Square root computation over even extension fields

Gora Adj [1] and Francisco Rodríguez-Henríquez[2]

[1] Université Claude Bernard Lyon 1,ISFA, France

[2] Computer Science Department, CINVESTAV-IPN, México

Abstract

This paper presents a comprehensive study of the computation of square roots over finite extension fields. We propose two novel algorithms for computing square roots over even field extensions of the form $\mathbb{F}_{q^2}$, with $q = p^n$, p an odd prime and $n \geq 1$. Both algorithms have an associate computational cost roughly equivalent to one exponentiation in $\mathbb{F}_{q^2}$. The first algorithm is devoted to the case when $q \equiv 1 \bmod 4$, whereas the second one handles the case when $q \equiv 3 \bmod 4$. Numerical comparisons show that the two algorithms presented in this paper are competitive and in some cases more efficient than the square root methods previously known.

keyword: Modular square root, finite field arithmetic.

I. Introduction

Taking square roots over finite fields is a classical number theoretical problem that has been addressed by mathematicians across the centuries. In modern times, the computation of modular square roots is especially relevant for elliptic curve cryptosystems, where hashing an arbitrary message to a random point that belongs to a given elliptic curve [10], point compression [24], [1], [6] and point counting over elliptic curves [26], [2], are some of its most relevant cryptographic applications. Quite often, the above applications require computing square roots in finite extension fields. In particular, a good number of pairing-based protocols defined over popular choices of pairing-friendly elliptic curves such as the Barreto-Naehrig, the Kachisa-Schaefer-Scott or the Barreto-Lynn-Scott elliptic curves, require computing square roots over either quadratic or cubic extension fields [5], [18], [16].

Let q be a positive power of a large odd prime p, i.e., $q = p^m$, with $m \geq 1$. It is known that q uniquely defines a finite field denoted as $\mathbb{F}_q$. The problem of computing a field square root of any arbitrary element $a \in \mathbb{F}_q$ consists of finding a second element $b \in \mathbb{F}_q$ such that $b^2 = a$. According to the Euler criterion, also known as the quadratic residue test, the square root of an element $a \in \mathbb{F}_q^*$ exist iff $a^{\frac{q-1}{2}} = 1$, we denote by $\chi_q(a)$ the value of $a^{\frac{q-1}{2}}$. If $\chi_q(a) = 1$, we say that the element a is a quadratic residue (QR) in $\mathbb{F}_q$. It is known that in $\mathbb{F}_q^*$ there exist exactly $(q-1)/2$ quadratic residues.

Two classical, non-deterministic techniques for computing square roots in prime extension fields are the Tonelli-Shanks [30] and the Cipolla-Lehmer [12] algorithms. However, it is normally the case that finding the square root of a field element a can be achieved more easily by using specialized methods as it is briefly discussed next.

In the case that $q \equiv 3 \pmod 4$, one can simply use a specialized version of the Tonelli-Shanks procedure, the Shanks algorithm, where the square root of a quadratic residue $a \in \mathbb{F}_q$, can be computed via one single exponentiation as, $b = a^{\frac{q+1}{4}}$. On the other hand, no simple and general algorithm for the class $q \equiv 1 \pmod 4$ is known. However, fast algorithms for computing a square root in $\mathbb{F}_q$ when $q \equiv 5 \pmod 8$ or $q \equiv 9 \pmod{16}$ have been reported.

For the case when $q \equiv 5 \pmod 8$, Atkin developed in 1992 an efficient and deterministic square root algorithm that is able to find the square root of a quadratic residue using only one field exponentiation plus some few multiplications in $\mathbb{F}_q$ [2]. A modification of the Atkin's algorithm was presented by Müller in [25], that allows one to compute square roots in $\mathbb{F}_q$ when $q \equiv 9 \pmod{16}$, at the price of two exponentiations. By exploiting a regular structure of the exponent $(q-9)/16$ when written in base p, authors in [22], were able to simplify the overall cost of the Müller procedure to only one exponentiation for half of the quadratic residues in $\mathbb{F}_q$, and two exponentiations for the other half.

It is worth mentioning that in the case when $q \equiv 1 \pmod{16}$, no specialized algorithm is known. Hence, for this class of extension fields one is forced to resort to the aforementioned classical methods, namely, the Tonelli-Shanks algorithm or a modified version of the Cipolla-Lehmer algorithm also presented by Müller in [25].

Square root computation of extension fields $\mathbb{F}_{p^m}$, with m odd. Several authors have analyzed the square root problem in odd finite extension fields. In [4], Barreto *et al.* presented an efficient algorithm that can compute square roots for fields of this form, whenever $p \equiv 3 \bmod 4$ or

$p \equiv 5 \bmod 8$. The latter case can be seen as a variant of the Atkin method mentioned above. The main idea of the Barreto *et al.* procedure is to rewrite the exponents required for computing the square root in base p. Then, those exponentiation operations can be calculated efficiently by exploiting a recursive procedure that is essentially the same as the one used in the Itoh-Tsujii inversion method [29]. This recursive procedure takes advantage of the fact that the Frobenius map in characteristic p, which consists of the exponentiation of a field element a to the p-th power, i.e. $a^p \in \mathbb{F}_q$, is a simple operation that can be computed at an inexpensive cost or even at no cost if the field elements are represented in normal basis [7].

The technique in [4] was systematically applied by Han-Choi-Kim in [17] for all the specialized methods when $p \equiv 3 \pmod 4$, $5 \pmod 8$ or $9 \pmod{16}$. Authors in [17] also improved the general Tonelli-Shanks method that is normally one of the best choices for tackling the difficult case when $p \equiv 1 \pmod{16}$. Let us write $p^m - 1$ as, $p^m - 1 = 2^s \cdot t$, where s is a positive integer and t an odd number. Then, in order to compute the square root of an arbitrary quadratic residue $a \in \mathbb{F}_q$, the single most expensive operation that the Tonelli-Shanks procedure performs, is the exponentiation $a^{\frac{t-1}{2}}$. As it was shown in [17], this operation can be considerably sped up by once again exploiting the idea of rewriting the exponent $(t-1)/2$ in base p.

Square root computation of extension fields $\mathbb{F}_{p^m}$, with m even. Relatively less work has been reported for even extension fields. Finding square roots for these fields can sometimes be achieved by *descending* some of the required computations in $\mathbb{F}_{p^m}$ to proper subfields of the form $\mathbb{F}_{p^i}$, with $i \geq 1$ and $i|m$. In this context, authors in [19], [20], [32] used a Tonelli-Shanks based approach in order to have most of the computations reduced to proper subfields of $\mathbb{F}_{p^m}$. More recently, authors in [14] presented an algorithm that takes roots over $\mathbb{F}_{p^m}$ by descending the computation until the base field $\mathbb{F}_p$ using the trace function. The complexity analysis presented in [14] is asymptotic.

Scott adapted in [27] the complex square root formula presented in [15] to the computation of square roots in quadratic extension fields of the form $\mathbb{F}_{q^2}$, $q = p^n$. The computational cost of this algorithm is of just two square roots, one quadratic residue test and one field inversion over $\mathbb{F}_q$. As it will be discussed in the rest of this paper, the complex method formula ranks among the most efficient methods for computing square roots over even extension fields.

Contributions of this paper. As a first contribution, we present a procedure that can compute $\chi_q(a)$, with $q = p^m$ at the cost of several Frobenius exponentiations over $\mathbb{F}_q$ plus the computation

of the Legendre symbol in the base field $\mathbb{F}_p$, which is more efficient than the recursive algorithm proposed by Bach and Huber in [3]. Furthermore, a general review of the classical square root algorithms over finite extension fields $\mathbb{F}_q$ is provided.

In the case of field extensions $\mathbb{F}_{p^m}$ with m odd, we revisit efficient formulations of several square root algorithms where the quadratic residue test of the input operand is interleaved in such a manner that only some constant number of multiplications are added to the overall algorithm computational cost.[1] A detailed complexity analysis of all the reviewed algorithms is also given. In particular and to the best of our knowledge, the complexity analysis of Algorithm 7 that corresponds to the Müller procedure for the subclass $q \equiv 1 \pmod{16}$, has not been reported before in the open literature.

Furthermore, we propose two new algorithms that given a field element $a \in \mathbb{F}_{q^2}$, with $q = p^n$, can compute a square root of a or show that it does not exist. These two algorithms are complementary in the sense that they cover separately the two congruence classes that odd primes define, namely, $q \equiv 1 \pmod 4$ and $q \equiv 3 \pmod 4$.

For the class $q \equiv 3 \pmod 4$, we present a deterministic procedure that in some sense can be seen as a generalized Shanks algorithm for finite fields with even extensions. In this case the proposed algorithm computes a square root by performing two exponentiations, each of them with associate exponents of bit-length N, with $N = \log_2(q)$.

For the class $q \equiv 1 \pmod 4$, one could compute the square root of a quadratic residue $a \in \mathbb{F}_{q^2}$ by directly working in that extension field. In contrast, our second proposed algorithm computes the square root by performing first one exponentiation in $\mathbb{F}_{q^2}$, with an exponent of length of about N bits, followed by the computation of one square root in the subfield $\mathbb{F}_q$.

Our experiments show that the two square roots algorithms proposed in this paper are competitive when compared against the complex method of [27], and the Tonelli-Shanks and the Müller's procedures. Fig. 1 shows a taxonomy of efficient algorithms that compute the square root over $\mathbb{F}_{p^m}$, with p an odd prime and $m \geq 1$

The rest of this paper is organized as follows. In Section II we give the notation and basic definitions of the arithmetic operations that will be used for evaluating the computational complexities of the square root algorithms studied in this paper. Then, in Section III an efficient

[1]With the only exception of Algorithm 7 that reproduces one of the procedures that Müller introduced in [25].

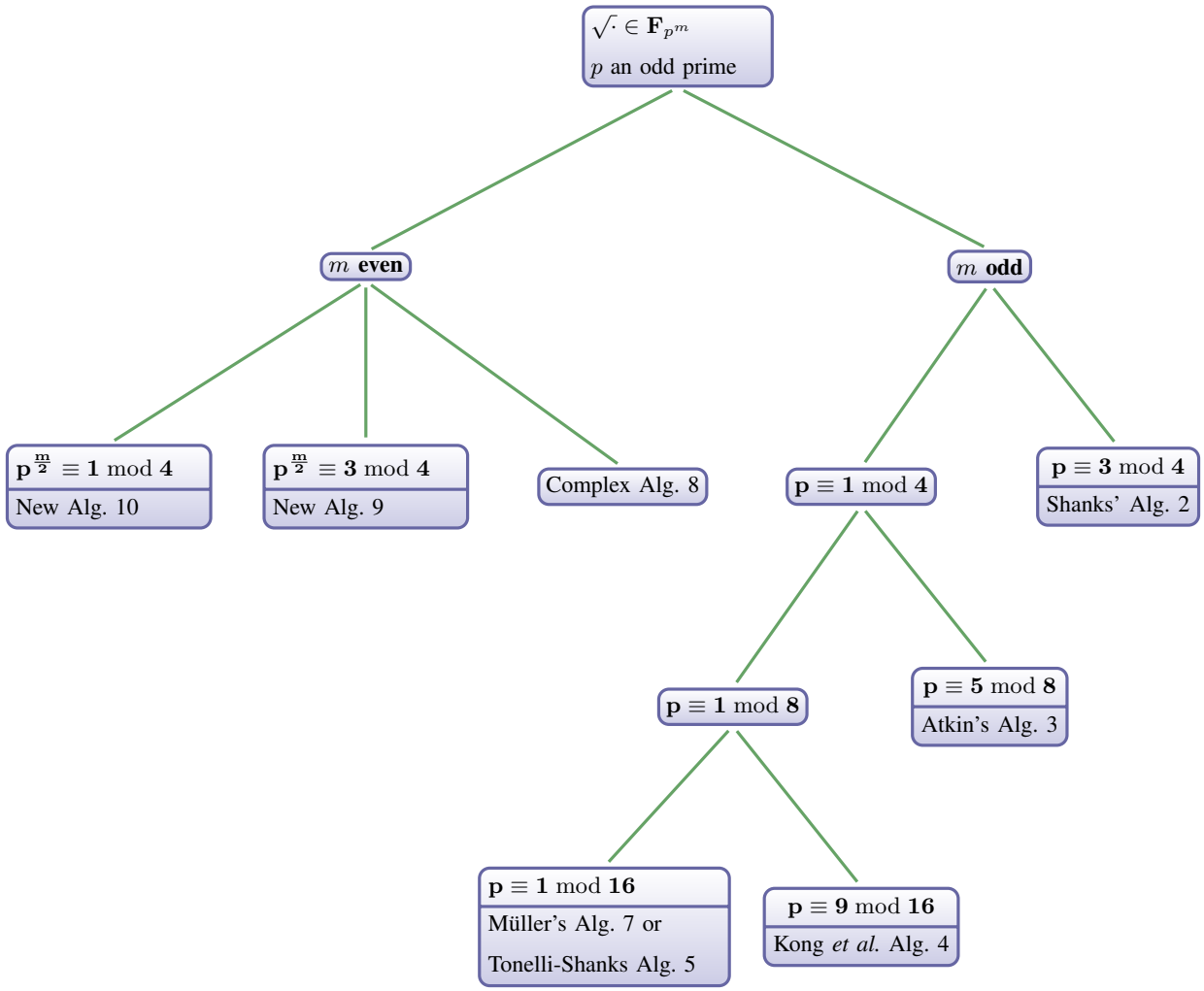

Figure 1. A taxonomy of efficient algorithms that compute the square root over $\mathbb{F}_{p^m}$, p an odd prime and $m \geq 1$

method for computing quadratic residue tests over field extensions is presented. Section IV gives a comprehensive review of known algorithms over extension fields $\mathbb{F}_{p^m}$ with m odd, whereas Section V studies the computation of square roots over extension fields $\mathbb{F}_{p^m}$ with m even. In Section VI a comparison of our algorithms against previously known methods by choosing BN curve primes [6] and NIST recommended primes for elliptic curve cryptography [1] is given. Finally, some conclusion remarks are drawn in Section VII.

II. PRELIMINARIES

Throughout this paper, most of the described algorithms have both *precomputation* and *computation* phases. However, as it is customary when evaluating the complexity of a given algorithm, we will not consider the precomputation effort and will give only the costs associated to the computation phase.

In a finite field $\mathbb{F}_q$, the square-and-multiply exponentiation method (also known as the binary exponentiation method) is a standard strategy for computing field exponentiations of the form a^s, where the exponent s is a positive integer, smaller than the order of the multiplicative group. In average, the binary strategy requires a total of $\lfloor \log_2(s) \rfloor$ squarings and $\mathrm{Hw}(s) - 1$ field multiplications, where $\mathrm{Hw}(s)$ is the Hamming weight of s. In the rest of this paper it will be assumed that the average Hamming weight of a random odd integer s is given as [23], $\frac{1}{2}\lfloor \log_2(s) \rfloor + \frac{3}{2}$.

For a quadratic non-residue (QNR) element $\beta \in \mathbb{F}_q$, the binomial $f(y) = y^2 - \beta$ is irreducible over $\mathbb{F}_q[y]$, which means that the quadratic extension $\mathbb{F}_{q^2}$ of the base field is isomorphic to $\mathbb{F}_q[y]/\left(f(y)\right)$. A field element $a \in \mathbb{F}_{q^2}$ can be represented as $a = a_0 + a_1 y$, with $a_0, a_1 \in \mathbb{F}_q$. A multiplication and a squaring in $\mathbb{F}_{q^2}$ can be computed at a cost of three and two multiplications in $\mathbb{F}_q$, and one and two multiplications by a constant in $\mathbb{F}_q$, respectively.[2] Likewise, a multiplication between an element of $\mathbb{F}_q$ and an element of $\mathbb{F}_{q^2}$ amounts for two multiplications in $\mathbb{F}_q$. Since $(a_0 + a_1 y)^{-1} = (a_0 - a_1 y)/(a_0^2 + \beta \cdot a_1^2)$, computing the inverse of $a \in \mathbb{F}_{q^2}$ requires one inversion and at most 5 multiplications in $\mathbb{F}_q$ (in fact, if $\beta = \pm 1$ only 4 multiplications in $\mathbb{F}_q$ are required). Applying the Frobenius operator over an arbitrary field element a is essentially free of cost since $(a_0 + a_1 y)^q = (a_0 - a_1 y)$, i.e., the result of raising an element to the power q is its conjugate. Notice also that this implies that $a^{q+1} = a \cdot \bar{a} = a_0^2 + a_1^2$ is in $\mathbb{F}_q$. Moreover, if the element a is a quadratic residue, then $a^{\frac{q+1}{2}}$ also lies in $\mathbb{F}_q$. We will consider that the addition operations have a negligible cost, and thus they will be ignored from our estimations.

The application of the Frobenius operator over a field element $a \in \mathbb{F}_{q^k}$, with $k > 2$, can be computed efficiently for reasonable choices of irreducible polynomials involved in the construction of the associated *field towering* [8], [21]. In this scenario the computation of a^q can

[2] using a multiplication *à la* Karatsuba and the so-called complex method, respectively. [13], [11].

be achieved at the price of at most $k-1$ field multiplications over $\mathbb{F}_q$ [9].[3]

In the remainder part of this paper, M_q, S_q and Mc_q will denote the cost of a multiplication, a squaring and a multiplication by a constant in $\mathbb{F}_q$, respectively. The cost of an inversion is denoted by I_q in any given field $\mathbb{F}_q$. Moreover, we state F_q as the cost of a Frobenius operation a^{p^i}, with $a \in \mathbb{F}_q$, $q = p^m$ and $1 \leq i < m$. $Lucas(k)$ will denote the complexity of computing the k-th element of a Lucas sequence, whereas $C_q(k,c)$ will denote the complexity cost of computing the Frobenius exponentiation, $a^{1+p^c+(p^c)^2+\cdots+(p^c)^{k-1}}$ in $\mathbb{F}_q$, with $c, k \geq 1$. Finally, we denote by $SQRT_q$, the complexity of computing a square root in the field $\mathbb{F}_q$ by using the most efficient method for that extension field.

III. A REMARK ON THE COMPUTATION OF QUADRATIC RESIDUE TEST OVER FIELD EXTENSIONS

In [3], Bach and Huber used the Legendre symbol for computing the quadratic character of an extension field element $a \in \mathbb{F}_q^*$, with $q = p^m$, p an odd prime and $m > 1$. By recursively invoking the law of quadratic reciprocity, the authors proved that the asymptotic cost of this method is of $O(\log q)^2$ bit operations. Here, we present an alternative formulation that computes the quadratic residue test by *descending* its computation to the base field $\mathbb{F}_p$ plus the evaluation of several Frobenius operations. This procedure is considerably more efficient than the algorithm of [3], provided that the Frobenius operator can be computed inexpensively.

As it was mentioned in the Introduction section, the quadratic residue test on an element $a \in \mathbb{F}_q^*$, with $q = p^m$ can be computed via the exponentiation, $a^{\frac{q-1}{2}}$. For $m \geq 1$, the following factorization of the exponent,

$$\frac{q-1}{2} = \frac{p-1}{2} \sum_{i=0}^{m-1} p^i, \tag{1}$$

can be used to descend the exponentiation $a^{\frac{q-1}{2}}$ to one quadratic residue test in the base field $\mathbb{F}_p$ after applying the addition chain exponentiation method that was first described in [7]. Indeed, the value $b = a^{\sum_{i=0}^{m-1} p^i}$ is nothing more than the norm of a in the sub-field $\mathbb{F}_p$ of $\mathbb{F}_q$, which implies that $b \in \mathbb{F}_p$. For the sake of efficiency, notice that after computing b, instead of performing the exponentiation, $a^{\frac{q-1}{2}} = b^{\frac{p-1}{2}}$, the customary Legendre symbol computation on $b \in \mathbb{F}_p$ can be

[3]We stress that if normal basis representation is used then the computation of the Frobenius operator is free of cost.

carried out as described in Alg. 1. The cost of computing b in polynomial basis is estimated in Appendix A as,

$$\frac{3}{2}\left[\lfloor \log_2 m \rfloor + 1\right](M_q + F_q),$$

whereas the computation of the Legendre symbol of a non-zero base field element b has a complexity similar to that of computing the greatest common divisor of b and p [3].

Algorithm 1 Quadratic residue test of $a \in \mathbb{F}_q, q = p^m, m > 1$

Require: $a \in \mathbb{F}_q$, $q = p^m$, $m > 1$.
Ensure: $\chi_q(a)$
1: $b \leftarrow C_{m-1,1}(a)$. {via Algorithm 11}
2: $c \leftarrow b^{\frac{p-1}{2}}$. {via a Legendre symbol computation $\left(\frac{b}{p}\right)$over $\mathbb{F}_p$}
3: **return** c.

IV. Square roots in odd extension fields

The algorithms that compute square roots over finite extension fields $\mathbb{F}_q$ where $q = p^m$, p is a large odd prime and $m > 1$, can be classified into two main cases. On the one hand, we have the class where q is congruent to $1 \pmod 4$ and on the other hand the class when q is congruent to $3 \pmod 4$.[4] We first describe the easiest case $q \equiv 3 \pmod 4$ before handling $q \equiv 1 \pmod 4$ which can be much more costly in some cases as it will be discussed later.

A. *Square roots in $\mathbb{F}_q$ when $q \equiv 3 \pmod 4$*

Computing the square root of an arbitrary quadratic residue $a \in \mathbb{F}_q$, where $q \equiv 3 \pmod 4$, can be done with only one exponentiation, via the computation of $a^{\frac{q+1}{4}}$, that can be seen as the simplest instance of the Shanks's method [28]. The quadratic residue test of an arbitrary field element $a \in \mathbb{F}_q$ has been integrated into Algorithm 2. If a is a quadratic residue it returns its square root and false otherwise.

[4]In the case of odd extension fields, if $p \equiv \pm 1 \pmod 4$ then also $p^m \equiv \pm 1 \pmod 4$.

Algorithm 2 Shanks's algorithm when $q \equiv 3 \pmod 4$

Require: $a \in \mathbb{F}_q^*$.
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.
1: $a_1 \leftarrow a^{\frac{q-3}{4}}$.
2: $a_0 \leftarrow a_1(a_1 a)$.
3: **if** $a_0 = -1$ **then**
4: **return** false.
5: **end if**
6: $x \leftarrow a_1 a$.
7: **return** x.

The computational cost of Algorithm 2 is one exponentiation and two multiplications. In 2007, Scott in [27] showed that the complexity of the exponentiation in Step 1 could be further reduced by rewriting the exponent in base p. This was rediscovered by Han *et al.* [17], who proposed factorizing the exponent $(q-3)/4$ as,

$$\frac{q-3}{4} = \alpha + p\left[p\alpha + (3\alpha + 2)\right] \sum_{i=0}^{(m-3)/2} p^{2i}, \tag{2}$$

where $\alpha = \frac{p-3}{4}$.

Using the factorization of the exponent $\frac{q-3}{4}$ given in Eq. (2), it can be shown that the average complexity of Algorithm 2 when a is a square, is given as (see Appendix B for details),

$$\left[\frac{1}{2}\lfloor \log_2(p) \rfloor + \frac{3}{2}\lfloor \log_2(m) \rfloor + \frac{5}{2}\right] M_q + \left[\lfloor \log_2(p) \rfloor - 2\right] S_q + \left[\frac{3}{2}\lfloor \log_2(m) \rfloor + 2\right] F_q. \tag{3}$$

B. Square roots in $\mathbb{F}_q$ when $q \equiv 1 \pmod 4$

For this class, it is customary to consider the sub-congruences modulo 8 or modulo 16. Indeed, despite the fact that there is no simple and general algorithm for $q \equiv 1 \pmod 4$, fast algorithms for computing a square root in $\mathbb{F}_q$ when $q \equiv 5 \pmod 8$ or $q \equiv 9 \pmod{16}$ are known.

1) Atkin's algorithm: When q is congruent to $5 \pmod 8$, Atkin [2] developed an efficient method to compute a square root of a QR in $\mathbb{F}_q$ by performing one exponentiation and a constant number of multiplications.

Algorithm 3 Atkin algorihtm when $q \equiv 5 \pmod 8$

Require: $a \in \mathbb{F}_q^*$.

Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.

PRECOMPUTATION

1: $t \leftarrow 2^{\frac{q-5}{8}}$.

COMPUTATION

1: $a_1 \leftarrow a^{\frac{q-5}{8}}$.

2: $a_0 \leftarrow (a_1^2 a)^2$.

3: **if** $a_0 = -1$ **then**

4: **return** false.

5: **end if**

6: $b \leftarrow ta_1$.

7: $i \leftarrow 2(ab)b$.

8: $x \leftarrow (ab)(i-1)$.

9: **return** x.

The computational cost of Algorithm 3 is one exponentiation, four multiplications and two squarings in $\mathbb{F}_q$. Han *et al.* [17] show that the exponent $q - 5/8$ can be rewritten in base p as,

$$\frac{q-5}{8} = \alpha + p\left[p\alpha + (5\alpha + 3)\right] \sum_{i=0}^{(m-3)/2} p^{2i}, \tag{4}$$

where $\alpha = \frac{p-5}{8}$.

Using the factorization of Eq. (4), it can be shown that the average complexity of Algorithm 3 when a is a square, is giving as (see Appendix B for details),

$$\left[\frac{1}{2}\lfloor \log_2(p) \rfloor + \frac{3}{2}\lfloor \log_2(m) \rfloor + 3\right] M_q + \lfloor \log_2(p) \rfloor S_q + \left[\frac{3}{2}\lfloor \log_2(m) \rfloor + 2\right] F_q. \tag{5}$$

2) Generalized Atkin's algorithm: The Atkin's method was generalized at first by Müller [25] for the case $q \equiv 9 \pmod{16}$. Müller showed that for this case the square root computation for a quadratic residue can be achieved at a cost of two exponentiations in $\mathbb{F}_q$. Later, Kong *et al.* [22] further improve that result by presenting a procedure that required only one exponentiation for half of the squares in $\mathbb{F}_q$, and two exponentiations for the remainder half. Nonetheless, by pre-computing some values, one can take a square root at the cost of only one exponentiation as shown in Algorithm 4.

Algorithm 4 Kong *et al.* algorihtm when $q \equiv 9 \pmod{16}$

Require: $a \in \mathbb{F}_q^*$.

Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.

PRECOMPUTATION

1: $c_0 \leftarrow 1$
2: **while** $c_0 = 1$ **do**
3: Select randomly $c \in \mathbb{F}_q^*$.
4: $c_0 \leftarrow \chi_q(c)$.
5: **end while**
6: $d \leftarrow c^{\frac{q-9}{8}}$,
7: $e \leftarrow c^2$, $t \leftarrow 2^{\frac{q-9}{16}}$.

COMPUTATION

1: $a_1 \leftarrow a^{\frac{q-9}{16}}$.
2: $a_0 \leftarrow (a_1^2 a)^4$.
3: **if** $a_0 = -1$ **then**
4: **return** false.
5: **end if**
6: $b \leftarrow ta_1$.
7: $i \leftarrow 2(ab)b$.
8: $r \leftarrow i^2$.
9: **if** $r = -1$ **then**
10: $x \leftarrow (ab)(i-1)$.
11: **else**
12: $u \leftarrow bd$.
13: $i \leftarrow 2u^2ea$.
14: $x \leftarrow uca(i-1)$.
15: **end if**
16: **return** x.

The computational cost of Algorithm 4 is one exponentiation, six and a half multiplications, and four and a half squarings in $\mathbb{F}_q$. For this case, the exponent $(q-9)/16$ can be rewritten in base p as,

$$\frac{q-9}{16} = \alpha + p\left[p\alpha + (9\alpha + 5)\right] \sum_{i=0}^{(m-3)/2} p^{2i}, \tag{6}$$

where $\alpha = \frac{p-9}{16}$.

Using the factorization of Eq. (6), it can be shown that the average complexity of Algorithm 4 when a is a square, is given as (see Appendix B for details),

$$\left[\frac{1}{2}\lfloor \log_2(p) \rfloor + \frac{3}{2}\lfloor \log_2(m) \rfloor + 10\right] M_q + \left[\lfloor \log_2(p) \rfloor + \frac{5}{2}\right] S_q + \left[\frac{3}{2}\lfloor \log_2(m) \rfloor + 2\right] F_q. \tag{7}$$

3) General square root algorithms in $\mathbb{F}_q$ for $q \equiv 1 \pmod{16}$: This sub-case is certainly the most costly, since there is no specialized algorithm to tackle it. The Tonelli-Shanks's [28], [30] and the Cipolla-Lehmer's [12] algorithms are the two general non-deterministic algorithms from which most of the methods for square root extraction are derived. In this subsection the Tonelli-Shank's algorithm and an improved Cipolla-Lehmer algorithm by Müller [25] are described. For the latter, we include a detailed analysis of its computational complexity that to the best of our knowledge, has not been reported before in the open literature.

Algorithm 5 Tonelli-Shanks Algorithm

Require: $a \in \mathbb{F}_q^*$

Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.

PRECOMPUTATION

1: Write $q-1 = 2^s t$, where t is odd.
2: $c_0 \leftarrow 1$.
3: **while** $c_0 = 1$ **do**
4: Select randomly $c \in \mathbb{F}_q^*$.
5: $z \leftarrow c^t$.
6: $c_0 \leftarrow c^{2^{s-1}}$.
7: **end while**

COMPUTATION

1: $\omega \leftarrow a^{\frac{t-1}{2}}$.
2: $a_0 \leftarrow (\omega^2 a)^{2^{s-1}}$.
3: **if** $a_0 = -1$ **then**
4: **return** false.
5: **end if**
6: $v \leftarrow s$, $x \leftarrow a\omega$, $b \leftarrow x\omega$.
7: **while** $b \neq 1$ **do**
8: Find least integer $k \geq 0$ such that $b^{2^k} = 1$.
9: $\omega \leftarrow z^{2^{v-k-1}}$, $z \leftarrow \omega^2$, $b \leftarrow bz$, $x \leftarrow x\omega$, $v \leftarrow k$.
10: **end while**
11: **return** x.

Algorithm 5 presents a variant of the Tonelli-Shanks procedure where the quadratic test of an arbitrary field element $a \in \mathbb{F}_q$ has been incorporated to the algorithm. It is noticed that the computational complexity of Algorithm 5 varies depending if the input is or not a quadratic residue in $\mathbb{F}_q$. By taking into account the average contribution of QR and QNR inputs, and using the complexity analysis given in [23] for the classical Tonelli-Shanks algorithm, it is not difficult to see that the average computational cost of Algorithm 5 is given as,

$$\frac{1}{2}\Big[\lfloor \log_2(q) \rfloor + 4\Big] M_q + \Big[\lfloor \log_2(q) \rfloor + \frac{1}{8}\big(s^2 + 3s - 16\big) + \frac{1}{2^s}\Big] S_q. \tag{8}$$

However, rewriting the exponent $(t-1)/2$ in base p as,

$$\frac{t-1}{2} = \alpha + p\left[\alpha(p+1) + 1 + 2^{s-1}t\right] \sum_{i=0}^{(m-3)/2} p^{2i},$$

where $q - 1 = 2^s t$, $p - 1 = 2^s x$, and $\alpha = \frac{x-1}{2}$, it can be shown that the average complexity of Algorithm 5 for any arbitrary field element a is given as (see Appendix B for details),

$$\begin{aligned} \text{Tonelli-Shanks Alg. cost} \;=\; & \left[\frac{1}{2}\lfloor \log_2(p) \rfloor + \frac{3}{2}\lfloor \log_2(m) \rfloor + \frac{s}{2} + 5\right] M_q \\ & + \left[\lfloor \log_2(p) \rfloor + \frac{1}{8}(s^2 + 11s - 16) + \frac{1}{2^s}\right] S_q + \left[\frac{3}{2}\lfloor \log_2(m) \rfloor + 2\right] F_q. \end{aligned}$$

Algorithm 6 Lucas sequence evaluation

Require: $\alpha \in \mathbb{F}_q$ and $k \geq 2$.
Ensure: $V_k(\alpha, 1)$.
1: Write $k = \sum_{j=0}^{l-1} b_j 2^j$ in binary form.
2: $d_0 \leftarrow \alpha$.
3: $d_1 \leftarrow \alpha^2 - 2$.
4: **for** j from $l-2$ down to 1 **do**
5: $\quad d_{1-b_j} \leftarrow d_0 d_1 - \alpha$, $d_{b_j} \leftarrow d_{1-b_j}^2 - 2$.
6: **end for**
7: **if** $b_0 = 1$ **then** $v \leftarrow d_0 d_1 - \alpha$ **else** $v \leftarrow d_0^2 - 2$.
8: **return** v.

As a second option for this sub-case, the improved Cipolla-Lehmer algorithm introduced in [25], uses the Lucas sequences to compute a square root over the field $\mathbb{F}_q$. Thus, we first briefly recall the definition of the Lucas sequences and subsequently give a fast algorithm that evaluates the k-th element of some instances of these sequences. For $(\alpha, \beta) \in \mathbb{F}_q$, the Lucas sequence $\Big(V_k(\alpha, \beta)\Big)_{k \geq 0}$ is defined as,

$$V_0 = 2, \quad V_1 = \alpha \ \text{ and } \ V_k = \alpha V_{k-1} - \beta V_{k-2}, \ \text{for } k > 1.$$

Algorithm 6 computes $V_k(\alpha, 1)$, for a given $\alpha \in \mathbb{F}_q$ and $k > 1$. It can be easily verified that to compute $V_k(\alpha, 1)$, this procedure requires about $(\lfloor \log_2(k) \rfloor + \frac{3}{2})S_q + (\lfloor \log_2(k) \rfloor + \frac{1}{2})M_q$ multiplications in $\mathbb{F}_q$.

Algorithm 7 Müller's algorithm [25]

Require: $a \in \mathbb{F}_q^*$.
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.
1: **if** $a = 4$ **then**
2: $\quad$ **return** 2.
3: **end if**
4: $t \leftarrow 1$.
5: $a_1 \leftarrow \chi_q(at^2 - 4)$.
6: **while** $a_1 = 1$ **do**
7: $\quad$ Select randomly $u \in \mathbb{F}_q^* \backslash \{1\}$.
8: $\quad t \leftarrow u$.
9: $\quad$ **if** $at^2 - 4 = 0$ **then**
10: $\quad\quad$ **return** $2t^{-1}$.
11: $\quad$ **end if**
12: $\quad a_1 \leftarrow \chi_q(at^2 - 4)$.
13: **end while**
14: $\alpha \leftarrow at^2 - 2$.
15: $x \leftarrow V_{\frac{q-1}{4}}(\alpha, 1)/t$.
16: $a_0 \leftarrow x^2 - a$.
17: **if** $a_0 \neq 0$ **then**
18: $\quad$ **return** false.
19: **end if**
20: **return** x.

Algorithm 7 shows essentially the same square root algorithm as it was presented in [25]. In order to assess the computational complexity of this procedure, the following two auxiliary lemmas are presented.

Lemma 1: In the field $\mathbb{F}_q$, the number of QR $a \in \mathbb{F}_q^*$ such that $a-4$ is a QNR is $\frac{q-1}{4}$.

Proof:

To prove this one can at first compute the number of QR $a \in \mathbb{F}_q^*$ such that $a-4$ is a QR, which is clearly half of the number of $b \in \mathbb{F}_q^*$ such that b^2-4 is a QR.

It was shown in [31, Lemma 3.1] that $\#\{b \in \mathbb{F}_q \mid b^2-4 \text{ is a QR in } F_q\} = \frac{q+1}{2}$. Now, when $b=0$, -4 is a QR in $\mathbb{F}_q$ since $q \equiv 1 \pmod 4$, thus we have $\#\{b \in \mathbb{F}_q^* \mid b^2-4 \text{ is a QR in } F_q\} = \frac{q-1}{2}$, and then $\#\{a \in \mathbb{F}_q^* \mid a \text{ and } a-4 \text{ are QRs in } F_q\} = \frac{q-1}{4}$. Hence, the number of QR $a \in \mathbb{F}_q^*$ such that $a-4$ is a QNR is $\frac{q-1}{2} - \frac{q-1}{4} = \frac{q-1}{4}$.

■

Lemma 2: Let $a \in \mathbb{F}_q^*$ be a QR, then the number of $t \in \mathbb{F}_q^*$ such that at^2-4 is a QNR is $\frac{q-1}{2}$.

Proof:

As in the proof of Lemma 1, let us start by computing the number of $t \in \mathbb{F}_q^*$ such that at^2-4 is a QR, i.e the number of $t \in \mathbb{F}_q^*$ such that there exists $s \in \mathbb{F}_q$ and $at^2-4=s^2$. For such a t, $at^2-4=s^2$ is equivalent to $a-4r^2=s^2r^2$, where $r=t^{-1}$, and then to $a=(s^2+4)r^2$. Thus the number of these t is equal to the number of $r \in \mathbb{F}_q$ such that there exist $s \in \mathbb{F}_q$ and $a=(s^2+4)r^2$.

Claim: The number of the above r's is the double of the number of QRs $c \in \mathbb{F}_q^*$ such that $c-4$ is also a QR in $\mathbb{F}_q$.

Indeed, suppose that we have such a c, let $s=\pm\sqrt{c-4}$, then $s^2+4=c$.

Hence, it can be seen that for each such c, one obtains two solutions for the equation $a=(s^2+4)r^2$, namely, $r_{1,2}=\pm\sqrt{a/(s^2+4)}$. Moreover, since for a different c' with properties as for c, this procedure gives two elements (r_1', r_2') with $(r_1', r_2') \neq (r_1, r_2)$ and $(r_1', r_2') \neq (r_2, r_1)$, in addition to the fact that the above procedure is reversible, one can conclude that:

$\#\{r \in \mathbb{F}_q \mid \exists\, s \in \mathbb{F}_q \text{ and } a=(s^2+4)r^2\} = 2\#\{c \in \mathbb{F}_q^* \mid c \text{ and } c-4 \text{ are QRs in } \mathbb{F}_q\}$.

Recalling from the proof of Lemma 1, we have $\#\{c \in \mathbb{F}_q^* \mid c-4 \text{ is a QR in } F_q\} = \frac{q-1}{4}$, and therefore $\#\{t \in \mathbb{F}_q^* \mid at^2-4 \text{ is a QR in } F_q\} = \frac{q-1}{2}$. Hence, the number of $t \in \mathbb{F}_q^*$ such that at^2-4 is a QNR is $q-1-\frac{q-1}{2} = \frac{q-1}{2}$. ■

Summarizing, Lemma 1 shows that for half of the QRs in $\mathbb{F}_q^*$, there is no need to search for a t in the main loop of Algorithm 7, and Lemma 2 ensures that for the remainder case only 2 iterations in the while-loop suffice on average. Now, the expected number of multiplications and squarings in the cases when $(a-4)^{\frac{q-1}{2}} = -1$ and $(a-4)^{\frac{q-1}{2}} = 1$, can be computed. Let I_q

denote the complexity of computing an inverse in $\mathbb{F}_q$. Then,

- If $(a-4)^{\frac{q-1}{2}} = -1$, on average, one has to compute one exponentiation and one Lucas sequence evaluation.
- If $(a-4)^{\frac{q-1}{2}} = 1$, on average, one has to compute three exponentiations, one Lucas sequence evaluation, one inversion, two multiplications and two squarings.

Once again, notice that the exponentiation of step 5 can be optimized by rewriting the exponent $(q-1)/2$ as,

$$\frac{q-1}{2} = \frac{p-1}{2} \sum_{i=0}^{(m-1)} p^i,$$

which gives an expected computational cost of Algorithm 7 over all QRs in $\mathbb{F}_q$ as (see appendix B for details),

$$\begin{aligned} \text{Müller Alg. cost} \;=\; & \left[\lfloor \log_2(q) \rfloor + \frac{15}{4}\lfloor \log_2(m) \rfloor + \frac{13}{4}\right] M_q + \left[\lfloor \log_2(q) \rfloor - \frac{1}{2}\right] S_q \\ & + \left[\frac{15}{4}\lfloor \log_2(m) \rfloor + \frac{17}{4}\right] F_q + \left[\lfloor \log_2(p) \rfloor - 3\right] M_p + \left[2\lfloor \log_2(p) \rfloor - 2\right] S_p + \frac{1}{2} I_p \end{aligned} \tag{9}$$

V. Square roots in even extension fields

Even extension fields $\mathbb{F}_{q^2}$ with $q = p^n$ and $n \geq 1$, can be constructed as, $\mathbb{F}_{q^2} \cong \mathbb{F}_q[y]/\left(y^2 - \beta\right)$, where $\beta \in \mathbb{F}_q$, is not a square. Unfortunately, none of the methods studied in the previous section lead to efficient computation of square roots for even extension fields as is briefly discussed next.

Notice that in this scenario, the identity $q^2 \equiv 1 \pmod 4$ always holds. Moreover, it is easy to see that the case $q^2 \equiv 5 \pmod 8$, can never occur. This automatically implies that the Shanks and the Atkin methods studied in the previous Section are both ruled out. In the case that $q^2 \equiv 9 \pmod{16}$, one can use the generalized Atkin's algorithm by Kong et al. that was also reviewed in the precedent Section. If however, $q^2 = 1 \pmod{16}$, the only remaining option is to select between either the Tonelli-Shanks's or the Müller's non-deterministic algorithms.

In the rest of this section, three efficient methods for computing square roots over even extension fields will be discussed. First, a detailed analysis of the complex method described in [27] will be given. Then, two novel algorithms for computing square roots in $\mathbb{F}_{q^2}$ will be presented. These two algorithms are complementary in the sense that they cover separately the two congruence classes that odd primes define, namely, $q \equiv 1 \pmod 4$ and $q \equiv 3 \pmod 4$. The

easiest case $q \equiv 3 \pmod 4$ is first presented followed by the slightly more involved case when $q \equiv 1 \pmod 4$.

Algorithm 8 Complex method for square root computation over $\mathbb{F}_{q^2}$

Require: Irreducible binomial $f(y) = y^2 - \beta$ such that $\mathbb{F}_{q^2} \cong \mathbb{F}_q[y]/\left(y^2 - \beta\right)$, $\beta \in \mathbb{F}_q$, with $q = p^n$, $a = a_0 + a_1 y \in \mathbb{F}_{q^2}^*$.
Ensure: If it exists, $x = x_0 + x_1 y \in \mathbb{F}_{q^2}$ satisfying $x^2 = a$, false otherwise.
1: **if** $a_1 = 0$ **then**
2: **return** $\mathrm{SQRT}_q(a_0)$.
3: **end if**
4: $\alpha \leftarrow a_0^2 - \beta \cdot a_1^2$.
5: $\gamma \leftarrow \chi_q(\alpha)$.
6: **if** $\gamma = -1$ **then**
7: **return** false.
8: **end if**
9: $\alpha \leftarrow \mathrm{SQRT}_q(\alpha)$.
10: $\delta \leftarrow \frac{a_0+\alpha}{2}$.
11: $\gamma \leftarrow \chi_q(\delta)$.
12: **if** $\gamma = -1$ **then**
13: $\delta \leftarrow \frac{a_0-\alpha}{2}$.
14: **end if**
15: $x_0 \leftarrow \mathrm{SQRT}_q(\delta)$.
16: $x_1 \leftarrow \frac{a_1}{2x_0}$.
17: $x \leftarrow x_0 + x_1 y$.
18: **return** x.

A. The complex method

Let the quadratic extension field be defined as, $\mathbb{F}_{q^2} \cong \mathbb{F}_q[y]/\left(y^2 - \beta\right)$, where $\beta \in \mathbb{F}_q$, is not a square, with $q = p^n$, $n \geq 1$. Then, a square root $x = x_0 + x_1 y \in \mathbb{F}_{q^2}$ of an arbitrary quadratic residue $a = a_0 + a_1 y \in \mathbb{F}_{q^2}^*$ can be found by observing that since $x^2 = {x_0}^2 + 2x_0x_1y + \beta {x_1}^2$, then x_0, x_1, must satisfy the following two equations

$$\begin{cases} {x_0}^2 + \beta {x_1}^2 = & a_0 \\ 2x_0x_1 = & a_1 \end{cases}$$

Solving this system of equations for x_0, and x_1 yields,

$$\begin{aligned} x_0 &= \left(\frac{a_0 \pm ({a_0}^2 - \beta {a_1}^2)^{\frac{1}{2}}}{2}\right)^{\frac{1}{2}} \\ x_1 &= \frac{a_1}{2x_0} \end{aligned} \tag{10}$$

Observe that $a = a_0 + a_1 y$, will be a quadratic residue in the quadratic extension, whenever $\alpha = {a_0}^2 - \beta {a_1}^2 \in \mathbb{F}_q$ is a quadratic residue over $\mathbb{F}_q$, as can be easily checked by noticing that,

$$\begin{aligned}(a_0 + a_1 y)^{\frac{q^2-1}{2}} &= \left((a_0 + a_1 y)^{q+1}\right)^{\frac{q-1}{2}} \\ &= \left((a_0 - a_1 y) \cdot (a_0 + a_1 y)\right)^{\frac{q-1}{2}} \\ &= \left({a_0}^2 - \beta {a_1}^2\right)^{\frac{q-1}{2}}\end{aligned}$$

Alg 8 uses the complex method for computing a square root in the quadratic extension $\mathbb{F}_{q^2}$ by computing $x = x_0 + x_1 y$ according to Eq. (10). Notice that Alg 8 performs two quadratic residue tests in steps 5 and 11, which can be computed efficiently by using the method described in §III. Besides these two tests, the cost of Alg 8 includes the computation of two square roots plus one field inversion over $\mathbb{F}_q$.

B. A deterministic algorithm when $q \equiv 3 \pmod 4$

A technique to compute a square root of a quadratic residue $a \in \mathbb{F}_{q^2}$ is to find an element $b \in \mathbb{F}_{q^2}$ for which there exists an odd integer s such that $b^2 a^s = 1$. In this case, a square root of a is given by $b a^{\frac{s+1}{2}}$. In order to find b and s with the above property, we proceed as follows.

Let b and s be defined as, $b = (1 + a^{\frac{q-1}{2}})^{\frac{q-1}{2}}$ and $s = \frac{q-1}{2}$. Let us consider first the case when $b \neq 0$. Then, it can be easily verified that the equality $b^2 a^s = 1$ holds since,

$$\begin{aligned}b^2 a^s &= (1 + a^{\frac{q-1}{2}})^{(q-1)} a^{\frac{q-1}{2}} \\ &= (1 + a^{\frac{q-1}{2}})^{q} (1 + a^{\frac{q-1}{2}})^{(-1)} a^{\frac{q-1}{2}} \\ &= (1 + a^{\frac{q-1}{2}q}) (1 + a^{\frac{q-1}{2}})^{(-1)} a^{\frac{q-1}{2}} \\ &= (a^{\frac{q-1}{2}} + a^{\frac{q-1}{2}(q+1)}) (1 + a^{\frac{q-1}{2}})^{(-1)} \\ &= (a^{\frac{q-1}{2}} + 1) (1 + a^{\frac{q-1}{2}})^{(-1)} \\ &= 1\end{aligned}$$

If on the contrary, $b = 0$, then by definition of b we have that $1 + a^{\frac{q-1}{2}} = 0$ and hence $a^{\frac{q-1}{2}} = -1$. In this case $x = i a^{\frac{q+1}{4}}$ is a square root of a, where $i = \sqrt{-1}$, as it can be easily verified by noticing that $x^2 = i^2 a^{\frac{q+1}{2}} = i^2 a^{\frac{q-1}{2}} a = (-1)(-1)a = a$.

In practice the value of i can be readily found, if the quadratic field extension $\mathbb{F}_{q^2}$ has been constructed using the binomial $f(y) = y^2 - \beta$, where $\beta \in \mathbb{F}_q$ is not a square. In this case, $i = \beta^{\frac{q-3}{4}} y$, yields $i^2 = \beta^{\frac{q-3}{2}} y^2 = \beta^{\frac{q-3}{2}} \beta = \beta^{\frac{q-1}{2}} = -1$, as required. However, since $p \equiv 3 \pmod 4$, typically $\beta = -1$ and therefore $i = y$.

Summarizing, the square root x of a quadratic residue $a \in \mathbb{F}_{q^2}$, with $q \equiv 3 \pmod 4$ can be found as,

$$x = \begin{cases} i a^{\frac{q+1}{4}} & \text{if } a^{\frac{q-1}{2}} = -1, \\ \left(1 + a^{\frac{q-1}{2}}\right)^{\frac{q-1}{2}} a^{\frac{q+1}{4}} & \text{otherwise.} \end{cases} \tag{11}$$

We remark the striking similarity that exists between the classic Shanks algorithm (see § IV) and our method. This leads us to state that Eq. (11) can be seen as a generalization of the Shanks algorithm for even extension fields.

Algorithm 9 Square root computation over $\mathbb{F}_{q^2}$, with $q \equiv 3 \pmod 4$

Require: $a \in \mathbb{F}_{q^2}^*$, $i \in \mathbb{F}_{q^2}$, such that $i = \sqrt{-1}$, with $q = p^n$.
Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.
1: $a_1 \leftarrow a^{\frac{q-3}{4}}$.
2: $\alpha \leftarrow a_1(a_1 a)$.
3: $a_0 \leftarrow \alpha^q \alpha$.
4: **if** $a_0 = -1$ **then**
5: **return** false.
6: **end if**
7: $x_0 \leftarrow a_1 a$.
8: **if** $\alpha = -1$ **then**
9: $x \leftarrow i x_0$.
10: **else**
11: $b \leftarrow (1 + \alpha)^{\frac{q-1}{2}}$.
12: $x \leftarrow b x_0$.
13: **end if**
14: **return** x.

Algorithm 9 shows an efficient procedure for computing square roots according to Eq.(11). After executing Steps 1-3 the variables α and a_0 are assigned as, $\alpha = a^{(q-1)/2}$ and $a_0 = a^{(q^2-1)/2}$, respectively. Therefore, in Steps 4-6 the quadratic residue test of a over $\mathbb{F}_{q^2}$ is performed. In the case that a is not a square the algorithm returns 'false'. Otherwise, after executing Step 7, the variable x_0 is assigned as $x_0 = a^{(q+1)/4}$. Then, according to Eq.(11), if in Step 8 it is determined that $\alpha = -1$, the square root of a is given as $x = i x_0$. Otherwise in Step 11, b is computed as, $b = \left(1 + a^{\frac{q-1}{2}}\right)^{\frac{q-1}{2}}$, and the value of the square root of a is computed in Step 12 as, $x = b x_0$.

Algorithm 9 performs at most two exponentiations in $\mathbb{F}_{q^2}$,in Steps 1 and 11. Additionally, in Steps 2, 3, 7 and 12 a total of five multiplications in $\mathbb{F}_{q^2}$ are required. As we have seen in§ IV,

the exponent $(q-3)/4$ of Step 1 can be written in terms of p as,

$$\frac{q-3}{4} = \alpha + p\left[p\alpha + (3\alpha+2)\right] \sum_{i=0}^{(n-3)/2} p^{2i},$$

where $\alpha = \frac{p-3}{4}$. Similarly, the exponent of $(q-1)/2$ of Step 11 can be written in terms of p as, $\frac{q-1}{2} = \frac{p-1}{2} + \sum_{i=0}^{n-1} p^i$.

Hence, the exponentiation $a^{\frac{q-3}{4}}$ can be computed by performing the exponentiation $a^{\frac{p-3}{4}}$, plus 4 multiplications, one squaring and two Frobenius over $\mathbb{F}_{q^2}$ plus one evaluation of the sequence $C_{q^2}(\frac{n-1}{2}, 2)$ that can be recursively computed using Algorithm 11 of Appendix A. The average cost of computing $a^{\frac{p-3}{4}}$ and $C_{q^2}(\frac{n-1}{2}, 2)$ is $(\frac{1}{2}\lfloor\log_2(p)\rfloor - \frac{3}{2})M_{q^2} + (\lfloor\log_2(p)\rfloor - 2)S_{q^2}$, and $\frac{3}{2}\lfloor\log_2 n\rfloor(M_{q^2} + F_{q^2})$, respectively. Similarly, the exponentiation $(1+\alpha)^{(q-1)/2}$ of Step 11 can be computed by performing the exponentiation $a^{\frac{p-3}{4}}$, plus 1 multiplication plus one evaluation of the sequence $C_{q^2}(n, 1)$. Therefore, the overall average computational cost associated to Algorithm 9 when a is a square is given as,

$$\text{Alg. 9 cost } = \left[\lfloor\log_2(p)\rfloor + 3\lfloor\log_2 n\rfloor + 7\right] M_{q^2} + \left[2\lfloor\log_2(p)\rfloor - 2\right] S_{q^2} + \left[3\lfloor\log_2 n\rfloor + 4\right] F_{q^2}$$

C. A descending algorithm when $q \equiv 1 \pmod 4$

The main idea of Algorithm 10 is to descend the square root problem from $\mathbb{F}_{q^2}$ to $\mathbb{F}_q$ with one exponentiation of $log_2(q)$ bits plus some precomputations. Once again, let us consider the problem of finding the square root of an arbitrary quadratic residue $a \in \mathbb{F}_{q^2}$. The approach of descending this problem from $\mathbb{F}_{q^2}$ to $\mathbb{F}_q$ can be achieved by the opportunistic usage of the identity,

$$\begin{aligned} a &= a\left(a^{\frac{q-1}{2}}\right)^{q+1} \\ &= a\left(a^{\frac{q-1}{2}}\right)^{q} a^{\frac{q-1}{2}} \\ &= \left(a^{\frac{q-1}{2}}\right)^{q} a^{\frac{q+1}{2}}, \end{aligned} \tag{12}$$

where the equality in Eq. (12) holds because of $a^{\frac{q^2-1}{2}} = 1$, since a in a quadratic residue in $\mathbb{F}_{q^2}$. Then, by taking the square root in both sides of Eq. (12) we get,

$$\sqrt{a} = \pm\left(a^{\frac{q-1}{4}}\right)^{q} \sqrt{a^{\frac{q+1}{2}}}. \tag{13}$$

Now, since $\left(a^{\frac{q+1}{2}}\right)^{q-1} = a^{\frac{q^2-1}{2}} = 1$, by the Fermat's little theorem, the value $a^{\frac{q+1}{2}}$ lives in $\mathbb{F}_q$. Moreover, if $a^{\frac{q+1}{2}}$ is a quadratic residue in $\mathbb{F}_q$, then it holds that $\left(a^{\frac{q+1}{2}}\right)^{\frac{q-1}{2}} = 1$. This implies that the problem of finding square roots in $\mathbb{F}_{q^2}$ has been reduced to the same problem but in the sub-field F_q, after one exponentiation with an exponent of roughly the same size of q. In the event that $a^{\frac{q+1}{2}}$ is not a quadratic residue, then finding a quadratic non-residue in $\mathbb{F}_{q^2}$ (independently of the form of a) allows us to recover easily the previous case as given in Algorithm 10.

Algorithm 10 Square root computation over $\mathbb{F}_{q^2}$, with $q \equiv 1 \pmod 4$

Require: $a \in \mathbb{F}_{q^2}^*$, with $q = p^n$, $n \geq 1$.

Ensure: If it exists, x satisfying $x^2 = a$, false otherwise.

PRECOMPUTATION

1: $c_0 \leftarrow 1$.
2: **while** $c_0 = 1$ **do**
3: Select randomly $c \in \mathbb{F}_{q^2}^*$.
4: $c_0 \leftarrow \chi_{q^2}(c)$.
5: **end while**
6: $d \leftarrow c^{\frac{q-1}{2}}$.
7: $e \leftarrow (dc)^{-1}$.
8: $f \leftarrow (dc)^2$.

COMPUTATION

1: $b \leftarrow a^{\frac{q-1}{4}}$.
2: $a_0 \leftarrow (b^2)^q b^2$.
3: **if** $a_0 = -1$ **then**
4: **return** false.
5: **end if**
6: **if** $b^q b = 1$ **then**
7: $x_0 \leftarrow \mathrm{SQRT}_q(b^2 a)$.
8: $x \leftarrow x_0 b^q$.
9: **else**
10: $x_0 \leftarrow \mathrm{SQRT}_q(b^2 af)$.
11: $x \leftarrow x_0 b^q e$.
12: **end if**
13: **return** x.

Theorem 1: Algorithm 10 computes a square root of $a \in \mathbb{F}_{q^2}$ with one exponentiation of $log_2(q)$ bits in $\mathbb{F}_{q^2}$ and one square root computation in the field $\mathbb{F}_q$.

Proof:

At Step 2 of the computation phase, the value of a_0 is,

$$(b^2)^q b^2 = (b^2)^{q+1} = \left[(a^{\frac{q-1}{4}})^2\right]^{q+1} = (a^{\frac{q-1}{2}})^{q+1},$$

which corresponds to the quadratic residue test of a in the field $\mathbb{F}_{q^2}$. Thus, if $a_0 = -1$, a is a non quadratic residue in $\mathbb{F}_{q^2}$ and then 'false' is returned. In the discussion that follows, it will be assumed that a is a a quadratic residue ($a_0 = 1$).

At Step 6, it is tested whether $b^q b = b^{q+1} = (a^{\frac{q+1}{2}})^{\frac{q-1}{2}}$ is or not one. If it is one, then it is concluded that $a^{\frac{q+1}{2}}$ is a quadratic residue in $\mathbb{F}_q$. For $b^q b = 1$, at Step 7, a square root x_0 of

$b^2a = a^{\frac{q+1}{2}}$ in $\mathbb{F}_q$ is computed. Then the square root of a is given by $x = x_0 b^q$, since,

$$x^2 = x_0^2 b^{2q} = a^{\frac{q+1}{2}}(a^{\frac{q-1}{4}})^{2q} = aa^{\frac{q-1}{2}}(a^{\frac{q-1}{2}})^q = a(a^{\frac{q-1}{2}})^{q+1} = aa_0 = a.$$

Now, let us assume that $b^q b = -1$. Notice that since in the precomputation phase, c_0 was selected as a QNR, then $d^q d = d^{q+1} = c^{\frac{q^2-1}{2}}$ is also a QNR. At Step 10, it is easy to see that the value b^2af lies in $\mathbb{F}_q$ where it is also a square. To see this, notice that,

$$(b^2af)^{\frac{q-1}{2}} = b^{q-1}a^{\frac{q-1}{2}}(dc)^{q-1} = b^{q-1}b^2d^{q-1}d^2 = (b^q b)(d^q d) = (-1)(-1) = 1.$$

After computing a square root x_0 of b^2af in $\mathbb{F}_q$, it can be proved that $x = x_0 b^q e$ is a square root of a since, $x^2 = (x_0 b^q e)^2 = b^2 af b^{2q} e^2 = ab^{2q+2}(dc)^2\left[(dc)^{-1}\right]^2 = ab^{2q+2} = aa_0 = a$.

■

The cost of Algorithm 10 includes the computation of one field exponentiation over $\mathbb{F}_{q^2}$, one square root in $\mathbb{F}_q$, 5 field multiplications, squarings and two Frobenius over $\mathbb{F}_{q^2}$. The exponent $(q-1)/4$ of Step 1 can be written in terms of p as, $\frac{q-1}{4} = \frac{p-1}{4} + \sum_{i=0}^{n-1} p^i$. Hence, $a^{(q-1)/4}$ can be computed by performing the exponentiation $a^{\frac{p-1}{4}}$, plus 1 multiplication plus one evaluation of the sequence $C_{q^2}(n, 1)$. Therefore, the overall average computational cost associated to Algorithm 10 when a is a square is given as,

Alg. 10 cost $= (\frac{1}{2}\lfloor\log_2(p)\rfloor + \frac{3}{2}\lfloor\log_2 m\rfloor + \frac{11}{2})M_{q^2} + (\lfloor\log_2(p)\rfloor - 2)S_{q^2} + (\frac{3}{2}\lfloor\log_2 m\rfloor + 3)F_{q^2} + SQRT_q$

VI. Comparisons

In this section, we compare the algorithms described above for the cases when one wants to compute square roots in $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$, where p is an odd prime. In the next experiments, two group of primes have been considered. The first group is composed by primes congruent to $3 \pmod 4$ where algorithm 9 apply. The second one considers primes $p \equiv 1 \pmod 4$, where one can use algorithm 10. The extensions $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$ are obtained by constructing the following field towering,

$$\mathbb{F}_p \subset \mathbb{F}_{p^3} \subset \mathbb{F}_{p^6} \subset \mathbb{F}_{p^{12}}.$$

In our comparisons, BN curve primes [6] and NIST recommended primes for elliptic curve cryptography [1] were selected. This choice was taken considering that one of the contexts

where computing square root over prime extensions fields is required is in the computation of bilinear pairings and elliptic curve cryptography. It is worth mentioning that BN curves is a rich family of elliptic curves defined over a prime field $\mathbb{F}_p$, where p is parametrized as, $p(u) = 36u^4 + 36u^3 + 24u^2 + 6u + 1$, with $u \in \mathbb{Z}$.

Table I

NUMBER OF OPERATIONS IN $\mathbb{F}_p$ FOR SQUARE ROOTS IN $\mathbb{F}_{q^2}$, $q = p$, $p \equiv 3 \pmod 4$

Parameter		$u = -(2^{62} + 2^{55} + 1)$	$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	$u = 2^{63} + 2^9 + 2^8 + 2^6 + 2^4 + 2^3 + 1$
Bit length of p		254	256	258
Algo 8	M_p	1261	1785	1427
	Mc_p	1091	1271	1157
	I_p	0	0	0
Complex Alg.	M_p	1176	1292	1528
	Mc_p	4	4	4
	I_p	1	1	1
Tonelli-Shanks	M_p	1574	6292	1660
	Mc_p	1202	6999	1244
	I_p	0	0	0
Müller's Algo	M_p	3120	3387	3245
	Mc_p	1521	1537	1546
	I_p	1	1	1

Table II

NUMBER OF OPERATIONS IN $\mathbb{F}_p$ FOR SQUARE ROOTS IN $\mathbb{F}_{q^2}$, $q = p$, $p \equiv 1 \pmod 4$

Parameter		$p = 2^{224} - 2^{96} + 1$	$u = 2^{62} - 2^{54} + 2^{44}$	$u = 2^{63} - 2^{49}$
Bit length of p		224	254	256
Algo 9	M_p	1975	1625	1782
	Mc_p	577	591	603
	I_p	1	0	0
Complex Alg.	M_p	2284	2425	2691
	Mc_p	5	5	5
	I_p	3	1	1
Tonelli-Shanks	M_p	6705	2934	3199
	Mc_p	6065	2402	2669
	I_p	0	0	0
Müller's Algo	M_p	2743	3197	3254
	Mc_p	1342	1521	1545
	I_p	1	1	1

For comparisons over $\mathbb{F}_{p^2}$, the quadratic extension was constructed as $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \delta)$, where δ is a quadratic non-residue over $\mathbb{F}_p$. Hence, every element a in $\mathbb{F}_{p^2}$ is represented as $a = a_0 + a_1 u$ with an arithmetic cost given as, $M_{p^2} = 3M_p + 1Mc_p$, $S_{p^2} = 2S_p + 2Mc_p$, $I_{p^2} = I_p + 4M_p + Mc_p$. Analogous costs also hold for the quadratic extensions $\mathbb{F}_{p^3} \subset \mathbb{F}_{p^6}$ and $\mathbb{F}_{p^6} \subset \mathbb{F}_{p^{12}}$.

The cubic extension $\mathbb{F}_p \subset \mathbb{F}_{p^3}$ is obtaining by considering a cubic non-residue $\xi \in \mathbb{F}_p$. We chose $p \equiv 1 \pmod 3$ in order to have a simple way for finding cubic non-residues, since in this case an element $\xi \in \mathbb{F}_p$ is a cubic non-residue iff $\xi^{\frac{p-1}{3}} \neq 1$.

Let $\xi \in \mathbb{F}_p$ be a cubic non-residue, then the polynomial $X^3 - \xi$ is irreducible over $\mathbb{F}_p$ so that the quotient $\mathbb{F}_p[u]/(u^3 - \xi)$ can be used to build the cubic field extension $\mathbb{F}_{p^3}$. In that field, an element α is represented as, $\alpha_2 u^2 + \alpha_1 u + \alpha_0 u$, $\alpha_0, \alpha_1, \alpha_2 \in \mathbb{F}_p$. The above construction leads to the following arithmetic costs over $\mathbb{F}_{p^3}$, $M_{p^3} = 8M_p + 2Mc_p$, $S_{p^3} = 7S_p + 2Mc_p$, $I_{p^3} = I_p + 12M_p + 4Mc_p$.

Since the cubic non-residue ξ over $\mathbb{F}_p$ was also selected to be a quadratic non-residue over $\mathbb{F}_p$, then the quadratic extension $\mathbb{F}_{p^3} \subset \mathbb{F}_{p^6}$ can be constructed as, $F_{p^6} \cong \mathbb{F}_{p^3}[v]/(v^2 - u)$, since the element u is a quadratic non-residue in $\mathbb{F}_{p^3}$.

For further comparisons when $p \equiv 1 \pmod 4$, we also consider the twelfth field extension $\mathbb{F}_{p^{12}}$. Notice that the element v is also a quadratic non-residue in $\mathbb{F}_{p^6}$. Hence, $\mathbb{F}_{p^{12}}$ can be seen as the quadratic extension $\mathbb{F}_q[v]/\left(v^2 - u\right)$.

Tables I-IV present our experimental results in terms of number of general field multiplications, multiplications by a constant and inversions in $\mathbb{F}_p$, for different choices of odd primes p.[5] For the case $p \equiv 3 \bmod 4$, it can be seen from Tables I and III that the complex method is the most efficient procedure followed by Algorithm 9. In the case when $p \equiv 3 \bmod 4$, it can be seen from Tables II and IV that the complex method and Algorithm 10 are the two most efficient solutions. All these three algorithms are considerable more efficient than the classical Tonelli-Shanks and Müller's procedures. In a scenario where the multiplication by constants is negligible (for example when the irreducible binomials that were used to build the extension field has a constant of value ± 1 and/or a small power of two), then Algorithm 10 outperforms the complex method in most scenarios. This is the case for the three primes considered in Table-IV for square root computation over the field extension $\mathbb{F}_{p^{12}}$.

Table III

NUMBER OF OPERATIONS IN $\mathbb{F}_p$ FOR SQUARE ROOTS IN $\mathbb{F}_{q^2}$, $q = p^3$, $p \equiv 3 \pmod 4$

Parameter		$u = -(2^{62} + 2^{55} + 1)$	$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	$u = 2^{63} + 2^9 + 2^8 + 2^6 + 2^4 + 2^3 + 1$
Bit length of p		254	256	258
Algo 8	M_p	6405	9025	7235
	Mc_p	3693	4921	4091
	I_p	0	0	0
Complex Alg.	M_p	3603	2657	3951
	Mc_p	1235	1590	1351
	I_p	1	1	1
Tonelli-Shanks	M_p	26670	52286	26972
	Mc_p	14474	29644	14649
	I_p	0	0	0
Müller's Algo	M_p	38681	39357	40538
	Mc_p	19837	20057	20162
	I_p	1	1	1

[5]The corresponding Maple and magma scripts can be downloaded at: http://delta.cs.cinvestav.mx/~francisco/codigo.html.

Table IV

NUMBER OF OPERATIONS IN $\mathbb{F}_p$ FOR SQUARE ROOTS IN $\mathbb{F}_{q^2}$, $q = p^6$, $p \equiv 1 \pmod 4$

Parameter		$p = 2^{224} - 2^{96} + 1$	$u = 2^{62} - 2^{54} + 2^{44}$	$u = 2^{63} - 2^{49}$
Bit length of p		224	254	256
Algo 9	M_p	24293	19937	21044
	Mc_p	10630	10105	10605
	I_p	1	0	0
Complex Alg.	M_p	31337	19932	22984
	Mc_p	10971	7603	8673
	I_p	7	3	3
Tonelli-Shanks	M_p	223455	174555	191310
	Mc_p	116154	90018	98844
	I_p	0	0	0
Müller's Algo	M_p	202534	218683	232429
	Mc_p	102354	115591	117467
	I_p	1	1	1

VII. CONCLUSION

In this paper the computation of square roots over extension fields of the form $\mathbb{F}_{q^2}$, with $q = p^n$, p an odd prime and $n \geq 1$, was studied, including two novel proposals for the cases $q \equiv 1 \bmod 4$ (Algorithm 9) and $q \equiv 3 \bmod 4$ (Algorithm 10). From the complexity analysis of these algorithms and corresponding experimental results, we conclude that the complex method of [27] is the most efficient option in the case when $q \equiv 3 \bmod 4$. For the case when $q \equiv 1 \bmod 4$, in many cases, Algorithm 10 is the most efficient approach closely followed by the complex method.

VIII. ACKNOWLEDGMENTS

We wish to thank Pierrick Gaudry, Andrew Sutherland and Paul Zimmermann for providing insightful comments on earlier versions of this draft.

REFERENCES

[1] IEEE P1363-2000 draft standard for traditional public-key cryptography, may 2006. available at: http://grouper.ieee.org/groups/1363/tradPK/index.html.

[2] A. Atkin. Probabilistic primality testing, summary by F. Morain. *Research Report 1779, INRIA*, pages 159–163, 1992.

[3] E. Bach and K. Huber. Note on taking square-roots modulo N. *IEEE Transactions on Information Theory*, 45(2):807–809, 1999.

[4] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2002.

[5] P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In S. Cimato, C. Galdi, and G. Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2003.

[6] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. E. Tavares, editors, *Selected Areas in Cryptography SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2005.

[7] P. S. L. M. Barreto and J. F. Voloch. Efficient computation of roots in finite fields. *Des. Codes Cryptography*, 39(2):275–280, 2006.

[8] N. Benger and M. Scott. Constructing tower extensions of finite fields for implementation of pairing-based cryptography. In M. A. Hasan and T. Helleseth, editors, *Arithmetic of Finite Fields, Third International Workshop, WAIFI 2010*, volume 6087 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 2010.

[9] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves. In M. Joye, A. Miyaji, and A. Otsuka, editors, *Pairing-Based Cryptography - Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2010.

[10] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, number 2248 in Lecture Notes in Computer Science, pages 514–532. Springer, 2001.

[11] S. Chatterjee, D. Hankerson, E. Knapp, and A. Menezes. Comparing two pairing-based aggregate signature schemes. *Des. Codes Cryptography*, 55(2):141–167, 2010.

[12] M. Cipolla. Un metodo per la risoluzione della congruenza di secondo grado. *Rend. Accad. Sci. Fis. Mat. Napoli*, vol. 9:154–163, 1903.

[13] M. S. D. Henkerson, A. Menezes. *Software Implementation of pairings*. IOS Press, 2009.

[14] J. Doliskani and É. Schost. Taking roots over high extensions of finite fields. *CoRR*, abs/1110.4350, 2011.

[15] P. Friedland. Algorithm 312: Absolute value and square root of a complex number. *Commun. ACM*, 10(10):665–, Oct. 1967.

[16] C. C. F. P. Geovandro, M. A. S. Jr., M. Naehrig, and P. S. L. M. Barreto. A family of implementation-friendly bn elliptic curves. *Journal of Systems and Software*, 84(8):1319–1326, 2011.

[17] D.-H. Han, D. Choi, and H. Kim. Improved computation of square roots in specific finite fields. *IEEE Transaction on Computers*, vol. 58, No. 2:188–196, 2009.

[18] E. J. Kachisa, E. F. Schaefer, and M. Scott. Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. In S. D. Galbraith and K. G. Paterson, editors, *Pairing-Based Cryptography - Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 126–135. Springer, 2008.

[19] H. Kato, Y. Nogami, and Y. Morikawa. A high-speed square root algorithm for extension fields. *Memoirs of the Faculty of Engineering, Okayama University*, vol. 43:99–107, 2009.

[20] H. Katou, F. Wang, Y. Nogami, and Y. Morikawa. A high-speed square root algorithm in extension fields. In M. S.

Rhee and B. Lee, editors, *Information Security and Cryptology - ICISC 2006*, volume 4296 of *Lecture Notes in Computer Science*, pages 94–106. Springer, 2006.

[21] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In N. P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer, 2005.

[22] F. Kong, Z. Cai, J. Yu, and D. Li. Improved generalized Atkin algorithm for computing square roots in finite fields. *Information Processing Letters*, vol. 98, no. 1:1–5, 2006.

[23] S. Lindhurst. An analysis of shanks's algorithm for computing square roots in finite fields. *CRM Proc. and Lecture Notes*, Vol. 19:231–242, 1999.

[24] V. S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology - CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

[25] S. Müller. On the computation of square roots in finite fields. *J. Design, Codes and Cryptography*, vol. 31:301–312, 2004.

[26] R. Schoof. Elliptic curves over finite fields and the computation of square roots $\mod p$. *Mathematics of Computation*, Vol. 44:483–494, April 1985.

[27] M. Scott. Implementing cryptographic pairings over Barreto-Naehrig curves. In T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto, editors, *Pairing-Based Cryptography - Pairing 2007, First International Conference*, volume 4575 of *Lecture Notes in Computer Science*, pages 177–196. Springer, 2007.

[28] D. Shanks. Five number-theoretic algorithms. *Proceedings of the second Manitoba conference on numerical mathematics*, pages 51–70, 1972.

[29] O. T. T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and Computation*, Vol. 78:171–177, 1988.

[30] A. Tonelli. Bemerkung uber die auflosung quadratischer congruenzen. *Götinger Nachrichten*, pages 344–346, 1891.

[31] G. Tornarìa. Square roots modulo p. *In LATIN 2002: Theoretical Informatics, Lecture Notes in Comput. Sci, Springer*, vol. 2286:430–434, 2002.

[32] F. Wang, Y. Nogami, and Y. Morikawa. An efficient square root computation in finite fields GF(p^{2^d}). *IEICE Transactions*, 88-A(10):2792–2799, 2005.

Algorithm 11 Computing the sequence $C_{k,c}(a) = a^{1+s+s^2+\cdots+s^{k-1}}$, with $s = p^c$, $1 \leq c < m$

Require: $a \in \mathbb{F}_q$, $q = p^m$, $s = p^c$, $1 \leq c < m$, k.
Ensure: $C_{k,s}(a) = a^{1+s+s^2+\cdots+s^k}$
1: **if** $k = 0$ **then**
2: **return** a.
3: **end if**
4: **if** $k \equiv 1 \bmod 2$ **then**
5: $n \leftarrow (k-1)/2$.
6: $C \leftarrow C_{n,c}(a)$
7: $C \leftarrow C \cdot C^{s^{n+1}}$
8: **else**
9: $n \leftarrow k/2$.
10: $C \leftarrow C_{n-1,c}(a)$
11: $C \leftarrow \left(C \cdot C^{s^n}\right)^s \cdot a$
12: **end if**
13: **return** C.

APPENDIX A: EVALUATION OF $C_q(k, c)$

Let $a \in \mathbb{F}_q$ and define $C_{k,c}(a) = a^{1+s+s^2+\cdots+s^{k-1}}$ with $s = p^c$. Then, we have [7]:

$$C_{k,c}(a) = \begin{cases} C_{n,c}(a)(C_{n,c}(a))^{s^n} & \text{if } k = 2n, \\ \left(C_{n,c}(a)\left(C_{n,c}(a)\right)^{s^n}\right)^s a & \text{if } k = 2n+1. \end{cases} \tag{14}$$

Algorithm 11 computes $C_{k,c}(a)$ by applying Eq. (14) recursively. It can be seen that the computational cost of Algorithm 11 is either one multiplication and one Frobenius operator over $\mathbb{F}_q$, if k is even; or two multiplications and two Frobenius operators over $\mathbb{F}_q$, if k is odd. Furthermore, notice that Algorithm 11 invokes itself exactly $\lfloor \log_2 k \rfloor$ times. Assuming that half of these invocations correspond to n even and the other half to n odd, the overall average complexity of Algorithm 11 for computing $C_{k,c}(a)$ can be estimated as,

$$\frac{3}{2}\left[\lfloor \log_2 k \rfloor + 1\right](M_q + F_q).$$

APPENDIX B: COMPLEXITY OF SQUARE ROOT ALGORITMS FOR ODD EXTENSIONS FIELDS

Algorithm 12 Computing the exponentiation $a^{\frac{q-3}{4}} \in \mathbb{F}_q^*$, with $q = p^m$

Require: $a \in \mathbb{F}_q^*$, p.
Ensure: $a^{\frac{q-3}{4}}$.
1: $\alpha \leftarrow \frac{p-3}{4}$
2: $y_1 \leftarrow a^{\alpha}$.
3: $y \leftarrow y_1 a$. $\{a^{\alpha+1}\}$
4: $y \leftarrow y^2$. $\{a^{2\alpha+2}\}$
5: $y \leftarrow y y_1$. $\{a^{3\alpha+2}\}$
6: $y_2 \leftarrow y_1^p$. $\{a^{p\alpha}\}$
7: $y \leftarrow y_2 y$. $\{a^{p\alpha+(3\alpha+2)}\}$
8: $y \leftarrow y^p$. $\{a^{p(p\alpha+(3\alpha+2))}\}$
9: $y \leftarrow C_{\frac{m-1}{2},2}(y)$. {via Algorithm 11}
10: $y \leftarrow y_1 y$. $\{a^{\alpha+p(p\alpha+(3\alpha+2))\sum_{i=0}^{(m-3)/2} p^{2i}}\}$
11: **return** y.

SHANKS' ALGORITHM

The computational cost of Algorithm 7 is one exponentiation, two multiplications and one squaring. Han *et al.* [17] show how to rewrite the exponent $(q-3)/4$ in terms of p as,

$$\frac{q-3}{4} = \alpha + p\left[p\alpha + (3\alpha+2)\right] \sum_{i=0}^{(m-3)/2} p^{2i},$$

where $\alpha = \frac{p-3}{4}$.

Algorithm 12 shows that the exponentiation $a^{\frac{q-3}{4}}$ can be computed by performing an exponentiation $a^{\frac{p-3}{4}}$, plus 4 multiplications, one squaring and two Frobenius over $\mathbb{F}_q$ plus one evaluation of the sequence $C_q(\frac{m-1}{2}, 2)$ that can be recursively computed using Algorithm 11. The

average cost of computing $a^{\frac{p-3}{4}}$ and $C_q(\frac{m-1}{2}, 2)$ is $(\frac{1}{2}\lfloor\log_2(p)\rfloor - \frac{3}{2})M_q + (\lfloor\log_2(p)\rfloor - 2)S_q$, and $\frac{3}{2}\lfloor\log_2 m\rfloor(M_q + F_q)$, respectively. Therefore, the overall average computational cost associated to the Shanks Algorithm 2 when a is a square is given as,

$$\text{Shanks Alg. cost} = \left[\frac{1}{2}\lfloor\log_2(p)\rfloor + \frac{3}{2}\lfloor\log_2(m)\rfloor + \frac{5}{2}\right] M_q + \left[\lfloor\log_2(p)\rfloor - 1\right] S_q + \left[\frac{3}{2}\lfloor\log_2(m)\rfloor + 2\right] F_q.$$

Algorithm 13 Computing the exponentiation $a^{\frac{q-5}{8}} \in \mathbb{F}_q^*$, with $q = p^m$

Require: $a \in \mathbb{F}_q^*$, p.
Ensure: $a^{\frac{q-5}{8}}$.
1: $\alpha \leftarrow \frac{p-5}{8}$
2: $y_1 \leftarrow a^{\alpha}$.
3: $y_2 \leftarrow y_1 a$. $\{a^{\alpha+1}\}$
4: $y \leftarrow y_1 y_2$. $\{a^{2\alpha+1}\}$
5: $y \leftarrow y^2$. $\{a^{4\alpha+2}\}$
6: $y \leftarrow y y_2$. $\{a^{5\alpha+3}\}$
7: $y_2 \leftarrow y_1^p$. $\{a^{p\alpha}\}$
8: $y \leftarrow y_2 y$. $\{a^{p\alpha+(5\alpha+3)}\}$
9: $y \leftarrow y^p$. $\{a^{p(p\alpha+(5\alpha+3))}\}$
10: $y \leftarrow C_{\frac{m-1}{2},2}(y)$. {via Algorithm 11}
11: $y \leftarrow y_1 y$. $\{a^{\alpha+p(p\alpha+(5\alpha+3))\sum_{i=0}^{(m-3)/2} p^{2i}}\}$
12: **return** y.

ATKIN'S ALGORITHM

The computational cost of Algorithm 3 is one exponentiation, four multiplications and two squarings in $\mathbb{F}_q$. Han *et al.* [17] show that the exponent $(q-5)/8$ can be rewritten in base p as,

$$\frac{q-5}{8} = \alpha + p\left[p\alpha + (5\alpha + 3)\right] \sum_{i=0}^{(m-3)/2} p^{2i},$$

where $\alpha = \frac{p-5}{8}$.

Algorithm 13 shows that the exponentiation $a^{\frac{q-5}{8}}$ can be computed by performing an exponentiation $a^{\frac{p-5}{8}}$, plus 5 multiplications, one squaring and two Frobenius over $\mathbb{F}_q$ plus one evaluation of the sequence $C_q(\frac{m-1}{2}, 2)$ that can be recursively computed using Algorithm 11. The average cost of computing $a^{\frac{p-5}{8}}$ and $C_q(\frac{m-1}{2}, 2)$ is $(\frac{1}{2}\lfloor\log_2(p)\rfloor - \frac{3}{2})M_q + (\lfloor\log_2(p)\rfloor - 3)S_q$, and $\frac{3}{2}\lfloor\log_2 m\rfloor(M_q + F_q)$, respectively. Therefore, the average computational cost associated to the Atkin Algorithm 2 when a is a square is given as,

$$\text{Atkin Alg. cost} = \left[\frac{1}{2}\lfloor\log_2(p)\rfloor + \frac{3}{2}\lfloor\log_2(m)\rfloor + 3\right] M_q + \lfloor\log_2(p)\rfloor S_q + \left[\frac{3}{2}\lfloor\log_2(m)\rfloor + 2\right] F_q.$$

Algorithm 14 Computing the exponentiation $a^{\frac{q-9}{16}} \in \mathbb{F}_q^*$, with $q = p^m$

Require: $a \in \mathbb{F}_q^*$, p.
Ensure: $a^{\frac{q-9}{16}}$.
1: $\alpha \leftarrow \frac{p-9}{16}$
2: $y_1 \leftarrow a^{\alpha}$.
3: $y_2 \leftarrow y_1 a$. $\{a^{\alpha+1}\}$
4: $y \leftarrow (y_1 y_2)^4$. $\{a^{8\alpha+4}\}$
5: $y \leftarrow y y_2$. $\{a^{9\alpha+5}\}$
6: $y_2 \leftarrow y_1^p$. $\{a^{p\alpha}\}$
7: $y \leftarrow y_2 y$. $\{a^{p\alpha+(9\alpha+5)}\}$
8: $y \leftarrow y^p$. $\{a^{p(p\alpha+(9\alpha+5))}\}$
9: $y \leftarrow C_{\frac{m-1}{2},2}(y)$. {via Algorithm 11}
10: $y \leftarrow y_1 y$. $\{a^{\alpha+p(p\alpha+(9\alpha+5))\sum_{i=0}^{(m-3)/2} p^{2i}}\}$
11: **return** y.

KONG *et al.* ALGORITHM

The computational cost of Algorithm 4 is one exponentiation, six and a half multiplications and four and a half squarings in $\mathbb{F}_q$. For this case, the exponent $(q-9)/16$ can be rewritten in base p as,

$$\frac{q-9}{16} = \alpha + p\left[p\alpha + (9\alpha+5)\right] \sum_{i=0}^{(m-3)/2} p^{2i},$$

where $\alpha = \frac{p-9}{16}$.

Algorithm 14 shows that the exponentiation $a^{\frac{q-9}{16}}$ can be computed by performing an exponentiation $a^{\frac{p-9}{16}}$, plus 5 multiplications, two squarings and two Frobenius over $\mathbb{F}_q$ plus one evaluation of the sequence $C_q(\frac{m-1}{2}, 2)$ that can be recursively computed using Algorithm 11. The average cost of computing $a^{\frac{p-9}{16}}$ and $C_q(\frac{m-1}{2}, 2)$ is $(\frac{1}{2}\lfloor \log_2(p) \rfloor - \frac{3}{2})M_q + (\lfloor \log_2(p) \rfloor - 4)S_q$, and $\frac{3}{2}(\lfloor \log_2 m \rfloor)(M_q + F_q)$, respectively. Therefore, the overall average computational cost associated to the Kong *et al.* Algorithm 4 is given as,

$$\text{Kong } et\ al.\ \text{Alg. cost} = \left[\frac{1}{2}\lfloor \log_2(p) \rfloor + \frac{3}{2}\lfloor \log_2(m) \rfloor + 10\right] M_q + \left[\lfloor \log_2(p) \rfloor + \frac{5}{2}\right] S_q + \left[\frac{3}{2}\lfloor \log_2(m) \rfloor + 2\right] F_q.$$

Algorithm 15 Computing the exponentiation $a^{\frac{t-1}{2}} \in \mathbb{F}_q^*$, with $q - 1 = p^m - 1 = 2^s * t$

Require: $a \in \mathbb{F}_q^*$, p.
Ensure: $a^{\frac{t-1}{2}}$.
1: $\alpha \leftarrow \frac{x-1}{2}$
2: $y_1 \leftarrow a^{\alpha}$.
3: $y \leftarrow y_1^2$. $\{a^{x-1}\}$
4: $y \leftarrow ya$. $\{a^{x}\}$
5: $y \leftarrow y^{2^{s-1}}$. $\{a^{x2^{s-1}}\}$
6: $y \leftarrow y_1 y$. $\{a^{\alpha+x2^{s-1}}\}$
7: $y_2 \leftarrow y_1^p$. $\{a^{p\alpha}\}$
8: $y \leftarrow y_2 y$. $\{a^{p\alpha+\alpha+x2^{s-1}}\}$
9: $y \leftarrow y^p$. $\{a^{p(p\alpha+\alpha+x2^{s-1})}\}$
10: $y \leftarrow C_{\frac{m-1}{2},2}(y)$. {via Algorithm 11}
11: $y \leftarrow y_1 y$. $\{a^{\alpha+p(p\alpha+\alpha+x2^{s-1})\sum_{i=0}^{(m-3)/2} p^{2i}}\}$
12: **return** y.

TONELLI-SHANKS ALGORITHM

The computational cost of Algorithm 5 varies depending if the input is or not a quadratic residue in $\mathbb{F}_q$. By taking into account the average contribution of QR and QNR inputs, and using the complexity analysis given in [23] for the classical Tonelli-Shanks algorithm it can be found that its average cost is

$$\frac{1}{2}\Big[\lfloor \log_2(q) \rfloor + 4\Big] M_q + \Big[\lfloor \log_2(q) \rfloor + \frac{1}{8}\big(s^2 + 3s - 16\big) + \frac{1}{2^s}\Big] S_q. \tag{15}$$

However, rewriting once again the exponent $q - 9/16$ in base p as,

$$\frac{t-1}{2} = \alpha + p\left[p\alpha + \alpha + 2^{s-1}x\right] \sum_{i=0}^{(m-3)/2} p^{2i},$$

where $q - 1 = 2^s t$, $p - 1 = 2^s x$, with t and x odd integers, and $\alpha = \frac{x-1}{2}$.

Algorithm 15 shows that the exponentiation $a^{\frac{t-1}{2}}$ can be computed by performing an exponentiation $a^{\frac{x-1}{2}}$, plus 4 multiplications, s squarings and two Frobenius over $\mathbb{F}_q$ plus one evaluation of the sequence $C_q(\frac{m-1}{2}, 2)$ that can be recursively computed using Algorithm 11. The average cost of computing $a^{\frac{x-1}{2}}$ and $C_q(\frac{m-1}{2}, 2)$ is $(\frac{1}{2}\lfloor \log_2(p) \rfloor - \frac{3}{2})M_q + (\lfloor \log_2(p) \rfloor - s - 1)S_q$, and $\frac{3}{2}\lfloor \log_2 m \rfloor (M_q + F_q)$, respectively. Therefore, the overall average computational cost associated to The Tonelli-Shanks Algorithm 5 is given as,

$$\begin{aligned} \text{Tonelli-Shanks Alg. cost} \;=\; & \left[\frac{1}{2}\lfloor \log_2(p) \rfloor + \frac{3}{2}\lfloor \log_2(m) \rfloor + \frac{s}{2} + 5\right] M_q \\ & + \left[\lfloor \log_2(p) \rfloor + \frac{1}{8}(s^2 + 11s - 16) + \frac{1}{2^s}\right] S_q + \left[\frac{3}{2}\lfloor \log_2(m) \rfloor + 2\right] F_q. \end{aligned}$$

MÜLLER'S ALGORITHM

Algorithm 7 requires the computation of the exponentiation $(at^2 - 4)^{\frac{q-1}{2}}$ in step 5, which we can be done as shown in the preliminary part by the computation of $C_q(m, 2)$ and an exponentiation with exponent $\frac{p-1}{2}$ in $\mathbb{F}_p$ of costs $\frac{3}{2}(\lfloor \log_2 m \rfloor + 1)(M_q + F_q)$, and $(\frac{1}{2}\lfloor \log_2(p) \rfloor - \frac{3}{2})M_p + (\lfloor \log_2(p) \rfloor - 1)S_p$, respectively. Moreover, the $\frac{q-1}{4} - th$ element of a Lucas Sequence can be found using Alg. 6 at a cost of $(\lfloor \log_2(q) \rfloor - \frac{5}{2})M_q + (\lfloor \log_2(q) \rfloor - \frac{3}{2})S_q$. Using the Itoh-Tsujii method, an inversion a^{-1} in $\mathbb{F}_q$ can be performed by doing the following:

$$a^{-1} = \left(a\left(a^{1+p+p^2+\cdots+p^{m-2}}\right)^p\right)^{-1}\left(a^{1+p+p^2+\cdots+p^{m-2}}\right)^p,$$

where the inversion $\left(a\left(a^{1+p+p^2+\cdots+p^{m-2}}\right)^p\right)^{-1}$ is computed in the base-field $\mathbb{F}_p$. Hence the cost of an inversion is $1C_q(m-1,1)+2M_q+1F_q+1I_p$. Therefore, the overall average computational cost associated to the Müller's Algorithm 7 is given as,

$$\text{Müller Alg. cost} = \begin{cases} \left[\lfloor\log_2(q)\rfloor + \frac{3}{2}\lfloor\log_2(m)\rfloor - 1\right] M_q + \left[\lfloor\log_2(q)\rfloor - \frac{3}{2}\right] S_q \\ + \left[\frac{3}{2}\lfloor\log_2(m)\rfloor + \frac{3}{2}\right] F_q + \left[\frac{1}{2}\lfloor\log_2(p)\rfloor - \frac{3}{2}\right] M_p + \left[\lfloor\log_2(p)\rfloor - 1\right] S_p & \text{if } (a-4)^{\frac{q-1}{2}} = -1, \\ \left[\lfloor\log_2(q)\rfloor + 6\lfloor\log_2(m)\rfloor + \frac{15}{2}\right] M_q + \left[\lfloor\log_2(q)\rfloor + \frac{1}{2}\right] S_q \\ + \left[6\lfloor\log_2(m)\rfloor + 7\right] F_q + \left[\frac{3}{2}\lfloor\log_2(p)\rfloor - \frac{9}{2}\right] M_p + \left[3\lfloor\log_2(p)\rfloor - 3\right] S_p + 1I_p & \text{if } (a-4)^{\frac{q-1}{2}} = 1. \end{cases}$$

Which gives in average,

$$\begin{aligned} \text{Müller Alg. cost} \;=\; & \left[\lfloor\log_2(q)\rfloor + \frac{15}{4}\lfloor\log_2(m)\rfloor + \frac{13}{4}\right] M_q + \left[\lfloor\log_2(q)\rfloor - \frac{1}{2}\right] S_q \\ & + \left[\frac{15}{4}\lfloor\log_2(m)\rfloor + \frac{17}{4}\right] F_q + \left[\lfloor\log_2(p)\rfloor - 3\right] M_p + \left[2\lfloor\log_2(p)\rfloor - 2\right] S_p + \frac{1}{2} I_p. \end{aligned}$$