A simple and fast probabilistic algorithm for computing square roots modulo a prime number," IEEE Transactions on Information Theory, 32, no. 6, pp. 846–847, MR868931 (87m:11125).

[11] Eric Bach, 1990, "A note on square roots in finite fields," IEEE Transactions on Information Theory, 36, no. 6, pp. 1494–1498, MR1080838 (91h:11140).

[12] Stephen M. Turner, 1994, "Square roots mod p," The American Mathematical Monthly, 101, no. 5, pp. 443–449, MR1272944 (95c:11004).

[13] Eric Bach and Klaus Huber, 1999, "Note on taking square-roots modulo N," IEEE Transactions on Information Theory, 45, no. 2, pp. 807–809, MR1677049 (99j:94036).

[14] Siguna Müller, 2000, "On probable prime testing and the computation of square roots mod n," Algorithmic Number Theory, 4^{th} International Symposium, ANTS-IV, Lecture Notes in Computer Science, 1838, Springer-Verlag, pp. 423–437, MR1850623 (2002h:11140).

[15] Daniel J. Bernstein, 2001, "Faster square roots in annoying finite fields," preprint. (http://cr.yp.to/papers/sqroot.pdf).

[16] René Schoof, 1985, "Elliptic curves over finite field and the computation of square roots mod p," Mathematics of Computation, 44, no. 170, pp. 483–494, MR777280.

[17] Tsz-Wo Sze, 2011, "On Taking Square Roots without Quadratic non-residues over finite fields," Mathematics of Computation, 80, no. 275, pp. 1797–1811.

[18] Cohen, H., 1993, A Course in Computational Algebraic number theory, Volume 138 of Graduate Texts in Mathematics, Springer Verlag, Berlin, pp. 32–33.

[19] Koblitz, N., 1994, A course in Number Theory and Cryptography (Second Edition), pp. 48.