Low Complexity Bit-Parallel Square Root Computation over $GF(2^m)$ for all Trinomials *

Francisco Rodríguez-Henríquez, Guillermo Morales-Luna
Computer Science Section
CINVESTAV-IPN
Av. IPN 2508, Mexico City, Mexico
{francisco, gmorales}@cs.cinvestav.mx

Julio López-Hernández
Institute of Computing
State University of Campinas, Brazil
jlopez@ic.unicamp.br

Abstract

In this contribution we introduce a low-complexity bit-parallel algorithm for computing square roots over binary extension fields. Our proposed method can be applied for any type of irreducible polynomials. We derive explicit formulae for the space and time complexities associated to the square root operator when working with binary extension fields generated using irreducible trinomials. We show that for those finite fields, it is possible to compute the square root of an arbitrary field element with equal or better hardware efficiency than the one associated to the field squaring operation. Furthermore, a practical application of the square root operator in the domain of field exponentiation computation is presented. It is shown that by using as building blocks squarers, multipliers and square root blocks, a parallel version of the classical square-and-multiply exponentiation algorithm can be obtained. A hardware implementation of that parallel version may provide a speedup of up to 50% percent when compared with the traditional version.

*Work partially supported by Mexican CONACYT under grant 45306

keyword: Finite field arithmetic; binary extension fields; cryptography

1 Introduction

Arithmetic over binary extension fields *GF*(2^m) has many important applications, particularly in the theory of error control coding, symmetric block ciphers and elliptic curve cryptosystems [1,2,3,4,5]. Those applications typically require high performance implementation of most if not all of the basic finite field operations such as field addition, subtraction, multiplication, division, exponentiation and square root [6].

In particular, field square root computation has become an important building block in the design of some elliptic curve primitives such as point compression and point halving [1, 3, 4]. Moreover, recently, a novel parallel formulation of the standard Itoh-Tsujii multiplicative inverse algorithm for multiplicative inverse computation over *GF*(2^m) using as main building blocks field multiplication, field squaring and field square root operators was proposed in [7]. A speedup of nearly 30% with respect to the original Itoh-Tsuii was reported.

For most applications, the efficiency of finite field arithmetic implemented in hardware is typically measured in terms of associated design space and time complexities. The space complexity is defined as the total amount of hardware resources needed to implement the circuit, i.e. the total number of logic gates required by the design. Time complexity, on the other hand, is simply defined as the total gate delay or critical path of the circuit, frequently formulated using gate delay T_x units.

Let P(x) be an irreducible polynomial over *GF*(2). Then, the binary extension field of degree $m \in \mathbb{N}^+$ *GF*(2^m) can be defined as,

$$GF(2^m) \cong GF(2)[x]/\left(P(x)\right) = \left\{a_0 + a_1 x + \cdots + a_{m-1}x^{m-1} \bmod P(x) | a_i \in GF(2)\right\}$$

Field square root operation is defined as follows. Let A be an arbitrary element in the field *GF*(2^m) as described above. The square root of A, denoted as $\sqrt{A}$ or $A^{\frac{1}{2}}$, is the element $D \in GF(2^m)$ such that $D^2 = A \bmod P(x)$.

A straightforward but rather expensive approach for computing $\sqrt{A}$ is based on Fermat's Little Theorem which establishes that for any nonzero element $A \in GF(2^m)$, the identity $A^{2^m} = A$ holds. Therefore, $\sqrt{A}$ may be computed as $D = A^{2^{m-1}}$ with a computational cost of $m-1$ field squarings [1].

A more efficient algorithm based on a refining of the above Fermat's Little Theorem method was proposed by Fong et al. in [8, 9], and it is based on the observation that $\sqrt{A}$ can be efficiently implemented by extracting the two half-length vectors $A_{even} = (a_{m-1}, a_{m-3}, \cdots, a_2, a_0)$ and $A_{odd} = (a_{m-2}, a_{m-4}, \cdots, a_3, a_1)$ and by performing a field multiplication of length $\lfloor \frac{m}{2} \rfloor$ of A_{odd} with the pre-computed value of the element $x^{\frac{1}{2}}$ followed by an addition with A_{even}. The cost of the algorithm presented in [8] consists of one field multiplication of length $m/2$ bits by a pre-computed constant, which is still an expensive operation.

However, if the irreducible polynomial $P(x)$ is a trinomial, $P(x) = x^m + x^n + 1$ with m an odd prime number. The authors in [8] observed that the square root of any arbitrary element $A \in GF(2^m)$ can be computed using relatively few additions and shift operations. In particular, in [8] is reported that the computational cost of a square root in the field $GF(2^{233})$, using $P(x) = x^{233} + x^{74} + 1$, requires roughly $1/8$ the time of a field multiplication when both operations are implemented in a software platform.

In this contribution, an alternative method for computing square roots over binary finite field is proposed. The most important findings presented in this paper are threefold. Firstly, after a careful analysis of the method proposed in [8], we derive a closed expression for the square root operator when using irreducible trinomials of the type $P(x) = x^m + x^n + 1$, with m odd, n even, and $\lceil \frac{m-1}{4} \rceil \leq n < \lfloor \frac{m-1}{3} \rfloor$. Secondly, we describe an alternative method for computing $\sqrt{A}$ which is based on the linear property exhibited by the field squaring operation in binary extension fields. Our proposed method can be applied for any type of irreducible polynomials. In particular, it is shown that for the important practical case of finite fields generated using irreducible trinomials, the square root operation can be performed with no more computational cost than the one associated to the field squaring operation. Moreover, we derive explicit formulae for both, field squaring and square root operations, detailing their corresponding area and time complexities when implemented in hardware platforms. Thirdly, we present a practical application of the square root operator in the domain of field exponentiation computation.

The rest of this paper is organized as follows. Section 2 provides an analysis of the method proposed in [8]. Furthermore, a closed expression for the square root operator when using irreducible trinomials of the type $P(x) = x^m + x^n + 1$, with m odd, n even, and $\lceil \frac{m-1}{4} \rceil \leq n < \lfloor \frac{m-1}{3} \rfloor$ is derived. In Section 3, the squaring and square root operations over binary finite fields generated by irreducible trinomials are analyzed from a linear algebra perspective. Then, in Section 4, the

proposed bit-parallel algorithm for square root computation is explained in detail. Moreover, we give explicit formulae for the efficient computation of squaring and square roots over binary extension fields generated by irreducible trinomials. Section 5 includes several illustrative examples of field square root computation. Section 6 describes how the square root operator can be applied for speeding up the exponentiation operation in binary extension fields. Finally, in Section 7 some conclusion remarks are drawn.

2 The Fong et al. Method for Computing Square Roots

As it was mentioned in the previous Section, the Fong et al. method for computing square roots over binary extension fields is based on the observation that the element $\sqrt{A}$ can be expressed in terms of the square root of the monomial x as,

$$A^{\frac{1}{2}} = \sum_{i=0}^{\frac{m-1}{2}} a_{2i} x^i + x^{\frac{1}{2}} \sum_{i=0}^{\frac{m-3}{2}} a_{2i+1} x^i \bmod P(x) \tag{1}$$

Eq. (1) can be efficiently implemented by extracting the two half-length vectors $A_{even} = (a_{m-1}, a_{m-3}, \cdots, a_2, a_0)$ and $A_{odd} = (a_{m-2}, a_{m-4}, \cdots, a_3, a_1)$ and by performing a field multiplication of length $\lfloor \frac{m}{2} \rfloor$ of A_{odd} with the precomputed value $x^{\frac{1}{2}}$ followed by an addition with A_{even}.

In the event that the irreducible polynomial $P(x)$ is a trinomial, $P(x) = x^m + x^n + 1$ with m an odd prime number, authors in [8] found the following handy equations for $\sqrt{A}$,

$$A^{\frac{1}{2}} = \begin{cases} A_{even} + (x^{\frac{m+1}{2}} + x^{\frac{n+1}{2}}) \cdot A_{odd} & n \text{ odd}, \\ A_{even} + \left[x^{-\frac{m-1}{2}} (x^{\frac{n}{2}} + 1) \cdot A_{odd} \right] \bmod P(x) & n \text{ even}. \end{cases} \tag{2}$$

According to Eq. (2), when the middle coefficient n of the irreducible trinomial $P(x)$ is odd, the square root of any arbitrary element $A \in GF(2^m)$ can be found using few additions and shift operations [1]. However, in the case that the middle coefficient n is even, the square root computation becomes a bit more computational intensive. In the following we analyze the latter case in more detail.

[1] We stress that the polynomial $(x^{\frac{m+1}{2}} + x^{\frac{n+1}{2}}) \cdot A_{odd}$ has degree $m-1$. Thus, in this case we do not need to perform a polynomial modular reduction.

Notice that in the quotient field $GF(2)[x]/\left(P(x)\right)$, isomorphic to $GF(2^m)$, the element $x^{-\frac{m-1}{2}}$ can be computed by using the reduced identity $x^{-s} = x^{n-s}+x^{m-s}$, for $1 \leq s \leq n$. If we impose now the restriction $\lceil \frac{m-1}{4} \rceil \leq n \leq \frac{m-1}{2}$, it follows that $\frac{m-1}{2} < 2n$. Then, in the field $GF(2)[x]/\left(P(x)\right)$ we can write, $x^{-n} = 1+x^{m-n}$ and letting $t = \frac{m-1}{2} - n$ we have the following identities modulo $P(x)$:

$$x^{-t} = x^{n-t} + x^{m-t} = x^{2n-\frac{m-1}{2}} + x^{n+\frac{m+1}{2}}$$

Then, the element $x^{-\frac{m-1}{2}} = x^{-t} \cdot x^{-n}$ can be written as,

$$\begin{aligned} x^{-\frac{m-1}{2}} &= x^{-t} \cdot x^{-n} \\ &= (x^{2n-\frac{m-1}{2}} + x^{n+\frac{m+1}{2}}) \cdot (1 + x^{m-n}) \\ &= x^{2n-\frac{m-1}{2}} + x^{n+\frac{m+1}{2}} + x^{n+\frac{m+1}{2}} + x^{m+\frac{m+1}{2}} \end{aligned}$$

Therefore,

$$x^{-\frac{m-1}{2}}(x^{\frac{n}{2}} + 1) = x^{2n-\frac{m-1}{2}} + x^{\frac{m+1}{2}} + x^{\frac{2n+m+1}{2}} + x^{\frac{5n-(m-1)}{2}} + x^{\frac{m+n+1}{2}} + x^{\frac{3n+m+1}{2}}$$

After examining above equation, it results convenient to impose an additional restriction to the middle coefficient n, namely, $n < \lfloor \frac{m-1}{3} \rfloor$. In this way, we assure that the exponent $\frac{3n+m+1}{2}$ will not exceed the value m.

Above result and Eq. (2) indicate us how to calculate the square root of an arbitrary nonzero field element $A \in GF(2^m)$, when the field has been generated using the irreducible trinomial $P(x) = x^m + x^n + 1$, with m an odd number and n an even number such that $\lceil \frac{m-1}{4} \rceil \leq n < \lfloor \frac{m-1}{3} \rfloor$. The explicit formula is given as,

$$\begin{aligned} A^{\frac{1}{2}} &= A_{even} + \\ &\quad A_{odd} \Big(x^{2n-\frac{m-1}{2}} + x^{\frac{m+1}{2}} + x^{\frac{2n+m+1}{2}} + \\ &\qquad x^{\frac{5n-(m-1)}{2}} + x^{\frac{m+n+1}{2}} + x^{\frac{3n+m+1}{2}} \Big). \end{aligned} \tag{3}$$

In the following Sections we will discuss alternative methods for deriving and developing even further Eqs. (2) and (3).

3 A Linear Algebra Analysis of the Squaring and Square Root Field Operations

Let $P(x) = x^m + x^n + 1$ be an irreducible trinomial with $m \geq 2$ and $1 \leq n \leq m-1$. As it was done in last Section, we will consider that the Galois

field $GF(2^m)$ is equivalent to the quotient field $GF(2)[x]/\left(P(x)\right)$. Thus in what follows all polynomial identities shall be understood modulo $P(x)$. Then, in the field $GF(2^m)$ the following *Reduction Rule* holds:

$$k \geq m \implies x^k = x^{k-m}x^m = x^{k-m}(1+x^n) = x^{k-m} + x^{k-m+n}, \tag{4}$$

in other words, the power x^k can be expressed as the addition of two lower powers whose exponents differ in n.

Let A be an arbitrary element of the field $GF(2^m)$, represented in the canonical basis as an $m-1$ degree polynomial, namely,

$$A(x) = \sum_{i=0}^{m-1} a_i x^i \quad , \quad a_i \in 0, 1,\ 0 \leq i \leq m-1.$$

Then the square $C = A^2 \bmod P(x)$ in $GF(2^m)$ may be obtained by computing first the polynomial product of A by itself, followed by a reduction step modulo $P(x)$. In fact, since the characteristic of the field is 2, the square map is linear, thus the polynomial square of A is,

$$A^2(x) = \left(\sum_{i=0}^{m-1} a_i x^i\right) \cdot \left(\sum_{i=0}^{m-1} a_i x^i\right) = \sum_{i=0}^{m-1} a_i x^{2i} \tag{5}$$

Let $X_1 = \left[1\ x\ \cdots\ x^{m-1}\right]^T$ be the m-dimensional column vector whose entries are the consecutive powers of x. Since squaring is linear and its kernel contains only the element $0 \in GF(2^m)$, the set of squares $\{x^{2i}\}_{i=0}^{m-1}$ is linearly independent, thus it forms a basis of $GF(2^m)$. Let $X_2 = \left[1\ x^2\ \cdots\ x^{2(m-1)}\right]^T$ be the column vector whose entries are the elements of that basis. Then, there is a non-singular $(m \times m)$-matrix N we have $X_2 = NX_1$. The matrix N can be partitioned as

$$N = \left[\begin{array}{c} L_0 \\ K_0 \end{array}\right] \tag{6}$$

where L_0 is a matrix of order $m_1 \times m$, K_0 is a matrix of order $m_2 \times m$, $m_1 = \lceil \frac{m}{2} \rceil$, $m_2 = m - m_1$, and the rows of L_0 are the canonical vectors with even indexes:

$$L_0 = [\ell_{ij}]_{0 \leq i \leq m_1 - 1}^{0 \leq j \leq m-1} \text{ and } \left(\ell_{ij} = 1 \iff j = 2i\right). \tag{7}$$

Consequently, the odd-indexed columns of L_0 are zero.

Furthermore, for $2i \geq m$, let us write $r = 2i - m$. Then, from the reduction rule (4), we have $x^{2i} = x^r + x^{n+r}$. Notice that whenever the second exponent of last expression is not lower than m, the reduction rule may be reapplied.

Thus, the first rows, namely $\lfloor m/2 \rfloor - \lfloor n/2 \rfloor$, of K_0 have just two values 1 and they are at entries whose separation is n. Eq. (5) asserts that, with respect to the polynomial basis, the column vector consisting of the coordinates, with respect to the polynomial basis, of A^2 is given as,

$$c = N^T a \tag{8}$$

where a is the coefficient vector of A.

Naturally, the inverse of N^T will represent the square root linear map, with respect to the polynomial basis. Let us calculate the inverse matrix N^{-1} of N. As we did in Eq. (6), let us introduce the following partition

$$N^{-1} = [\ L_1 \ \ K_1\ ] \tag{9}$$

where L_1 has order $m \times m_1$, K_1 has order $m \times m_2$, $m_1 = \lceil \frac{m}{2} \rceil$ and $m_2 = m - m_1$. From (6) and (9) we get,

$$\mathbf{1}_m = N\ N^{-1} = \begin{bmatrix} L_0 \\ K_0 \end{bmatrix} [\ L_1 \ \ K_1\ ] = \begin{bmatrix} L_0 L_1 & L_0 K_1 \\ K_0 L_1 & K_0 K_1 \end{bmatrix}$$

where $\mathbf{1}_m$ is the $(m \times m)$-identity matrix. From above equations, it follows that,

$$\mathbf{1}_{m_1} = L_0 L_1 \quad , \quad \mathbf{0}_{m_2 \times m_1} = K_0 L_1 \tag{10}$$

$$\mathbf{0}_{m_1 \times m_2} = L_0 K_1 \quad , \quad \mathbf{1}_{m_2} = K_0 K_1. \tag{11}$$

Eqs. (10) assert that the columns of L_1 should, on the one hand, form a biorthonormal system with the rows of L_0, and, on the other, be orthogonal to the rows of K_0.

We observe that the rows of L_0 form an orthonormal basis of a space S_0 of dimension m_1 over $GF(2^m)$ considered as the m-dimensional vector space over the prime field $GF(2)$. Thus the orthogonal complement of S_0 has dimension m_2. In other words, there are exactly 2^{m_2} vectors which are orthogonal to all rows in L_0. Hence, there exist $2^{m_1 m_2}$ matrices L of order $m \times m_1$ such that $L_0 L = \mathbf{0}_{m_1}$.

Due to the special form of L_0, if the even-indexed rows of a matrix L are selected, a zero matrix is gotten. Let $\mathbf{L}_0^{\perp}$ denote the collection of such matrices.

Also we have $L_0 L_0^T = \mathbf{1}_{m_1}$. It follows that the general solution of the first equation in (10) is given as,

$$L_1 = L_0^T + L \qquad \text{with } L \in \mathbf{L}_0^{\perp}. \tag{12}$$

In order to satisfy the second equation in (10) we must have

$$K_0 L_0^T = K_0 L. \tag{13}$$

Let us write $K_0 = (k_{ij})_{0 \le i \le m_2 - 1}^{0 \le j \le m-1}$. From relation (7), it follows that the general entry of $K_0 L_0^T$ is $\sum_{\mu=0}^{m-1} k_{i\mu} \ell_{j\mu} = k_{i,2j}$, for $0 \le i \le m_2 - 1$, $0 \le j \le m_1 - 1$.

Equation (13) will be satisfied if and only if the inner product of the i-th row of K_0 and the j-th column of L gives the value $k_{i,2j}$, $0 \le i \le m_2 - 1$, $0 \le j \le m_1 - 1$. Hence, matrix K_0 should satisfy the following conditions:

$$k_{i,2j} = 0 \implies \begin{pmatrix} \text{there is an even number of 1's in the } i\text{-th row of} \\ K_0 \text{ appearing at odd indexed entries} \end{pmatrix} \tag{14}$$

$$k_{i,2j} = 1 \implies \exists j_i \text{ odd} : k_{i,j_i} = 1 \tag{15}$$

Conditions (14) and (15) along with the fact that the matrix N is non-singular, determine uniquely the matrix $L \in \mathbf{L}_0^{\perp}$. As a consequence, the submatrix L_1 that satisfies eqs. (10) is also uniquely determined. In a similar but more complicated way, it can be verified that the submatrix K_1 satisfying eqs. (11) is uniquely determined.

Based on the conclusions obtained from the linear algebra approach just outlined, we derive in the next Section explicit formulae for the field square root operation.

4 Explicit Formulae for the Squaring and Square Root Field Operations

Once again, let us consider binary extension fields constructed using irreducible trinomials of the form $P(x) = x^m + x^n + 1$, with $m \ge 2$. It is convenient to consider, without loss of generality, the additional restriction $1 \le n \le \lfloor \frac{m}{2} \rfloor$ [2].

[2] It is known that if $P(x) = x^m + x^n + 1$ is irreducible over $GF(2)$, so is $P(x) = x^m + x^{m-n} + 1$ [10]. Hence, provided that at least one irreducible trinomial of degree m exists, it is always possible to find another irreducible trinomial such that its middle coefficient n satisfies the restriction $1 \le n \le \lfloor \frac{m}{2} \rfloor$.

The rest of this Section is organized as follows. First, in Subsection 4.1, we give the corresponding formulae needed for computing the field squaring operation when considering arbitrary irreducible trinomials. Those equations are then used in Subsection 4.2 to find the corresponding ones for the field square root operator.

4.1 Field Squaring Computation

Let $A = \sum_{i=0}^{m-1} a_i x^i$ be an arbitrary element of $GF(2^m)$. Then, according to Eq. (5) its square, A^2, can be represented by the $2m$-coefficient vector,

$$\begin{aligned} A^2(x) &= \left[0 \quad a_{m-1} \quad 0 \quad a_{m-2} \quad \ldots \quad 0 \quad a_1 \quad 0 \quad a_0\right] \\ &= \left[a'_{2m-1} \quad a'_{m-2} \quad \ldots \quad a'_{m-1} \quad a'_m \;;\; a'_{m-1} \quad a'_2 \quad \ldots \quad a'_1 \quad a'_0\right] \end{aligned} \tag{16}$$

where $a'_i = 0$ for i odd. Let us reduce this vector using rule (4). Hence, the upper half of A^2 (i.e., the m most significant bits) in Eq. (16) is mapped into the first m coordinates by performing addition and shift operations only. Let us write

$$\begin{aligned} C &= A^2 \bmod P(x) \\ &= A'_{[0,m-1]} + A'_{[m,2m-1]} + A'_{[m,2m-1-n]}x^n \\ &\quad + \left(A'_{[2m-n,2m-1]} + A'_{[2m-n,2m-1]}x^n\right) \end{aligned} \tag{17}$$

Thus, the reduction step is computed by the addition of four terms,

$$\begin{aligned} W &= A'_{[0,m-1]} \\ X &= A'_{[m,2m-1]} \\ Y &= A'_{[m,2m-1-n]}x^n \\ Z &= A'_{[2m-n,2m-1]} + A'_{[2m-n,2m-1]}x^n \end{aligned}$$

This procedure is shown schematically in Fig. 1. Notice that for those designs implemented in hardware platforms, the field squaring computation procedure just outlined can be instrumented by using XOR logic gates only. Nevertheless, the exact computational complexity of this arithmetic operation depends on the explicit form of m and the middle coefficient n in the trinomial $P(x)$.

In order to investigate the exact cost of the field squaring operation, in this contribution we categorize all the irreducible trinomials over $GF(2)$ into four different types. For all four types considered and by means of Eqs. (16) and (17), the following explicit formulae for the field squaring operation, which happened to be consistent with Eq. (8), were found.

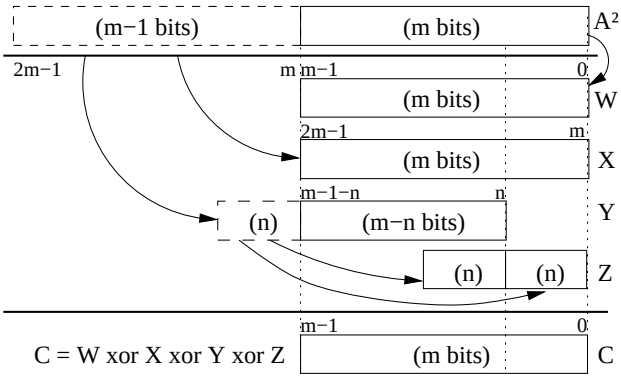

Figure 1: Reduction scheme.

Type I: Computing $C = A^2 \bmod P(x)$, with $P(x) = x^m + x^n + 1$, m even, n odd and $n < \frac{m}{2}$,

$$c_i = \begin{cases} a_{\frac{i}{2}} + a_{\frac{m+i}{2}} & i \text{ even}, i < n \text{ or } i \geq 2n, \\ a_{\frac{i}{2}} + a_{\frac{m+i}{2}} + a_{m-n+\frac{i}{2}} & i \text{ even}, n < i < 2n, \\ a_{m+1-\frac{n+i}{2}} & i \text{ odd}, i < n, \\ a_{\frac{m-n+i}{2}} & i \text{ odd}, i \geq n, \end{cases} \tag{18}$$

for $i = 0, 1, \cdots, m-1$. It can be verified that Eq. (18) has an associated cost of $\frac{m+n-1}{2}$ XOR gates and $2T_x$ delays.

Type II: Computing $C = A^2 \bmod P(x)$, with $P(x) = x^m + x^n + 1$, m even, n odd and $n = \frac{m}{2}$,

$$c_i = \begin{cases} a_{\frac{i}{2}} + a_{\frac{m+i}{2}} & i \text{ even}, i < n, \\ a_{\frac{i}{2}} & i \text{ even}, i > n, \\ a_{m+1-\frac{n+i}{2}} & i \text{ odd}, i < n, \\ a_{\frac{n+i}{2}} & i \text{ odd}, i \geq n, \end{cases} \tag{19}$$

for $i = 0, 1, \cdots, m-1$. It can be verified that Eq. (19) has an associated cost of $\frac{m+2}{4}$ XOR gates and one T_x delay.

Type III: Computing $C = A^2 \bmod P(x)$, with $P(x) = x^m + x^n + 1$, m, n odd

numbers and $n < \frac{m+1}{2}$,

$$c_i = \begin{cases} a_{\frac{i}{2}} & i \text{ even}, i < n, \\ a_{\frac{i}{2}} + a_{\frac{i}{2}+\frac{m-n}{2}} + a_{\frac{i}{2}+(m-n)} & i \text{ even}, n < i < 2n, \\ a_{\frac{i}{2}} + a_{\frac{i}{2}+\frac{m-n}{2}} & i \text{ even}, i \geq 2n, \\ a_{\frac{m+i}{2}} + a_{\frac{m+i}{2}+\frac{m-n}{2}} & i \text{ odd}, i < n, \\ a_{\frac{m+i}{2}} & i \text{ odd}, i \geq n, \end{cases} \tag{20}$$

for $i = 0, 1, \cdots, m-1$. It can be verified that Eq. (20) has an associated cost of $\frac{m-1}{2}$ XOR gates and $2T_x$ delays.

Type IV: Computing $C = A^2 \bmod P(x)$, with $P(x) = x^m + x^n + 1$, m odd, n even and $n < \frac{m+1}{2}$,

$$c_i = \begin{cases} a_{\frac{i}{2}} + a_{\frac{i}{2}+m-\frac{n}{2}} & i \text{ even}, i < n, \\ a_{\frac{i}{2}} + a_{\frac{i}{2}+m-n} & i \text{ even}, n \leq i < 2n, \\ a_{\frac{i}{2}} & i \text{ even}, i \geq 2n, \\ a_{\frac{m+i}{2}} & i \text{ odd}, i < n, \\ a_{\frac{m+i}{2}} + a_{\frac{m+i}{2}-\frac{n}{2}} & i \text{ odd}, i > n, \end{cases} \tag{21}$$

for $i = 0, 1, \cdots, m-1$. It can be verified that Eq. (21) has an associated cost of $\frac{m+n-1}{2}$ XOR gates and one T_x delay.

The complexity costs found on Equations (18) through (21) are in consonance with the ones analytically derived in [11, 12].

4.2 Field Square Root Computation

In the following, we keep the assumption that the middle coefficient n of the generating trinomial $P(x) = x^m + x^n + 1$ satisfies the restriction $1 \leq n \leq \frac{m}{2}$.

Clearly, Eqs. (18)-(21) are a consequence of the fact that in binary extension fields, squaring is a linear operation. The linear nature of binary extension field squaring, allow us to describe this operator in terms of an $(m \times m)$-matrix as,

$$C = A^2 = MA \tag{22}$$

where $M = N^T$, and N is as described in Eq. (6).

Furthermore, based on Eq. (22), it follows that computing the square root of an arbitrary field element A means finding a field element $D = \sqrt{A}$ such that $D^2 = MD = A$. Hence,

$$D = M^{-1}A \tag{23}$$

Eq. (23) is especially attractive for fields *GF*(2^m) with order sufficiently large, i.e., $m >> 2$, where the matrixes M corresponding to Eqs. (18)-(21) are all highly spare (each row has at most three nonzero values).

Hence, for the trinomial types I, II, III and IV as described above, the element $D = \sqrt{A}$ given by Eq. (23) can be found by using the matrix form of Eqs. (18)-(21), respectively, followed by the computation of the inverse of the corresponding matrix M.

Using Eqs. (10) and (11), together with the fact that $M^{-1} = (N^{-1})^T$ allow us to determine the m coordinates of the field element $\sqrt{a} = d = M^{-1}a$ as described bellow.

Type I: Computing D such that $D^2 = A \bmod P(x)$, with $P(x) = x^m + x^n + 1$, m even, n odd, and $n < \frac{m}{2}$:

$$d_i = \begin{cases} a_{2i} + a_{2i+n} & i \leq \lfloor \frac{n}{2} \rfloor, \\ a_{2i} + a_{(2i+n) \bmod m} + a_{2i-n} & \lfloor \frac{n}{2} \rfloor < i < n, \\ a_{2i} + a_{(2i+n) \bmod m} & n \leq i < \frac{m}{2}, \\ a_{(2i+n) \bmod m} & \frac{m}{2} \leq i < m \end{cases} \tag{24}$$

for $i = 0, 1, \cdots, m-1$. It can be verified that Eq. (24) has an associated cost of $\frac{m+n-1}{2}$ XOR gates and $2T_x$ delays.

Type II: Computing D such that $D^2 = A \bmod P(x)$, with $P(x) = x^m + x^n + 1$, m even, n odd and $n = \frac{m}{2}$:

$$d_i = \begin{cases} a_{2i} + a_{2i+\frac{m}{2}} & i < \frac{m+2}{4}, \\ a_{2i} & \frac{m+2}{4} \leq i < \frac{m}{2} \\ a_{(2i+\frac{m}{2}) \bmod m} & \frac{m}{2} \leq i < m \end{cases} \tag{25}$$

for $i = 0, 1, \cdots, m-1$. It can be verified that Eq. (25) has an associated cost of $\frac{m+2}{4}$ XOR gates and one T_x delay.

Type III: Computing D such that $D^2 = A \bmod P(x)$, with $P(x) = x^m + x^n + 1$, m, n odd numbers and $n < \frac{m+1}{2}$,

$$d_i = \begin{cases} a_{2i} & i < \frac{n+1}{2}, \\ a_{2i} + a_{2i-n} & \frac{n+1}{2} \leq i < \frac{m+1}{2}, \\ a_{2i-n} + a_{2i-m} & \frac{m+1}{2} \leq i < \frac{m+n}{2}, \\ a_{2i-m} & \frac{m+n}{2} \leq i < m \end{cases} \tag{26}$$

for $i = 0, 1, \cdots, m-1$. It can be verified that Eq. (26) has an associated cost of $\frac{m-1}{2}$ XOR gates and one T_x

delay.

Type IV: Computing D such that $D^2 = A \bmod P(x)$, with $P(x) = x^m + x^n + 1$, m, odd, n even and $\lceil \frac{m-1}{4} \rceil \leq n < \lfloor \frac{m-1}{3} \rfloor$.

$$d_i = \begin{cases} a_{2i} + a_{2i+(m-n)} + a_{2i+(m-2n)} + a_{2i+(m-3n)} & i < \frac{4n-(m-1)}{2}, \\ a_{2i} + a_{2i+(m-n)} + a_{2i+(m-2n)} + a_{2i+(m-3n)} & \\ +a_{2i+(m-4n)} & \frac{4n-(m-1)}{2} \leq i < \frac{n}{2}, \\ a_{2i} + a_{2i+(m-2n)} + a_{2i+(m-3n)} + a_{2i+(m-4n)} & \frac{n}{2} \leq i < \frac{5n-(m-1)}{2}, \\ a_{2i} + a_{2i+(m-2n)} + a_{2i+(m-3n)} + a_{2i+(m-4n)} & \\ +a_{2i+(m-5n)} & \frac{5n-(m-1)}{2} \leq i < n, \\ a_{2i} & n \leq i \leq \frac{m-1}{2}, \\ a_{2i-m} & \frac{m+1}{2} \leq i < \frac{n+m+1}{2}, \\ a_{2i-m} + a_{2i-(m+n)} & \frac{n+m+1}{2} \leq i < \frac{2n+m+1}{2} \\ a_{2i-m} + a_{2i-(m+n)} + a_{2i-(m+2n)} & \frac{2n+m+1}{2} \leq i < \frac{3n+m+1}{2} \\ a_{2i-m} + a_{2i-(m+n)} + a_{2i-(m+2n)} + a_{2i-(m+3n)} & \frac{3n+m+1}{2} \leq i < m \end{cases} \tag{27}$$

for $i = 0, 1, \cdots, m-1$. At first glance, Eq. (27) can be implemented with an XOR gate cost of,

$$3 \cdot \frac{4n-(m-1)}{2} + 4 \cdot \frac{m-3n-1}{2} + 3 \cdot \frac{4n-(m-1)}{2} + 4 \cdot \frac{m-3n-1}{2} + \frac{n}{2} + 2 \cdot \frac{n}{2} + 3 \cdot \frac{m-3n-1}{2} = 5 \cdot \frac{m-n-1}{2} - \frac{n}{2}.$$

However, taking advantage of the high redundancy of the terms involved in Eq. (27), it can be shown (after a tedious long derivation) that actually $\frac{m+n-1}{2}$ XOR gates are sufficient to implement it with a $2T_x$ gate delays.

Table 1: Summary of complexity results

Type	Trinomial $P(x) = x^m + x^n + 1$	Operation	XOR gates	Time delay
I	m even, n odd	Squaring	$(m+n-1)/2$	$2T_x$
II	m even, $n = m/2$	Squaring	$(m+2)/4$	T_x
III	m odd, n odd	Squaring	$(m-1)/2$	$2T_x$
IV	m odd, n even	Squaring	$(m+n-1)/2$	T_x
I	m even, n odd	Square root	$(m+n-1)/2$	$2T_x$
II	m even, $n = m/2$	Square root	$(m+2)/4$	T_x
III	m odd, n odd	Square root	$(m-1)/2$	T_x
IV	m odd, n even	Square root	$(m+n-1)/2$	$2T_x$

Table 2: Irreducible trinomials $P(x) = x^m + x^n + 1$ of degree $m \in [160, 571]$ encoded as $m(n)$, with m a prime number

$m(n)$	Type	$m(n)$	Type	$m(n)$	Type
167(35)	III	281(93)	III	439(49)	III
191(9)	III	313(79)	III	449(167)	III
193(15)	III	337(55)	III	457(61)	III
199(67)	III	353(69)	III	463(93)	III
223(33)	III	359(117)	III	479(105)	III
233(74)	IV	367(21)	III	487(127)	III
239(81)	III	383(135)	III	503(3)	III
241(70)	IV	401(152)	IV	521(158)	IV
257(41)	III	409(87)	III	569(77)	III
263(93)	III	431(120)	IV		
271(70)	IV	433(33)	III		

Table 1 summarizes the area and time complexities just derived for the cases considered. Furthermore, in Table 2 we list all preferred irreducible trinomials $P(x) = x^m + x^n + 1$ of degree $m \in [160, 571]$ with m a prime number. In all the instances considered the computational complexity of computing the square root operator is comparable or better than that of the field squaring.

5 Illustrative Examples

In order to illustrate the approach just outlined, we include in this Section several examples using first the artificially small finite field $GF(2^{15})$ and then more realistic fields, in terms of practical cryptographic applications.

Table 3: Squaring matrix M of Eq. (22)

$$M = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}$$

Example 5.1. *Field Square Root Computation over* $GF(2^{15})$

Let us consider $GF(2^{15})$ generated with the irreducible Type III trinomial $P(x) = x^{15} + x^7 + 1$. As it was discussed before, one can find the square root of any arbitrary field element $A \in GF(2^{15})$ by applying Eq. (23). In order to follow this approach, based on Eq. (20), we first determine the matrix M of Eq. (22) as

Table 4: Square root matrix M^{-1} of Eq. (23)

$$M^{-1} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}$$

Table 5: Square and square root coefficient vectors.

$$C = \begin{bmatrix}
a_0 \\
a_8 + a_{12} \\
a_1 \\
a_9 + a_{13} \\
a_2 \\
a_{10} + a_{14} \\
a_3 \\
a_{11} \\
a_4 + a_8 + a_{12} \\
a_{12} \\
a_5 + a_9 + a_{13} \\
a_{13} \\
a_6 + a_{10} + a_{14} \\
a_{14} \\
a_7 + a_{11}
\end{bmatrix}, \quad D = \begin{bmatrix}
a_0 \\
a_2 \\
a_4 \\
a_6 \\
a_1 + a_8 \\
a_3 + a_{10} \\
a_5 + a_{12} \\
a_7 + a_{14} \\
a_1 + a_9 \\
a_3 + a_{11} \\
a_5 + a_{13} \\
a_7 \\
a_9 \\
a_{11} \\
a_{13}
\end{bmatrix}$$

shown in Table 3. Then, the inverse matrix of M modulus two, M^{-1}, is obtained as shown in Table 4. Afterwards, the polynomial coefficients, in terms of the coefficients of A, corresponding to the field square $C = A^2$ and the field square root $D = \sqrt{A}$ elements can be found from Eqs. (22) and (23) as shown in Table 5.

As predicted by Eq. (20), field squaring can be computed at a cost of $(m-1)/2 = (15-1)/2 = 7$ XOR gates and one T_x delay. In the same way, the square root operation can be computed at a cost of $\frac{(m-1)}{2} = \frac{(15-1)}{2} = 7$ XOR gates with an incurred delay time of one T_x, which matches Eq. (26) prediction. It is noticed that in this binary extension field, computing a field square root requires the same computational effort than the one associated to field squaring.

Example 5.2. *Field Square Root Computation over GF*(2^{162})

Let us consider *GF*(2^{162}) generated using the irreducible Type II trinomial, $P(x) = x^{162} + x^{81} + 1$. Using the same approach as for the precedent example, we can obtain the square root polynomial coefficients of an arbitrary element A from the field *GF*(2^{162}) as,

$$d_i = \begin{cases} a_{2i} + a_{2i+81} & i < 41, \\ a_{2i} & 41 \leq i < 81 \\ a_{(2i+81) \bmod 162} & 81 \leq i \end{cases} \tag{28}$$

for $i = 0, 1, \cdots, 161$. As predicted by Eq. (25) the associated cost of the field square root computation for this field is given as, $\frac{(m+2)}{4} = \frac{(162+2)}{4} = 41$ XOR gates with an incurred delay time of one T_x.

Example 5.3. *Field Square Root Computation over GF*(2^{409})

Let *GF*(2^{409}) be a field generated with the Type III irreducible trinomial, $P(x) = x^{409} + x^{87} + 1$ [3]. The square root of any arbitrary field element A is given as,

$$d_i = \begin{cases} a_{2i} & i < 44, \\ a_{2i} + a_{2i-87} & 44 \leq i < 205, \\ a_{2i-87} + a_{2i-409} & 205 \leq i < 248, \\ a_{2i-409} & 248 \leq i \end{cases} \tag{29}$$

[3] This is a NIST recommended finite field for elliptic curve applications [13]. It was used as an illustrative example in [8, 9]

for $i = 0, 1, \cdots, 408$. Eq. (29) can be implemented with an XOR gate cost of $\frac{m-1}{2} = 204$ XOR gates with a $2T_x$ gate delay, which agrees with the value predicted by Eq. (26).

Example 5.4. *Field Square Root Computation over GF*(2^{233})

Let *GF*(2^{233}) be a field generated with the Type III irreducible trinomial, $P(x) = x^{233} + x^{74} + 1$ [4]. The square root of any arbitrary field element A is given as,

$$d_i = \begin{cases} a_{2i} + a_{2i+159} + a_{2i+85} + a_{2i+11} & i < 32, \\ a_{2i} + a_{2i+159} + a_{2i+85} + a_{2i+11} + a_{2i-63} & 32 \leq i < 37, \\ a_{2i} + a_{2i+85} + a_{2i+11} + a_{2i-63} & 37 \leq i < 69, \\ a_{2i} + a_{2i+85} + a_{2i+11} + a_{2i-63} + a_{2i-137} & 69 \leq i < 74, \\ a_{2i} & 74 \leq i \leq 116, \\ a_{2i-233} & 116 \leq i < 154, \\ a_{2i-233} + a_{2i-307} & 154 \leq i < 191 \\ a_{2i-233} + a_{2i-307} + a_{2i-381} & 191 \leq i < 228 \\ a_{2i-233} + a_{2i-307} + a_{2i-381} + a_{2i-455} & 228 \leq i < 233 \end{cases} \quad (30)$$

for $i = 0, 1, \cdots, 232$. Eq. (30) can be implemented with an XOR gate cost of $\frac{m+n-1}{2} = 153$ XOR gates with a $4T_x$ gate delay, which agrees with the value predicted by Eq. (27).

6 Applications

The square root operation has several relevant applications in the domain of elliptic curve cryptography, particularly as an important building block for implementing the so-called *point halving* primitive [14, 15, 3].

In the rest of this Section we describe how the square root operator can be applied for speeding up the computation of the exponentiation in binary extension fields.

[4] This is a NIST recommended finite field for elliptic curve applications [13]. It was used as an illustrative example in [8, 9]

6.1 Exponentiation over binary finite fields

Exponentiation over binary finite fields is used for inverse computation via Fermat Little theorem [7] and key agreement schemes such as the Diffie-Hellman protocol, among other applications.

For binary extension fields $GF(2^m)$, generated using the m-degree irreducible polynomial $P(x)$, irreducible over $GF(2)$. Let e be an arbitrary m-bit positive integer e, with a binary expansion representation given as,

$$e \;=\; (1e_{m-2}\ldots e_1e_0)_2 \;=\; 2^{m-1} + \sum_{i=0}^{m-2} 2^i e_i.$$

Then,

$$\begin{aligned} b = a^e \;&=\; a^{2^{m-1}+\sum_{i=0}^{m-2} 2^i e_i} \\ &=\; a^{2^{m-1}} \cdot a^{2^{m-2}e_{m-2}} \cdot \cdots \cdot a^{2^1 e_1} \cdot a^{2^0 e_0} = a^{2^{m-1}} \cdot \prod_{i=0}^{m-2} a^{2^i e_i} \end{aligned} \tag{31}$$

Algorithm 1 MSB-first binary exponentiation.

Require: The irreducible polynomial $P(x)$, $a \in GF(2^m)$, $e = (e_{m-1}\ldots e_1e_0)_2$
Ensure: $b = a^e \bmod P(x)$

1: $b = a$;
2: **for** $i = m-2$ **downto** 0 **do**
3: $\quad b = b^2$;
4: $\quad$ **if** $e_i == 1$ **then**
5: $\quad\quad b = b \cdot a \bmod P(x)$;
6: **Return** b

Binary strategies evaluate (31) by scanning the bits of the exponent e one by one, either from left to right (MSB-first binary algorithm) or from right to left (LSB-first binary algorithm) applying the so-called Horner's rule. Both strategies require a total of $m-1$ iterations. At each iteration a squaring operation is performed, and if the value of the scanned bit is one, a subsequent field multiplication is performed. Therefore, the binary strategy requires a total of $m-1$ squarings and $H(e)-1$ field multiplications, where $H(e)$ is the Hamming weight of the binary representation of e. The pseudo-code of the MSB-first binary algorithm is shown in Algorithm 1.

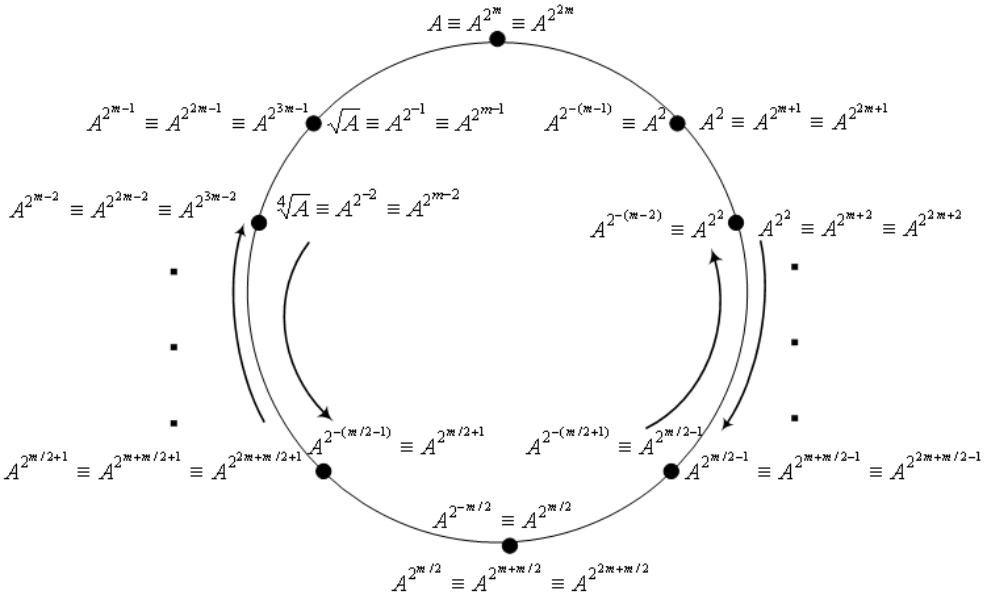

Figure 2: An illustration of the squaring and square root Abelian Groups (with A an arbitrary field element and m an even number)

On the other hand, it is known from Fermat Little Theorem that for any nonzero $a \in GF(2^m)$, we have $a^{2^m-1} = 1$ which implies $a^{2^m} = a$ and by taking square root in both sides of the last relation we get $a^{2^{m-1}} = \sqrt{a} = a^{2^{-1}}$. In general, the i-th square-root of a, with $i \geq 1$ can be written as,

$$a^{2^{m-i}} = a^{2^{-i}}.$$

In other words, the squaring and the square root operators form a multiplicative *Abelian group* of order m as is depicted in Fig. 2. Considering an arbitrary element $A \in GF(2^m)$, with m even, Fig. 2 illustrates, in the clockwise direction, all the m field elements that can be generated by repeatedly computing squarings, i.e., A^{2^i} for $i = 0, 1, \cdots, m-1$. On the other hand, in the counterclockwise direction, Fig. 2 illustrates all the m field elements that can be generated by repeatedly computing the square root operator, i.e., $A^{2^{-i}}$ for $i = 0, 1, \cdots, m-1$.

Hence, Eq. (31) can be reformulated in terms of the square root operator as,

$$\begin{aligned} a^e &= a^{2^{m-1}} \cdot \prod_{i=0}^{m-2} a^{2^i e_i} = a^{2^{m-1}} \cdot a^{2^{m-2} e_{m-2}} \cdot \cdots \cdot a^{2^1 e_1} \cdot a^{2^0 e_0} \\ &= a^{2^{-1}} \cdot a^{2^{-2} e_{m-2}} \cdot \cdots \cdot a^{2^{-(m-1)} e_1} \cdot a^{2^0 e_0} = \sqrt{a} \cdot \prod_{i=2}^{m-1} a^{2^{-i} e_i} \cdot a^{e_0} \end{aligned} \tag{32}$$

Algorithm 2 Square root LSB-first binary exponentiation.

Require: The irreducible polynomial $P(x)$, $a \in GF(2^m)$, $e = (e_{m-1} \ldots e_1 e_0)_2$
Ensure: $b = a^e \bmod P(x)$

1: $b = a$;
2: $e_m = e_0$;
3: **for** $i = 1$ **to** m **do**
4: $\quad b = \sqrt{b}$;
5: $\quad$ **if** $e_i == 1$ **then**
6: $\quad\quad b = b \cdot a \bmod P(x)$;
7: **Return** b

Therefore, the novel square root LSB-first binary strategy requires a total of $m - 1$ square root computations and $H(e) - 1$ field multiplications, where $H(e)$ is the Hamming weight of the binary representation of e. The pseudo-code of the square root LSB-first binary algorithm is shown in Algorithm 2.

Algorithms 1 and 2 suggest a parallel version that can combine both ideas. This parallel version is especially attractive for hardware platforms implementations. Algorithm 3 shows this suggesting algorithm. Notice that both loop computations can be performed in parallel provided that the architecture has two independent field multiplier units. The computational time speedup can be estimated in about 50% when compared with Algorithms 1 and 2.

7 Conclusion

In this work, a low-complexity bit-parallel algorithm for computing square roots over binary extension fields was presented. Although our proposed method can be applied for any type of irreducible polynomials, we were particularly interested in studying the case of irreducible trinomials. Hence, in order to investigate the

Algorithm 3 Squaring and Square root parallel exponentiation.

Require: The irreducible polynomial $P(x)$, $a \in GF(2^m)$, $e = (e_{m-1} \dots e_1 e_0)_2$
Ensure: $b = a^e \bmod P(x)$

1: $b = c = a$;
2: $e_m = 0$;
3: $N = \lfloor \frac{m}{2} \rfloor$;
4: **for** $i = N$ **downto** 0 **do** | **for** $j = N + 1$ **to** m **do**
5: $b = b^2$; | $c = \sqrt{c}$;
6: **if** $e_i == 1$ **then** | **if** $e_j == 1$ **then**
7: $b = b \cdot a$; | $c = c \cdot a$;
8: $b = b \cdot c$;
9: **Return** b

exact cost of the square root operator, we categorized irreducible trinomials over $GF(2)$ into four different types. For all four types considered, explicit area and time complexity formulae were found for both, field squaring and field square root operators. It was shown that for the important practical case of finite fields generated using irreducible trinomials, the square root operation can be performed with no more computational cost than the one associated to the field squaring operation

It results instructive to also study the important practical case of finite fields generated by irreducible pentanomials. Unfortunately, our experiments show that the square root computation becomes much more expensive for this case. As a means of illustrating the pentanomial case, we applied the method described in Section 4.2 to the finite field $GF(2^{163})$ generated with the irreducible pentanomial $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$. The corresponding formulae for the square root computation of an arbitrary field element A are shown in Appendix A. The total gate count is of about 1028 two-input XOR gates with a 6 T_x gate delays.

Field square root operator has several relevant applications in cryptography. In this contribution, we propose a novel application of the square root operator in the domain of exponentiation computation over binary extension fields. It was shown that by using as building blocks squarers, multipliers and square root blocks, a parallel version of the classical square-and-multiply exponentiation algorithm can be obtained. A hardware implementation of that parallel version may provide a speedup of up to 50% percent when compared with the traditional version.

Future work includes exploring other promising applications for the square root operator. In particular, we are interested in studying applications in elliptic

curve cryptography.

ACKNOWLEDGMENTS

The authors are indebted to Darrel Hankerson and Alfred Menezes for suggestions of how to improve this paper.

References

[1] IEEE standards documents, *IEEE P1363: Standard specifications for public key cryptography. Draft Version D18*. IEEE, November 2004, http://grouper.ieee.org/groups/1363/.

[2] J. Daemen and V. Rijmen, *The design of Rijndael, AES-The Advance Encryption Standard*. Springer-Verlag Berlin-Heidelberg, New York, 2002.

[3] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Cryptography*. Springer-Verlag, New York, 2004.

[4] R. Schroeppel, C. Beaver, R. Gonzales, R. Miller, and T. Draelos, "A low-power desing for an elliptic curve digital signature chip," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, B. Kaliski, C. Koc, and C. Paar, Eds., vol. LNCS 2523. Springer-Verlag, August 2002, pp. 366–380.

[5] G. Orlando and C. Paar, "An efficient squaring architecture for GF(2^m) and its applications in cryptographic systems," *IEE Electronic Letters*, vol. 36, no. 13, pp. 1116–1117, June 2000.

[6] D. E. Knuth, *The Art of Computer Programming 3rd. ed.* Reading, Massachusetts: Addison-Wesley, 1997.

[7] F. Rodríguez-Henríquez, G. Morales-Luna, N. Saqib, and N. Cruz-Cortés, "Parallel itoh-tsujii multiplicative inversion algorithm for a special class of trinomials," Cryptology ePrint Archive, Report 2006/035, 2006, http://eprint.iacr.org/.

[8] K. Fong, D. Hankerson, J. López, and A. Menezes, "Field inversion and point halving revisited." *IEEE Trans. Computers*, vol. 53, no. 8, pp. 1047–1059, 2004.

[9] R. Dahab, D. Hankerson, F. Hu, M. Long, J. Lopez, and A. Menezes, "Software multiplication using normal bases," Dept. of Combinatorics and Optimization, Univ. of Waterloo, Canada, Tech. Rep. Technical Report CACR 2004-12, 21 pages, 2004.

[10] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, October 1996.

[11] H. Wu, "Low complexity bit-parallel finite field arithmetic using polynomial basis," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES 99)*, ser. Lecture Notes in Computer Science, C. Koc and C. Paar, Eds., vol. 1717. Springer-Verlag, August 1999, pp. 280–291.

[12] ——, "On complexity of squaring using polynomial basis in GF(2^m)," in *Workshop on Selected Areas in Cryptography (SAC 2000)*, S. T. D. Stinson, Ed., vol. LNCS 2012. Springer-Verlag, September 2000, pp. 118–129.

[13] National Institute of Standards and Technology (NIST), *Recommended Elliptic Curves for Federal Government Use*. NIST Special Publication, July 1999, http://csrc.nist.gov/csrc/fedstandards.html.

[14] R. Schroeppel, "Elliptic curve point ambiguity resolution apparatus and method," International Application Number PCT/US00/31014, filed 9 November 2000.

[15] E. W. Knudsen, "Elliptic scalar multiplication using point halving." in *Advances in Cryptology - ASIACRYPT '99*, ser. Lecture Notes in Computer Science, K.-Y. Lam, E. Okamoto, and C. Xing, Eds., vol. 1716. Springer, 1999, pp. 135–149.

Appendix A: Field Square Root Computation over $GF(2^{163})$

Let $GF(2^{163})$ be a field generated with the irreducible pentanomial, $P(x) = x^{163} + x^7 + x^6 + x^3 + 11$ [5]. Given any arbitrary field element A, the m coefficients of the field element $D = \sqrt{A}$ can be computed as shown in Table 7.

$$
\begin{array}{lcl}
d_0 & = & \sum_{j=0}^{24} a_{159-6j} \oplus a_9 \oplus a_3 \oplus a_0 \\
d_1 & = & \sum_{j=0}^{26} a_{6j+5} \oplus a_2 \\
d_2 & = & \sum_{j=0}^{25} a_{6j+7} \oplus a_4 \\
d_3 & = & a_6 \oplus a_3 \\
d_4 & = & a_8 \oplus a_5 \oplus a_1 \\
d_5 & = & a_{10} \oplus a_7 \oplus a_3 \oplus a_1 \\
d_6 & = & \sum_{j=0}^{24} a_{159-6j} \oplus a_{12} \oplus a_5 \\
d_{80} & = & \sum_{j=0}^{26} a_{6j+5} \oplus a_{160} \sum_{j=0}^{25} a_{6j+3} \\
d_{81} & = & \sum_{j=0}^{26} a_{6j+1} \oplus a_{162} \sum_{j=0}^{25} a_{6j+5} \\
d_{160} & = & \sum_{j=0}^{26} a_{6j+3} \\
d_{161} & = & \sum_{j=0}^{26} a_{6j+5} \\
d_{162} & = & \sum_{j=0}^{26} a_{6j+1} \\
d_{3i+7} & = & \sum_{j=0}^{26} a_{6j+3} \oplus \sum_{j=0}^{24-i} a_{161-6j} \oplus a_{6i+14} \oplus \sum_{j=0}^{i+1} a_{6j+1}, \text{ for } i \in [0, 24] \\
d_{3i+8} & = & \sum_{j=0}^{26} a_{6j+5} \oplus \sum_{j=0}^{23-i} a_{6(i+j)+19} \oplus a_{6i+16} \oplus \sum_{j=0}^{i+1} a_{6j+3}, \text{ for } i \in [0, 23] \\
d_{3i+9} & = & \sum_{j=0}^{26} a_{6j+1} \oplus \sum_{j=0}^{23-i} a_{6(i+j)+21} \oplus a_{6i+18} \oplus \sum_{j=0}^{i+1} a_{6j+5}, \text{for } i \in [0, 23] \\
d_{3i+82} & = & \sum_{j=0}^{26} a_{6j+3} \oplus \sum_{j=0}^{25-i} a_{157-6j}, \text{ for } i \in [0, 25] \\
d_{3i+83} & = & \sum_{j=0}^{26} a_{6j+5} \oplus \sum_{j=0}^{25-i} a_{159-6j}, \text{ for } i \in [0, 25] \\
d_{3i+84} & = & \sum_{j=0}^{26} a_{6j+1} \oplus \sum_{j=0}^{25-i} a_{161-6j}, \text{ for } i \in [0, 25]
\end{array}
$$

Table 6: m-coordinates of the $GF(2^{163})$ field element $D = \sqrt{A}$

We synthesized the equations shown in Table 7 using Xilinx ISE 8.1i design tool, targeting the FPGA virtex2v3000 device. The obtained area and Timing performance figures are shown below.

- The incurred area cost on xor's gates is:

```
Total number of XOR gates          : 524
 1-bit xor7                        : 1
```

[5] This is a NIST recommended finite field for elliptic curve applications [13].

```
1-bit xor9                              : 1
1-bit xor12                             : 2
1-bit xor4                              : 57
1-bit xor5                              : 15

1-bit xor6                              : 67
1-bit xor3                              : 45
1-bit xor2                              : 336
```

- Maximum combinational path delay: 21.243ns