Computing Square Roots and Solve Equations of ECC over Galois Fields

Mohammed Kadhim Rahma[1], Valeriy Hlukhov[2]

1. Department of Computer Engineering, Lviv Polytechnic National University, UKRAINE, Lviv, S. Bandery street 12, E-mail: muhamed_kadhem@yahoo.com

2. Department of Computer Engineering, Lviv Polytechnic National University, UKRAINE, Lviv, S. Bandery street 12, E-mail: valeriygl@ukr.net

Abstract – Computing square roots in finite fields are important computational problems with significant applications to cryptography. Therefore, in this paper, we introduced some methods for finding square roots. Our proposed method calculates the Square root using multiplier over. The proposed method is competitive compared with other existing methods. It is introduced development to decrease of complexity of arithmetic units.

In addition, we approach a novel technique for Computation square roots presented using Half_even_odd unit mixed with multiplier and adder. One approach of Square Root presented using the same multiplier for arithmetic units of ECC. Thus, we goes to gets on reduced complexity for arithmetic units.

Key words: Galois Fields, Elliptical curve cryptography (ECC), square roots, Computation the trace, Computation square roots.

I. Introduction

Computing square roots over finite fields has found many applications in computer Engineering. Our own interest comes from elliptic curve cryptography; there, square root computations into Computation point equation of ECC. In addition, many applications requires several operations of square root and multipliers. This paper presents for used efficient square root based on the multiplier and the adder. The Multiplier Architecture based on the Interleaved MSB-first multiplication. Thus, we have compared multiplication operation with square root.

Moreover, the organization of this paper is as follows: Previous Work, Methods of computing square roots and quadratic equation over, Implementation square roots in Active-VHDL with the simulation and synthesis results followed by conclusion and references.

II. Previous Work

"Irreducibility and r-th root finding over finite fields" that had discuss a conjecture significantly weaker than the Generalized Riemann hypothesis to get a deterministic poly-time algorithm for *r*-th root finding [1]. "Computing *p*-th roots in extended finite fields of prime characteristic $p \geq 2$ "that is Very efficient, direct *p*-th root computation in extended finite fields of characteristic $p \geq 2$ working even for random irreducible reduction polynomial and for any finite field extension was proposed [2]. "Polynomial Representations for *n*-th Roots in Finite Fields" that generalize the results by considering n-th roots over finite fields for arbitrary $n > 2$ and polynomial representation is one of computational problems for find polynomial functions [3]. "Square root computation over even extension fields" that presents a comprehensive study of the computation of square roots over finite extension fields and propose two novel algorithms for computing square roots over even field extensions [4]. "An Efficient Method for Finding Square Root" that look into some methods for finding square roots that need more than one exponentiation in finite field [5]. In addition to the discussed above, there exist other literature in [6], [7].

Here, we will describe some methods that compute the square root and quadratic equation that enable us on the solution of ECC equation, are as follows:

III. Computation the trace

Let, with $c_i \in \{0,1\}$, represented as the $c = (c_{n-1},...,c_0)$, a primitive method for computing $Tr(c)$ uses the definition of trace, requiring *m-1* field squaring's and *m-1* field additions. A much more efficient method makes use of the property that the trace is linear [8], [9]: $Tr(c) = Tr(\sum_{i=0}^{n-1} c_i z^i) = \sum_{i=0}^{n-1} c_i (Tr(z)^i)$.

Trace operator has the important properties that $Tr(y^2) = Tr(y)$ and $Tr(x+y) = Tr(x) + Tr(y)$ for all $x, y \in GF(2^n)$.

The values $Tr(z)^i$ may be precomputed, allowing the trace of an element to be found efficiently, especially if $Tr(z)^i = 0$ for most *i*. Next are examples of computing traces of elements in $GF(2^{163})$ with reduction polynomial $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$. A routine calculation shows that $Tr(z)^i = 1$ if and only if (iff) $i \in \{0,157\}$. As examples, $Tr(z^{160} + z^{46}) = 0$, $Tr(z^{157} + z^{46}) = 1$ and $Tr(z^{157} + z^{46} + 1) = 0$.

IV. Computation square roots

The basic method over $GF(2^n)$ is based on the little theorem as following [8]:

$c^{2^n} = c$. Then $\sqrt{c} = c^{2^{n-1}}$ over $GF(2^m)$, it can be computed with $m-1$ squarings.

A more efficient method is obtained from the observation that $\sqrt{c}$ can be expressed in terms of the square root of the element *z*.

Let $GF(2^n)$, $c_i \in \{0,1\}$. Since squaring is a linear operation in $GF(2^n)$, the square root of *c* can be written as

$$\sqrt{c} = \left(\sum_{i=0}^{m-1} c_i z^i\right)^{2^{m-1}} = \sum_{i=0}^{m-1} c_i (z^{2^{m-1}})^i .$$

Splitting c into even and odd powers, we have

$$\sqrt{c} = \sum_{i=0}^{\frac{m-1}{2}} c_{2i}(z^{2^{m-1}})^{2i} + \sum_{i=0}^{\frac{m-3}{2}} c_{2i+1}(z^{2^{m-1}})^{2i+1} =$$

$$= \sum_{i=0}^{\frac{m-1}{2}} c_{2i} z^{i} + \sum_{i=0}^{\frac{m-3}{2}} c_{2i+1} z^{2^{m-1}} z^{i} =$$

$$= \sum_{i_{even}} c_i z^{\frac{i}{2}} + \sqrt{z} \sum_{i_{odd}} c_i z^{\frac{i-1}{2}} .$$

This reveals an efficient method for $\sqrt{c}$ computing: extract the two half-length vectors $c_{even} = (c_{n-1},...,c_4,c_2,c_0)$ and $c_{odd} = (c_{n-2},...,c_5,c_3,c_1)$ from c (assuming m is odd), perform a field multiplication of c_{odd} of length $\left\lfloor \frac{m}{2} \right\rfloor$ with the precomputed value $\sqrt{z}$, and finally add these results together (Fig.1). There is example of square root calculation for $GF(2^4)$, z=2, $\sqrt{z} = 5 = 0101$, $f(z) = z + 1$, c=1110, $\sqrt{c} = 1101$.

In the case that the reduction polynomial f is a trinomial, the computation of $\sqrt{c}$ can be further accelerated by the observation that an efficient formula for $\sqrt{z}$ can be derived directly from f. Let $f(z) = z^m + z^k + 1$ be an irreducible trinomial of degree m, where $m>2$ is prime. Consider the case that k is odd. Note that $1 \equiv z^m + z^k (mod\ f(z))$. Then multiplying by z and taking the square root, we get $\sqrt{z} \equiv z^{m+\frac{1}{2}} + z^{k+\frac{1}{2}} (mod\ f(z))$.

Thus, the product $\sqrt{z} \cdot c_{odd}$ requires two shift-left operations and one modular reduction. Now suppose k is even. Observe that $z^m \equiv z^k + 1 (mod\ f(z))$. Then dividing by z^{m-1} and taking the square root, we get

$$\sqrt{z} \equiv z^{\frac{-(m-1)}{2}} (z^{\frac{k}{2}} + 1) (mod\ f(z)).$$

In order to compute z^{-s} modulo, where $s = \frac{m-1}{2}$, one can use the congruence's $z^{-t} \equiv z^{k-t} + z^{m-t} (mod\ f(z))$ for $1 \le t \le k$ for writing z^{-s} as a sum of few positive powers of z. Hence, the product $\sqrt{z} \cdot c_{odd}$ can be performed with few shift-left operations and one modular reduction. For example:

Square roots in $GF(2^{409})$**:** The reduction polynomial for the NIST recommended is the trinomial $f(z) = z^{409} + z^{87} + 1$. Then, the new formula for computing the square root of $c \in GF(2^{409})$ is $\sqrt{c} = (c_{evem} + z^{205} \cdot c_{odd} + z^{44} \cdot c_{odd}) mod\ f(z)$.

Square roots in $GF(2^{233})$**:** The reduction polynomial for the NIST recommended is the trinomial $f(z) = z^{233} + z^{74} + 1$. Since k=74 is even, we have $\sqrt{z} = z^{-116} \cdot (z^{37} + 1) mod\ f(z)$. Note that $z^{-74} \equiv (z^{159} + 1) mod\ f(z)$ and $z^{-42} \equiv (z^{32} + z^{191}) mod\ f(z)$. Then one gets that $z^{-116} \equiv (z^{32} + z^{117} + z^{191}) mod\ f(z)$. Hence, the new method for computing the square root of $c \in GF(2^{233})$ is $\sqrt{c} = (c_{evem} + (z^{32} + z^{117} + z^{191})(z^{37} + 1) \cdot c_{odd}) mod\ f(z)$. In addition to above, the computation of quadratic equation is solve with repeated or without repeated roots, with repeated roots, a quadratic equation of the type $y^2 + c = 0$, where $c \in GF(2^n)$ and $c \in GF(2^n)$, we must extract the square root of c. Since in any field of characteristic two we have the identity $(x+y)^2 = x^2 + y^2$ and similarly, $(x+y)^{1/2} = x^{1/2} + y^{1/2}$ the square root is a linear operation [7].

In terms of a fixed basis of $GF(2^n)$, namely $u_1, u_2, ..., u_n$ we may write $c = \sum_{i=1}^{n} c_i u_i$, where $c \in GF(2)$. Because of the linearity of the square root, we then have $\sqrt{c} = \sum_{i=1}^{n} c_i u_i^{1/2}$. Of course, $u_i^{1/2}$ can also be represented in terms of same basis, as $u_i^{1/2} = \sum_{i=1}^{n} R_{i,j} u_j$, with $R_{i,j} \in GF(2)$. We then have $\sqrt{c} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i R_{i,j} u_j$, $\sqrt{c} = \sum_{j=1}^{n} (\sum_{i=1}^{n} c_i R_{i,j}) u_j$.

V. Computation quadratic equation without repeated roots

In general, we can transformation equation of ECC over binary field to the quadratic equation that be written as $y^2 + by + c = 0$ where $b = x$ and $c = x^3 + x^2 + 1$. We have just seen that if $b = 0$, this equation has a unique solution in $GF(2^n)$, and that this solution may be found by multiplying the vector representing c by the matrix R, which extracts square roots.

If $b \neq 0$ we first transform the equation by introducing the new variable $t = y/b \rightarrow y = tb$. This new variable satisfies the equation $b^2 t^2 + b^2 t + c = 0$, or $t^2 + t = d$ where $t = c/b^2$.

We now notice that if $t_i^2 + t_i = v_i$ and $t_j^2 + t_j = v_j$, then $(t_i + t_j)^2 + (t_i + t_j) = v_i + v_j$. Hence, a solution of the equation $t^2 + t = d = \sum d_i v_i;\ d_i \in GF(2)$, is given by $t = \sum d_i t_i$, where t_i is a solution of the equation $t_i^2 + t_i = v_i$.

This shows that the set of v for which the equation $t^2 + t = v$ has a solution in $GF(2^n)$ forms a linear subspace of the vector space $GF(2^n)$, and since each value of v corresponds to two values of t, the dimensionality of the subspace is evidently n-1.

Consequently, the solutions of the equation $t^2 + t + d = 0$ may be represented in terms of solutions to the equations $t_i^2 + t_i + v_i = 0$, for i = 1, 2, , …, n-1, where the v_i span the space of v's for which $t^2 + t + v = 0$ has solutions in $GF(2^n)$.

If d is not expressible as a sum of such v's, the equation $t^2 + t + d = 0$ has no solutions in $GF(2^n)$. If $d = \sum d_i v_i$, then $t = \sum d_i t_i$ is a solution of $t^2 + t + v = 0$. The other solution is found by adding to the first type solution a solution of $t^2 + t = 0$, namely, $t = 1$.

If $t_i^2 + t_i = v_i$, then we may square both sides to obtain $(t_i^2)^2 + t_i^2 = v_i^2$. By repeatedly squaring, we find that $t_i^{2j+1} + t_i^{2j} = v_i^{2j}$. Summing on j gives

$$\sum_{j=0}^{n-1} (t_i^{2j+1} + t_i^{2j}) = \sum_{j=0}^{n-1} v_i^{2j}.$$

The left-hand side of this equation is equal to $t_i^{2^n} + t_i$, which is 0 for all $t_i \in GF(2^n)$. Therefore, if the quadratic equation $t^2 + t = v$ has solutions in $GF(2^n)$, then $Tr(v) = 0$, where $Tr(v)$ is defined as $\sum_{j=0}^{n-1} v^{2^j}$.

However, all elements in $GF(2^n)$ are roots of the equation $y^{2^n} + y = 0$. From the factorization

$y^{2^n} + y = (y + y^2 + y^{2^2} + ... + y^{2^{n-1}})$,

$(1 + y + y^2 + y^{2^2} + ... + y^{2^{n-1}}) = Tr(y)(Tr(y) + 1)$, we see that exactly half of the elements in $GF(2^n)$ have $Tr(y) = 0$ and exactly half have $Tr(y) = 1$. Since the space of v's for which $y^2 + y = d$ has solutions in $GF(2^n)$ has dimension n-1, we have the following theorem: If $d \in GF(2^n)$ the quadratic equation $y^2 + y = d$ has solutions in $GF(2^n)$ iff $Tr(d) = 0$.

Further uses of the trace operator it have seen that the quadratic equation $y^2 + xy = x^3 + x^2 + 1$ over $GF(2^n)$ has solutions in $GF(2^n)$ iff $Tr(\frac{x^3 + x^2 + 1}{x^2}) = 0$.

On the other hand there is next theorem: The cubic equation $x^3 + x = h$, $h \in GF(2^n)$, $h \neq 0$ has a unique solution $x \in GF(2^n)$ iff $Tr(h^{-1}) \neq Tr(1)$.

VI. Implementation square roots calculation in Active-VHDL

This section introduces engineering aspects of implementing of square roots over $GF(2^n)$ calculation that uses computation equations of elliptic curves and cryptographic solutions efficiently and securely in specific environments.

The proposed method of ECC enhance arithmetic unit with implementation of square root operation into control unit level. It uses first level arithmetic units such as Adder and Multiplier. It is also uses ROM and Half_even_odd unit. Half_even_odd divides Galois field elements into Even and Odd groups. Its VHDL-description is below. The proposed design of square root unit (SRU) and its block diagram are presented in Fig. 1 and Fig. 2. Its work is based on the solution of quadratic equation (with or without repeated roots) over $GF(2^n)$.

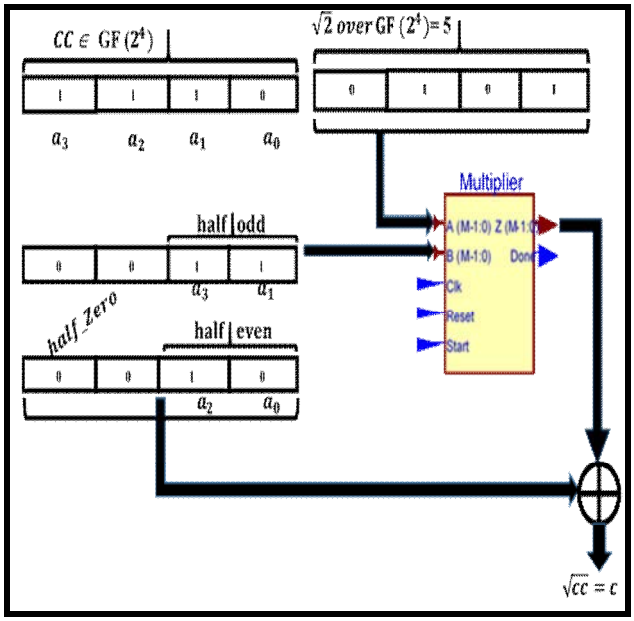

Fig. 1. The proposed design of square root unit with Multiplier.

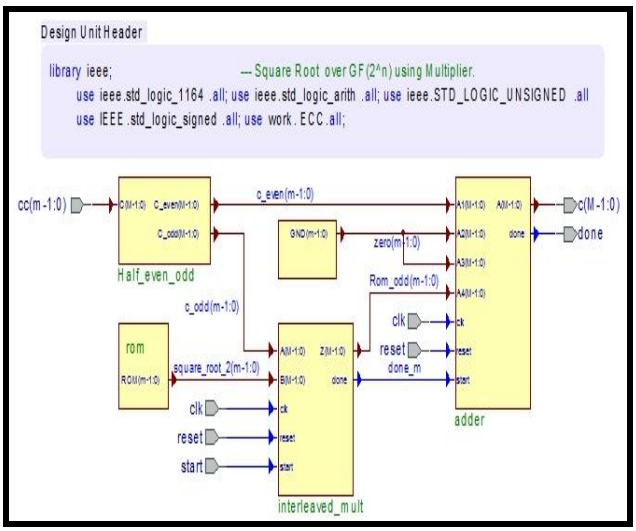

Fig. 2. Block Diagram of Proposed Square root Unit

```
----------------------------------------------------
--- VHDL-description  = Half_even_odd of c(x)
----------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.ECC.all;
ENTITY Half_even_odd IS

PORT(
 C: IN STD_LOGIC_VECTOR(M-1 DOWNTO 0);
 C_even,C_odd,square_root_2: OUT
STD_LOGIC_VECTOR(M-1 DOWNTO 0));
END Half_even_odd;

ARCHITECTURE Half_even_odd OF

Half_even_odd IS

SIGNAL h_even: STD_LOGIC_VECTOR (((M-((m mod
2)*1))/2)-(1-(m mod 2)) DOWNTO 0);
SIGNAL h_odd : STD_LOGIC_VECTOR (((M-((m mod
2)*3))/2)-(1-(m mod 2)) DOWNTO 0);
SIGNAL even_0: STD_LOGIC_VECTOR (((M-((m mod
2)*3))/2)-(1-(m mod 2)) DOWNTO 0):=(others => '0');
SIGNAL odd_0 : STD_LOGIC_VECTOR (((M-((m mod
2)*1))/2)-(1-(m mod 2)) DOWNTO 0):=(others => '0');
signal mult_done: STD_LOGIC;

begin

half_even:FOR i IN 0 TO ((M-((m mod 2)*1))/2)-(1-(m
mod 2))
          GENERATE h_even(i)<=c(2*i);
          end GENERATE;

half_odd : FOR i IN 0 TO ((M-((m mod 2)*3))/2)-(1-(m
mod 2))
          GENERATE h_odd(i)<=c(2*i+1);
          end GENERATE;

          c_odd  <= odd_0 & h_odd;
          c_even <= even_0 & h_even;

END Half_even_odd;
```

Conclusion

In this paper new method of Square Root Calculation in arithmetic units which are used in elliptic curve cryptography (ECC) is presented. The method uses traces of Galois Field elements and also can be used to solve quadratic equations in $GF(2^n)$.

The method is based on usage of existing ECC multipliers. The new approach reduces ECC arithmetic units complexity.

References

[1] Vishwas Bhargava, Gábor Ivanyos, Rajat Mittal, Nitin Saxena, "Irreducibility and r-th root finding over finite fields," Cornell University , USA, 2017.

[2] M. Repka, "Computing pth roots in extended finite fields of prime characteristic p ≥ 2," *The Institution of Engineering and Technology*, vol. 52, no. 9, p. 718 – 719, 2016.

[3] Chang, Seunghwan; Kim, Bihtnara; Lee, Hyang-Sook, "POLYNOMIAL REPRESENTATIONS FOR n-TH ROOTS IN FINITE FIELDS," *Journal of the Korean Mathematical Society,* vol. 52, no. 1, pp. pp.209-224, 2015.

[4] Gora Adj and Francisco Rodrıguez-Henrıquez, "Square root computation over even extension fields," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2829 – 2841, Nov 2014.

[5] S. J. Aboud, "An Efficient Method for Finding Square Root," *International Journal of Statistics, ISSN:2051-8285,* vol. 37, no. 1, pp. 1103-1106, 2013.

[6] PAULO S. L. M. BARRETO, JOSE FELIPE VOLOCH, Efficient Computation of Roots in Finite Fields, Netherlands: Kluwer Academic Publishers, 2004.

[7] E. R. BERLEKAMP,H. RUMSEY, AND G. SOLOMON, "On the Solution of Algebraic Equations over Finite Fields," INFORMATION ANn CONTROL 1O, 553-564, Jet Propulsion Laboralory, Pasadena, California 91103,USA, June 1967.

[8] Darrel Hankerson, Alfred Menezes, Scott Vanstone, Guide to Elliptic Curve Cryptography, United States of America: ISBN 0-387-95273-X, 2004.

[9] K. H and P. Rosen, HANDBOOK OF DISCRETE MATHEMATICS and ITS APPLICATIONS, Boca Raton, FL 33487-2742: Taylor & Francis Group, LLC, 2013.