IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 14, 2019, accepted April 9, 2019, date of publication April 15, 2019, date of current version April 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2911012

Fast Asymptotic Square Root for Two Types of Special Pentanomials

YU ZHANG, YIN LI, AND QING CHEN
Department of Computer Science and Technology, Xinyang Normal University, Xinyang 464000, China

Corresponding author: Yin Li (yunfeiyangli@gmail.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61402393 and Grant 61601396, and in part by the Nanhu Scholars Program for Young Scholars of XYNU.

ABSTRACT Inspired by the Montgomery and generalized polynomial basis (GPB) squaring operation, we introduce and study the notion of *asymptotic square root* over binary extension fields $GF(2^m)$. This new notion is a natural generalization of Montgomery-like square root, a special form of square root that was recently introduced by Li *et al*. Given an arbitrary element $A \in GF(2^m)$ and a fixed element ω, the asymptotic square root is defined as $A^{1/2} \cdot \omega$. We show that, by choosing a proper parameter ω regarding different kind of irreducible polynomials that define $GF(2^m)$, such square root operation can achieve better space and time complexity. Meanwhile, we have proved that the complexity of asymptotic is linear with the generating polynomials. Specifically, for the field $GF(2^m)$ is defined by two type of specific irreducible pentanomials, i.e., Type C.1 and Type C.2 pentanomials, we derived explicit formulae for space and time complexities associated with the asymptotic square root operator. On top of that, a practical application of the asymptotic square root in exponentiation computation is also presented.

INDEX TERMS Finite field, square root, exponentiation, cryptography.

I. INTRODUCTION

Arithmetic operations in finite field $GF(2^m)$ have many important applications such as public key cryptography and coding theory [1], [2]. These applications usually require high efficient implementations of such arithmetic operations, e.g., field addition, multiplication, inversion and squaring. Besides, field square root is also an important building block in the design of some elliptic curve primitives, and thus it has attracted some attentions during recent years [4], [5]. Additionally, because of the similarity and independence of squaring and square root, square root operator has been applied in some parallel exponentiation algorithms [6], [8], where both squaring and square root are utilizing as the main building blocks. The authors have shown that, if squaring and square root computation costs the same circuit delay, the parallel algorithm will achieve twice implementation efficiency compared with classic squaring based exponentiation.

Generally speaking, the efficiency of hardware implementation of field arithmetic is typically evaluated by space and time complexity. The space compleixty is defined as the amount of hardware resources, usually expressed as the number of logic gates (XOR and AND). Accordingly, the time complexity is often expressed as the total logic gates delay of the circuit, denoted by T_A (delay of a XOR gate) and T_X (delay of an AND gate), respectively.

Let $f(x)$ be an irreducible polynomial over $\mathbb{F}_2$. The finite field $GF(2^m)$ generated with $f(x)$ is defined as $GF(2^m) \cong \mathbb{F}_2[x]/(f(x))$. An arbitrary element $A \in GF(2^m)$ here in polynomial basis (PB) can be recognized a polynomial over $\mathbb{F}_2$ of degree less than m. Then, the field square root computation of A, denoted as $\sqrt{A}$ or $A^{1/2}$, is to find $D \in GF(2^m)$ such that $D^2 = A$. Please note that the algorithms [13]–[15] for computing square root in $\mathbb{F}_p$, $(p > 2)$ can not apply here, as $GF(2^m)$ has different algebra structure compared with $\mathbb{F}_p$. Based on Fermat's Little Theorem [3], we know that the identity $A^{2^m} = A$ holds. Obviously, $\sqrt{A} = D = A^{2^{m-1}}$. Under normal basis (NB) representation, such an operation is simply a circular operation and does not cost any logic gates. However, in PB representation, the direct computation of $A^{2^{m-1}}$ requires $m-1$ field squares, which is rather expensive compared with the squaring operation.

A more practical algorithm for field square root computation was proposed by Fong *et al.* [7]. They split A into two parts according to the parity of the exponent of x and utilize the pre-computation value $x^{1/2}$. As a result, the square root of

The associate editor coordinating the review of this manuscript and approving it for publication was Remigiusz Wisniewski.

A can be obtained by $A_{even} + x^{1/2}A_{odd}$, where A_{even} and A_{odd} are polynomials of degree $m/2$ and they consists of the even or odd coefficients of A only. Besides, Rodríguez-Henríquez et al. proposed an alternative algorithm based upon the inversion of the Mastrovito matrix $\mathbf{M}$, which is constructed for the squaring operation. Theoretically, one can obtain the square root formulae for any type of generating polynomials using Rodríguez-Henríquez approach. Nevertheless, classic square root is relatively more complicated than squaring operation, even for certain type of trinomials. For example, given an irreducible trinomial $f(x) = x^m + x^k + 1$ with m odd k even, the corresponding square root operation costs $3T_X$ delay, while the squaring of same case only cost $1T_X$ delay. If $f(x)$ is pentanomial or contains more coefficients, the formulation of the square root will become more complicated.

Recently, Li *et al.* proposed a new type of square root operator for all trinomials [17]. Instead of computing a square root directly, they actually compute a square root multiplying a specific parameter. Following the definition of Montgomery squaring, such a operation is named as as Montgomery-like square root operation. It is demonstrated that, the Montgomery-like square root for trinomial is better than the class one, and cost almost the same space and time complexity as Montgomery squaring.

In this contribution, we extend the work of [17], presenting two types of results. First, we generalize the notion of Montgomery-like square root. Second, we study such square roots for more types of irreducible polynomials. More concretely, we obtain the following main results:

- We presented a general definition of such square root for all the polynomials. To avoid ambiguity with the definition presented in [16], we renamed this operation as *asymptotic square root*. We also prove that the computation complexity for asymptotic square root is linear with the number of terms included in irreducible polynomials.
- We investigate the asymptotic square root for two recently proposed pentanomials, i.e., Type C.1 and Type C.2 pentanomials, which exist for all degrees of practical interest [11]. Explicit formulae for corresponding asymptotic square root are given, and their space and time complexities are investigated. As a result, it is shown these formulae are quite simple and have small space and time complexities in parallel implementation. Coincidentally, this type of square root operator has at least the same circuit delay compared with generalized polynomial basis (GPB) squarings [11] for Type C.1 and C.2 pentanomials.
- Based on previous observation, we describe a new parallel exponentiation algorithm that uses the GPB squaring and asymptotic square root operation as the main building blocks.

The rest of this paper is organized as follows: in Section II, we briefly introduce the elementary knowledge about square root and some other notations. Then, we give the generalized definition of asymptotic square root in Section III. Specifically, explicit formulae of asymptotic square root operation for Type C.1 and C.2 pentanomials are given, the space and time complexities are also evaluated in Section IV. Section V describes a parallel exponentiation algorithm based on GPB squaring and such square root operators. Finally, some conclusions are drawn in Section VI.

II. PRELIMINARY

In this section, we briefly review Fong et al. and Rodríguez-Henríquez square root computation approaches as well as some related notations. In fact, these algorithms are two main ways for square root computation so far. We briefly review their approach and then analyze the limitation of square root operation.

A. THE FONG ET AL. METHOD

Fong *et al.* [7] provided explicit formulations for the square root operator, where the finite field $GF(2^m)$ is generated with an irreducible trinomial $f(x) = x^m + x^k + 1$. However, their approach works only if m is odd, k is even and $\lceil \frac{m-1}{4} \rceil \leq k < \lfloor \frac{m-1}{3} \rfloor$. Their algorithm is based on the following observation: Let $A = \sum_{i=0}^{m-1} a_i x^i$ be an arbitrary element in $GF(2^m)$ and $D \in GF(2^m)$ be the square root of A. We know that

$$\begin{aligned} D = A^{\frac{1}{2}} &= (\sum_{i=0}^{n} a_{2i}x^{2i} + \sum_{i=0}^{n-1} a_{2i+1}x^{2i+1})^{\frac{1}{2}} \\ &= (A_{even}^2 + x \cdot A_{odd}^2)^{\frac{1}{2}} \\ &= A_{even} + x^{\frac{1}{2}} \cdot A_{odd}, \end{aligned}$$

where $n = \frac{m-1}{2}, A_{even} = \sum_{i=0}^{n} a_{2i}x^i$ and $A_{odd} = \sum_{i=0}^{n-1} a_{2i+1}x^i$. Therefore, the square root of A can be obtained by

$$A^{\frac{1}{2}} = A_{even} + x^{\frac{1}{2}}A_{odd} \bmod f(x). \tag{1}$$

It is obvious that the computation of $x^{1/2}$ is crucial to the square root computation. Recall that in this case m is an odd number, Fong et al. found the following handy equation

$$x^{\frac{1}{2}} = \begin{cases} x^{\frac{m+1}{2}} + x^{\frac{k+1}{2}}, & k \text{ is odd}, \\ x^{-\frac{m-1}{2}}(x^{\frac{k}{2}} + 1), & k \text{ is even}. \end{cases} \tag{2}$$

Based upon (2), it is clear that if the middle coefficient k is odd, the square root computation can be calculated using a few additions and shift operations. However, if k is even, the square root computation becomes a little more computationally intensive. When put the limitation that $\lceil \frac{m-1}{4} \rceil \leq k < \lfloor \frac{m-1}{3} \rfloor$ to $f(x)$, the square root of A finally has the form [7]:

$$\begin{aligned} D = A_{even} + A_{odd} \Big(& x^{2k-n} + x^{n+1} + x^{k+n+1} \\ & + x^{\frac{5k}{2}-n} + x^{n+\frac{k}{2}+1} + x^{\frac{3k}{2}+n+1} \Big). \end{aligned}$$

The work in [7] did not give all the square root formulations as other cases of m, k will lead to more complicated expressions.

B. INVERSIVE MULTIPLICATIVE MATRIX APPROACH

Rodríguez-Henríquez *et al.* proposed an alternative method for deriving the square root formulae [8]. Their main strategy is based on an inversion of the Mastrovito matrix for square. In fact, field squaring is a special type of field multiplication, which can be performed by a matrix-vector multiplication. Let A be defined as previous subsection, and we have

$$C = A^2 \bmod f(x) = \mathbf{M}_A \cdot \mathbf{A},$$

where $\mathbf{A}$ is the coefficient vector of A and $\mathbf{M}_A$ is the Mastrovito matrix derived from A^2. Moreover, based on above identity, it follows that computing the square root of A means finding a field element $D = A^{1/2}$. Clearly, since $A = D^2 = \mathbf{M}_D \cdot \mathbf{D}$, we have

$$D = \mathbf{M}_D^{-1} \cdot \mathbf{A}.$$

Since the Mastrovito matrix construction has been fully studied in [9], [10], we can construct any corresponding $\mathbf{M}_D$ according to $f(x)$. The inverse matrix $\mathbf{M}_D^{-1}$ can also be obtained easily. Thus, this approach can apply to any type of $f(x)$ theoretically. Based on this approach, Rodríguez-Henríquez et al. had given all the square root formulation for all trinomials.

C. LIMITATION OF SQUARE ROOT

To sum up, Rodríguez-Henríquez approach can compute the classic square root efficiently. Nevertheless, it is easy to check that the computations of $\mathbf{M}_D^{-1}$ of subsection II.B highly rely on the terms contained in $f(x)$. If $f(x)$ is an irreducible trinomial, the organization of $\mathbf{M}_D$ is relatively simple, so that its inverse matrix $\mathbf{M}_D^{-1}$ can be obtained easily. But, with the increase of the terms contain in one irreducible polynomial, the square root computation became harder. Here, we give a toy example to illustrate the fact.

Given an irreducible pentanomial $f(x) = x^{15} + x^{13} + x^5 + x^2 + 1$, $A = \sum_{i=0}^{14} a_i x^i \in \mathbb{F}_2[x]/(f(x))$ is an arbitrary field element. Its square root formula is given by

$$\begin{aligned} A^{1/2} = {} & (a_7 + a_9 + a_{11} + a_5 + a_3 + a_1 + a_{13})x^{14} + (a_7 + a_9 \\ & + a_{11} + a_5 + a_3 + a_1)x^{13} + a_{11}x^{12} + a_9x^{11} + a_7x^{10} \\ & + a_5x^9 + (a_{13} + a_3)x^8 + (a_{11} + a_1 + a_{13} + a_{14})x^7 \\ & + (a_9 + a_{11} + a_{12} + a_{13})x^6 + (a_7 + a_9 + a_{11} + a_{10} \\ & + a_{13})x^5 + (a_1 + a_3 + a_8)x^4 + (a_1 + a_6)x^3 + a_4x^2 \\ & + (a_7 + a_9 + a_{11} + a_5 + a_3 + a_2 + a_1 + a_{13})x + a_0 \\ & + a_7 + a_9 + a_{11} + a_5 + a_3 + a_1 + a_{13}. \end{aligned}$$

In addition, the ordinary square formula is

$$\begin{aligned} A^2 = {} & (a_{10} + a_{11} + a_{13} + a_7 + a_8 + a_9)x^{14} + (a_{13} + a_{14})x^{13} \\ & + (a_{11} + a_{12} + a_{13} + a_{14} + a_6)x^{12} + a_{12}x^{11} + (a_{10} \\ & + a_{11} + a_{12} + a_{13} + a_{14} + a_5)x^{10} + a_{11}x^9 + (a_{10} \\ & + a_{11} + a_{12} + a_{13} + a_4 + a_9)x^8 + a_{10}x^7 + (a_{10} + a_{11} \\ & + a_{12} + a_{14} + a_3 + a_8 + a_9)x^6 + a_9x^5 + a_2x^4 + (a_{13} \\ & + a_{14} + a_8)x^3 + (a_1 + a_{14})x + (a_{10} + a_{11} + a_{12} \\ & + a_{14} + a_8 + a_9)x + a_0 + a_{14}. \end{aligned}$$

In parallel implementation, one can check that this square root costs 39 XOR gates and square costs 35 XOR gates (ignore the reused gates). Both square and square root require at least $3T_X$. Conversely, there exist a GPB squaring [19] in this field. By choosing $\omega = x^{10} + x^8 + 1$ as a GPB parameter, the GPB squaring only requires 27 XOR gates with $2T_X$ delay.

$$\begin{aligned} A^2\omega = {} & a_2x^{14} + (a_{13} + a_{14} + a_8)x^{13} + (a_1 + a_{14} + a_2)x^{12} \\ & + (a_{12} + a_7 + a_8)x^{11} + (a_0 + a_1 + a_{13} + a_{14})x^{10} \\ & + (a_{11} + a_6 + a_7)x^9 + (a_0 + a_{12} + a_{13})x^8 + (a_{10} \\ & + a_5 + a_6)x^7 + (a_{11} + a_{12})x^6 + (a_{14} + a_4 + a_5 \\ & + a_9)x^5 + (a_{10} + a_{11} + a_2)x^4 + (a_3 + a_4)x^3 + (a_1 \\ & + a_{10} + a_{14} + a_9)x + a_3x + a_9 + a_0 + a_{14}. \end{aligned}$$

We can check that both square and square root for irreducible pentanomial are less efficient than those of trinomials, but a square multiplying a specific parameter can be computed more efficiently. It is interesting to investigate whether or not there exist a square root with a parameter that can be easily computed. In [17], Li *et al.* proposed an alternative type of square root by multiplying a specific parameter to the classic square root. Such a type of square root imitates the Montgomery squaring operation and can achieve a relatively lower space and time complexity for all trinomials. In the following section, we generalize this notion and investigate such a square root for more kind of irreducible polynomials.

III. ASYMPTOTIC SQUARE ROOT

Consider a finite field $GF(2^m)$ defined by an arbitrary irreducible polynomial $f(x) = x^m + x^{k_{s-1}} + x^{k_{s-2}} + \cdots + x^{k_1} + 1$, where $k_{s-1} > k_{s-2} > \cdots > k_1 > 0$. Let $A = \sum_{i=0}^{m-1} a_i x^i$ be an arbitrary element of $GF(2^m)^*$ using PB representation. According to (1), to compute the square root of A, we first partition A into two parts according to the degree parity of its intermediate, i.e., $A = A_{even}^2 + xA_{odd}^2$. Once again, the element $x^{1/2}$ is a constant that can be pre-computed. Thus, the square root operation can be implemented as performing a field multiplication between A_{odd} and $x^{1/2}$, and then adding with A_{even}. However, the pre-computation of $x^{1/2}$ relies on the form of $f(x)$. More explicitly, we have the following identity:

$$\begin{aligned} 1 &= x^m + x^{k_{s-1}} + x^{k_{s-2}} + \cdots + x^{k_1}, \\ \Rightarrow x &= x^{m+1} + x^{k_{s-1}+1} + x^{k_{s-2}+1} + \cdots + x^{k_1+1}. \end{aligned}$$

When we calculate square root of x, it is necessary to know the parity of $m+1, k_{s-1}+1, \cdots, k_1+1$. If all these numbers are even, then the formula of $x^{1/2}$ is direct, i.e.,

$$x^{\frac{1}{2}} = x^{\frac{m+1}{2}} + x^{\frac{k_{s-1}+1}{2}} + x^{\frac{k_{s-2}+1}{2}} + \cdots + x^{\frac{k_1+1}{2}}.$$

Except this, even one of these numbers is odd, the formulation of $x^{1/2}$ is not straightforward. Provide that there are $t \leq s$ odd numbers $c_1, c_2, \cdots, c_t \in \{m+1, k_{s-1}+1, \cdots, k_1+1\}$, while $c_{t+1}, c_{t+2}, \cdots, c_s \in \{m+1, k_{s-1}+1, \cdots, k_1+1\}$ are

even, then we have

$$\begin{aligned} x^{\frac{1}{2}} &= x^{\frac{c_1-1}{2}+\frac{1}{2}} + \cdots + x^{\frac{c_t-1}{2}+\frac{1}{2}} + x^{\frac{c_{t+1}}{2}} + \cdots + x^{\frac{c_s}{2}}, \\ &\Rightarrow x^{\frac{1}{2}}(1 + x^{\frac{c_1-1}{2}} + \cdots + x^{\frac{c_t-1}{2}}) = x^{\frac{c_{t+1}}{2}} + \cdots + x^{\frac{c_s}{2}}, \\ &\Rightarrow x^{\frac{1}{2}} = (x^{\frac{c_{t+1}}{2}} + \cdots + x^{\frac{c_s}{2}}) \cdot (1 + x^{\frac{c_1-1}{2}} + \cdots + x^{\frac{c_t-1}{2}})^{-1}. \end{aligned}$$

Special case: If $k_1 = 1$, the deduction of $x^{1/2}$ is a slightly different:

$$\begin{aligned} x &= x^m + x^{k_{s-1}} + x^{k_{s-2}} + \cdots + x^{k_2} + 1, \\ &\Rightarrow x^{\frac{1}{2}} = x^{\frac{m}{2}} + x^{\frac{k_{s-1}}{2}} + x^{\frac{k_{s-2}}{2}} + \cdots + x^{\frac{k_2}{2}} + 1. \end{aligned}$$

Analogous with previous deduction, assume that $c_1, c_2, \cdots, c_t \in \{m, k_{s-1}, \cdots, k_2\}$ are odd numbers, while $c_{t+1}, c_{t+2}, \cdots, c_{s-1} \in \{m, k_{s-1}, \cdots, k_2\}$ are even numbers. Then,

$$\begin{aligned} x^{\frac{1}{2}} &= x^{\frac{c_1-1}{2}+\frac{1}{2}} + \cdots + x^{\frac{c_t-1}{2}+\frac{1}{2}} + x^{\frac{c_{t+1}}{2}} + \cdots + x^{\frac{c_{s-1}}{2}} + 1, \\ &\Rightarrow x^{\frac{1}{2}}(1 + x^{\frac{c_1-1}{2}} + \cdots + x^{\frac{c_t-1}{2}}) = 1 + x^{\frac{c_{t+1}}{2}} + \cdots + x^{\frac{c_{s-1}}{2}}, \\ &\Rightarrow x^{\frac{1}{2}} = (1 + x^{\frac{c_{t+1}}{2}} + \cdots + x^{\frac{c_{s-1}}{2}}) \\ &\quad \cdot (1 + x^{\frac{c_1-1}{2}} + \cdots + x^{\frac{c_t-1}{2}})^{-1}. \end{aligned}$$

From the above expression, it is obvious that in this case, the formula of $x^{1/2}$ depends on the inversion of $1 + x^{\frac{c_1-1}{2}} + \cdots + x^{\frac{c_t-1}{2}}$, where $c_1, c_2, \cdots, c_t$ are odd. Clearly, with the increase of these odd numbers, the formulation of $x^{1/2}$ became more complicated. Accordingly, the complexity of square root formulae also increases. Therefore, if we can simplify the form of $x^{1/2}$, the square root computation will be simplified as well. Inspired by SPB squaring, we multiply the square root by a proper factor and introduce a new type of square root operation. In above case, we compute $A^{1/2} \cdot \omega$ instead of $A^{1/2}$, where $\omega = 1 + x^{\frac{c_1-1}{2}} + \cdots + x^{\frac{c_t-1}{2}}$. Clearly, one can check that

$$\begin{aligned} A^{\frac{1}{2}} \cdot \omega &= A_{even} \cdot \omega + x^{\frac{1}{2}} A_{odd} \cdot \omega \\ &= A_{even} \cdot \omega + A_{odd} \cdot \varepsilon, \end{aligned} \tag{3}$$

where $\varepsilon = x^{\frac{c_{t+1}}{2}} + \cdots + x^{\frac{c_s}{2}}$. Please note that this square root is not equal to the original one. We call this square root as "Asymptotic Square Root". Its definition is given by

Definition 1: Assume that $f(x)$ is an irreducible polynomial over $\mathbb{F}_2$. Let A be an arbitrary element of $GF(2^m)^*$, which is generated by $f(x)$, and ω is defined as above. Then the asymptotic square root of A is defined as $A^{1/2} \cdot \omega$, while ω is named as the asymptotic factor.

As shown in (3), one can check that the computation of asymptotic square is relatively easy. More explicitly, for the space complexity, we have following proposition:

Proposition 1: If the degree of $f(x)$ is fixed, the space complexity of asymptotic square root computation is linear with the number of terms including in $f(x)$.

Proof: As shown previously, $f(x)$ consists of $s + 1$ nonzero terms. Based on the formulation of ω, ε, it is noteworthy that ω and ε totally contain s nonzero terms. Meanwhile, from (3), we can note that the computation of $A^{\frac{1}{2}} \cdot \omega$ is equivalent to adding s parts which consist of about $m/2$ terms. We immediately know this calculation costs at most $\frac{m(s-1)}{2}$, which is linear with s. Thus, we conclude the proposition.

In addition, in parallel implementation, its circuit delay highly relies on the number of terms in ω and ε. If ω and ε consist of the same number of terms, i.e., $s/2$ terms, the computation of (3) in parallel costs at most $(\lceil \log_2 s \rceil - 1)T_X$. Thus, one can roughly check that the less terms contained in $f(x)$, the more efficient asymptotic square root will be. It is straightforward that the asymptotic square root for trinomial is efficient. In the following sections, we show that besides trinomials, some specific type of polynomial can also develop efficient architecture for asymptotic square root computation. More explicitly, we investigate the explicit formulation of asymptotic square root for two newly proposed pentanomials and describe their application.

IV. ASYMPTOTIC SQUARE ROOT FOR TYPE C.1 AND C.2 PENTANOMIALS

In 2013, Cilardo [11] proposed two new types of irreducible pentanomials, i.e.,

- Type C.1: $x^m + x^{m-1} + x^k + x + 1$, $(m - 1 > k > 1)$,
- Type C.2: $x^m + x^{m-k_1} + x^{k_2} + x^{k_1}$, $(m - k_1 > k_2 > 2k_1 > 1)$.

One of the main reasons to recommend such pentanomials is that they are extremely abundant. There exists at least one Type C.1 or C.2 pentanomial for any degree less than $m \leq 10000$. On top of that, the bit-parallel multipliers using these types of pentanomials combined with GPB match or outperform the best multipliers for other commonly used pentanomials, e.g., Type I and II pentanomials [20], [21] from both the area and time point of view. Efficient squarer for Type C.1 pentanomial [18] and Type C.2 pentanomial [19] are also proposed. These squarers only require $2T_X$ delay for parallel implementation, which is faster than classic squaring.

In this section, we study the asymptotic square root for these types of pentanomials in order to build a parallel exponentiation algorithm which uses GPB squarer and square root simultaneously. Additionally, consider the reciprocal property of these types of pentanomials [11], in this contribution, we only investigate the case of $k < m - 1$ (Type C.1 pentanomial) or the case of $m - k_1 > 2k_2$ (Type C.2 pentanomial).

A. ASYMPTOTIC SQUARE ROOT FOR TYPE C.1 PENTANOMIAL

Firstly, let us consider finite field generated by an irreducible Type C.1 pentanomial $x^m + x^{m-1} + x^k + x + 1$. Based on related analysis in Section III, it is clear that

$$x = x^m + x^{m-1} + x^k + 1.$$

Thus, the formulation of $x^{1/2}$ depends on the parity of m, $m - 1$ and k. Note that in this case, the parity of m also determines the parity of $m - 1$. Combined with previous description in Section III, there are four cases need to be considered:

TABLE 1. Asymptotic factor ω and ε for different cases of Type C.1 pentanomial.

Case	ω	ε
m even, k odd, $n = m/2$	$1 + x^{n-1} + x^{\frac{k-1}{2}}$	$1 + x^n$
m even, k even, $n = m/2$	$1 + x^{n-1}$	$1 + x^n + x^{\frac{k}{2}}$
m, k odd, $n = (m-1)/2$	$1 + x^n + x^{\frac{k-1}{2}}$	$1 + x^n$
m odd, k even, $n = (m-1)/2$	$1 + x^n$	$1 + x^n + x^{\frac{k}{2}}$

Case 1: m even, k odd. In this case, we know that $2 \mid m$, but $2 \nmid m-1$ and $2 \nmid k$. Let $n = m/2$, then we have

$$\begin{aligned} x^{\frac{1}{2}} &= x^n + x^{n-1} \cdot x^{\frac{1}{2}} + x^{\frac{k-1}{2}} \cdot x^{\frac{1}{2}} + 1 \\ &\Rightarrow x^{\frac{1}{2}}(1 + x^{n-1} + x^{\frac{k-1}{2}}) = 1 + x^n, \\ &\Rightarrow x^{\frac{1}{2}} = (1 + x^n) \cdot (1 + x^{n-1} + x^{\frac{k-1}{2}})^{-1}. \end{aligned}$$

Let $\omega = 1 + x^{n-1} + x^{\frac{k-1}{2}}$ and $\varepsilon = 1 + x^n$, then

$$\begin{aligned} A^{\frac{1}{2}} \cdot \omega &= A_{even} \cdot \omega + A_{odd} \cdot \varepsilon \\ &= A_{even} \cdot (1 + x^{n-1} + x^{\frac{k-1}{2}}) + A_{odd} \cdot (1 + x^n), \end{aligned}$$

where $A_{even} = \sum_{i=0}^{n-1} a_{2i}x^i$ and $A_{odd} = \sum_{i=0}^{n-1} a_{2i+1}x^i$. Note that both A_{even} and A_{odd} consist of n terms. There is no overlap between A_{odd} and $A_{odd}x^n$. Meanwhile, A_{even} overlaps $A_{even}x^{n-1}$ with only one bit. Furthermore, since the degree of A_{even} and A_{odd} are $n-1$, $\deg(A_{even}\omega) = n-1+n-1 = 2n-2 < m-1$ and $\deg(A_{odd} + A_{odd}x^n) = n+n-1 = m-1$. Therefore, no further reduction is needed in above expression.

Now, we can write the explicit formulation for asymptotic square root of A in this case. Let $D = \sum_{i=0}^{m-1} d_i x^i$ denote the result of $A^{1/2} \cdot \omega$. Then, the coefficients of D are given by

$$d_i = \begin{cases} a_{2i+1} + a_{2i}, & 0 \le i \le \dfrac{k-3}{2}, \\ a_{2i-k+1} + a_{2i+1} + a_{2i}, & \dfrac{k-1}{2} \le i \le n-2, \\ a_{m-k-1} + a_{m-1} + a_{m-2} + a_0, & i = n-1, \\ a_{2i-k+1} + a_{2(i-n+1)} + a_{2(i-n)+1}, & n \le i \le n-1+\dfrac{k-1}{2}, \\ a_{2(i-n+1)} + a_{2(i-n)+1}, & n+\dfrac{k-1}{2} \le i \le m-2, \\ a_{m-1}, & i = m-1. \end{cases} \tag{4}$$

It can be verified that above expression has an associated cost of $\frac{3m}{2}$ XOR gates and two T_X delays in parallel implementation. The asymptotic square root computation for the rest of cases of Type C.1 pentanomial follows the same line as we did in Case 1. For simplicity, we do not present all the deduction details. The following table gives the values of ω and ε for different cases:

Then, the explicit asymptotic square root formulae are given in details.

Case 2: m even, k even, n = m/2.

$$d_i = \begin{cases} a_{2i+1} + a_{2i}, & 0 \le i \le \dfrac{k}{2} - 1, \\ a_{2i-k+1} + a_{2i+1} + a_{2i}, & \dfrac{k}{2} \le i \le n-2, \\ a_{m-k-1} + a_{m-1} + a_{m-2} + a_0, & i = n-1, \\ a_{2i-k+1} + a_{2(i-n+1)} + a_{2(i-n)+1}, & n \le i \le n-1+\dfrac{k}{2}, \\ a_{2(i-n+1)} + a_{2(i-n)+1}, & n+\dfrac{k}{2} \le i \le m-2, \\ a_{m-1}, & i = m-1. \end{cases} \tag{5}$$

Case 3: Both m and k are odd, n = (m − 1)/2. Let $A_{even} = \sum_{i=0}^{n} a_{2i}x^i$ and $A_{odd} = \sum_{i=0}^{n-1} a_{2i+1}x^i$, then,

$$d_i = \begin{cases} a_{2i+1} + a_{2i}, & 0 \le i \le \dfrac{k-3}{2}, \\ a_{2i-k+1} + a_{2i+1} + a_{2i}, & \dfrac{k-1}{2} \le i \le n-1, \\ a_{m-k} + a_1 + a_{m-1} + a_0, & i = n, \\ a_{2i-k+1} + a_{2(i-n)} + a_{2(i-n)+1}, & n+1 \le i \le n+\dfrac{k-1}{2}, \\ a_{2(i-n)} + a_{2(i-n)+1}, & n+\dfrac{k-1}{2}+1 \le i \le m-2, \\ a_{m-1}, & i = m-1. \end{cases} \tag{6}$$

Case 4: m odd, k even, n = (m − 1)/2.

$$d_i = \begin{cases} a_{2i+1} + a_{2i}, & 0 \le i \le \dfrac{k}{2} - 1, \\ a_{2i-k+1} + a_{2i+1} + a_{2i}, & \dfrac{k}{2} \le i \le n-1, \\ a_{m-k} + a_1 + a_{m-1} + a_0, & i = n, \\ a_{2i-k+1} + a_{2(i-n)} + a_{2(i-n)+1}, & n+1 \le i \le n+\dfrac{k}{2}-1, \\ a_{2(i-n)} + a_{2(i-n)+1}, & n+\dfrac{k}{2} \le i \le m-2, \\ a_{m-1}, & i = m-1. \end{cases} \tag{7}$$

B. ASYMPTOTIC SQUARE ROOT FOR TYPE C.2 PENTANOMIAL

Since Type C.2 pentanomial $x^m + x^{m-k_1} + x^{k_2} + x^{k_1} + 1$, $k_1 > 1$ has four uncertain parameters, the computation of $x^{1/2}$

TABLE 2. Asymptotic factor ω and ε for different cases of Type C.1 pentanomial.

Case	Parity of m	Parity of k_2	Parity of k_1
1	odd	odd	odd
2	even	even	odd
3	odd	odd	even
4	odd	even	odd
5	even	odd	odd
6	even	odd	even
7	odd	even	even

is little more complicated than that of Type C.1 pentanomial, which leads to more cases needed to be analyzed. Note that $x = x^{m+1} + x^{m-k_1+1} + x^{k_2+1} + x^{k_1+1}$. Then,

$$x^{\frac{1}{2}} = x^{\frac{m+1}{2}} + x^{\frac{m-k_1+1}{2}} + x^{\frac{k_2+1}{2}} + x^{\frac{k_1+1}{2}}. \tag{8}$$

Obviously, to obtain the formulae of ω and ε, we have to consider the parity of m, k_2, k_1. Here, the parity of $m - k_1$ is determined by m and k_1. In fact, irreducible Type C.2 pentanomials can be divided into seven categories, which are presented in the following table.

Analogous to Type C.1 pentanomial, we can give the asymptotic square root formulation for each case according to the parities of m, k_1, k_2. Without loss of generality, we analyze the first square root computation and put the rest of the cases in the appendices A and B.

Case 1: m, k_2 and k_1 are all odd. In this case, it is clear that $m+1, k_2+1$ and k_1+1 are all even numbers, while $m-k_1+1$ is odd. Then, (8) can be rewritten as:

$$x^{\frac{1}{2}}(1 + x^{\frac{m-k_1}{2}}) = x^{\frac{m+1}{2}} + x^{\frac{k_2+1}{2}} + x^{\frac{k_1+1}{2}}.$$

Thus, in this case we have $\omega = 1 + x^{\frac{m-k_1}{2}}$ and $\varepsilon = x^{\frac{m+1}{2}} + x^{\frac{k_2+1}{2}} + x^{\frac{k_1+1}{2}}$. The asymptotic square root here is

$$A^{\frac{1}{2}} \cdot \omega = A_{even} \cdot (1 + x^{\frac{m-k_1}{2}}) + A_{odd} \cdot (x^{n+1} + x^{\frac{k_2+1}{2}} + x^{\frac{k_1+1}{2}}),$$

in which $A_{even} = \sum_{i=0}^{n} a_{2i} x^i$, $A_{odd} = \sum_{i=0}^{n-1} a_{2i+1} x^i$ and $n = (m-1)/2$. One can check that $\deg A_{even} \cdot \omega = n + (m-k_1)/2 = m - (k_1+1)/2 \le m-1$ and $\deg A_{odd} \cdot \varepsilon = n-1+(m+1)/2 = m-1$. The degree of these two expressions are less than $m-1$, which means no further reduction is needed. Once again, we compute $D = \sum_{i=0}^{m-1} d_i x^i$ such that $D = A^{1/2} \cdot \omega \bmod x^m + x^{m-k_1} + x^{k_2} + x^{k_1} + 1$. Firstly, notice that $A_{odd} x^{n+1}$ does not overlap with A_{even}. Therefore, no logic gate is needed to compute $A_{even} + A_{odd} x^{n+1}$. Then, $m-k_1 > 2k_2, k_2 > 2k_1$, we have $\frac{m-k_1}{2} > k_2 > \frac{k_2+1}{2} > \frac{k_1+1}{2}$.

The asymptotic square root formula is given by

$$d_i = \begin{cases} a_{2i}, & 0 \le i \le \dfrac{k_1-1}{2}, \\ a_{2i} + a_{2i-k_1}, & \dfrac{k_1+1}{2} \le i \le \dfrac{k_2-1}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1}, & \dfrac{k_2+1}{2} \le i \le \dfrac{m-k_1-2}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1}, & \dfrac{m-k_1}{2} \le i \le \dfrac{m-1}{2}, \\ a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1} + a_{2i-m}, & \dfrac{m+1}{2} \le i \le \dfrac{m+k_1-2}{2}, \\ a_{2i-k_2} + a_{2i-m+k_1} + a_{2i-m}, & \dfrac{m+k_1}{2} \le i \le \dfrac{m+k_2-2}{2}, \\ a_{2i-m+k_1} + a_{2i-m}, & \dfrac{m+k_2}{2} \le i \le \dfrac{2m-k_1-1}{2}, \\ a_{2i-m}, & \dfrac{2m-k_1+1}{2} \le i \le m-1. \end{cases} \tag{9}$$

TABLE 3. Space and time complexities of asymptotic square root operations for Type C.1 and C.2 pentanomials.

Pentanomial	Cases	#XOR	Time Delay
Type C.1	m even	$\frac{3m}{2}$	$2T_X$
	m, k odd	$\frac{3m+1}{2}$	$2T_X$
	m odd, k even,	$\frac{3m-1}{2}$	$2T_X$
Type C.2	m odd, k_2 odd, k_1 odd	$\frac{2m+k_1-1}{2}$	$2T_X$
	m even, k_1 odd	$\frac{3m}{2}$	$2T_X$
	m odd, k_2 odd, k_1 even	$\frac{3m-1}{2}$	$2T_X$
	m odd, k_2 even,k_1 odd	$\frac{2m+k_1+1}{2}$	$2T_X$
	m even, k_2 odd, k_1 even	$\frac{2m+k_1}{2}$	$2T_X$
	m odd, k_2 even, k_1 even	$\frac{3m+1}{2}$	$2T_X$

Similarly, we can obtain all the asymptotic square root formulations. In appendix A, Table A1 presents the explicit formulations with respect to the factors ω and ε. In appendix B, expressions (13-18) present the explicit asymptotic square root formulations. One can check that all these formulae can be implemented in only $2T_X$, with no more than $\frac{3m}{2}$ XOR gates. In Table 3, we summarize the space and time

complexities of the asymptotic square root for all Type C.1 and C.2 pentanomials. For simplicity, we combined some cases with the same space and time complexities. Compared with trinomial, the asymptotic square roots in these cases have one more T_X delay. But Type C.1 and C.2 pentanomials are more abundant, this result will be interesting for the m where corresponding irreducible trinomial does not exist.

Furthermore, these operations coincidentally have at least the same circuit delay compared with SPB squarings [18], [19]. In the following section, we describe a new parallel exponentiation algorithm for Type C.1 and C.2 pentanomials, which utilize asymptotic square root and SPB squaring as mainly building blocks.

A Small Example: Still consider the finite field $GF(2^{15})$ defined by $x^{15}+x^{13}+x^5+x^2+1$. According to previous analysis, we have $\omega = x+1$ and $\varepsilon = x^8+x^7+x^3$. Given an arbitrary element $A = \sum_{i=0}^{14} a_i x^i$, the asymptotic square root of A is $A^{1/2}\cdot(x+1) = \sum_{i=0}^{14} d_i x^i$, where

$$d_0 = a_0, \quad d_5 = a_5 + a_{10} + a_8, \; d_{10} = a_5 + a_7,$$

$$d_1 = a_0 + a_2, \quad d_6 = a_7 + a_{10} + a_{12}, \; d_{11} = a_9 + a_7,$$

$$d_2 = a_2 + a_4, \quad d_7 = a_{14} + a_1 + a_{12} + a_9, \; d_{12} = a_9 + a_{11},$$

$$d_3 = a_1 + a_4 + a_6, \quad d_8 = a_1 + a_3 + a_{11} + a_{14},$$

$$d_{13} = a_{11} + a_{13}, d_4 = a_3 + a_6 + a_8, \quad d_9 = a_{13} + a_3 + a_5,$$

$$d_{14} = a_{13}.$$

Obviously, one can check that the computation of above coefficients d_i requires 22 XOR gates and $2T_X$ in parallel. Specifically, such a square root has the same delay compared with GPB squaring, as shown in Section II.

V. APPLICATION: EXPONENTIATION BASED ON SPB SQUARING AND ASYMPTOTIC SQUARE ROOT

In general, the square root has several applications in elliptic-curve cryptography (ECC), particularly as an important building block for implementing the point halving primitive. On top of that, square root can also be used to speed up the field exponentiation [8], which is important to scalar multiplication in ECC. In this section, we show that asymptotic square root can substitute the ordinary one in exponentiation algorithm under a slight modification.

In fact, the algorithm proposed in [8] is a parallel exponentiation algorithm over $GF(2^m)$. Such an algorithm comprises two sub-exponentiations that are based on squaring and square root operation, respectively. For square root based exponentiation, it mainly utilized the equation

$$A^{2^{m-i}} = A^{2^{-i}}, i = 1, 2, \cdots, m-1,$$

where $A \in GF(2^m)$ is an arbitrary nonzero element. The above equation can be easily proved using Fermat Little Theorem [3]. Let $e = (e_{m-1}, \cdots, e_1, e_0)_2$ be a m-bit nonzero integer. By substituting half the squaring operations with square root operations, the exponentiation A^e can be written as

$$\begin{aligned} A^e &= \prod_{i=0}^{m-1} A^{2^i e_i} = \prod_{i=n}^{m-1} A^{2^i e_i} \cdot \prod_{i=0}^{n-1} A^{2^i e_i} \\ &= \prod_{i=n}^{m-1} A^{2^{-(m-i)} e_i} \cdot \prod_{i=0}^{n-1} A^{2^i e_i}. \end{aligned} \tag{10}$$

where $0 < n < m$. In [8], the parameter n is chosen as $\lfloor \frac{m}{2} \rfloor$. One can check that the two sub-exponentiations presented in the above expression can be performed simultaneously, thus, the whole algorithm achieved better performance compared with the classic one. However, it is clear that the performance of such an algorithm depends on the efficiency of square root as well as squaring. In our case, the asymptotic square root has at least the same gates delay as SPB squaring [18], [19]. However, they both have extra parameters. Based on this fact, Algorithm 1 describes an improved exponentiation algorithm for A^e, which utilized SPB squaring and square root operator.

Proposition 2: The compensatory parameter ω^* in Algorithm 1 can be obtained as

$$\omega^* = \omega^{2^n-2} \cdot (r)^{2^n-1}. \tag{11}$$

Proof: According to steps 4-7 of Algorithm 1, it totally performs $m-n+1$ asymptotic square root operations and n SPB squarings. Each time we execute the body in the loops, there are one r (for GPB squaring) and one ω (for asymptotic square root) multiplying the results. In the end, it follows that the extra parameter is

$$\begin{aligned} &\omega \cdot \omega^{\frac{1}{2}} \cdot \omega^{\frac{1}{4}} \cdots \omega^{\frac{1}{2^{m-n}}} \cdot r \cdot r^2 \cdots r^{2^{n-1}} \\ &= \omega^{2-2^{-(m-n)}} \cdot r^{2^n-1} \\ &= \omega^{2-2^n} \cdot r^{2^n-1}. \end{aligned} \tag{12}$$

Since ω^* is the compensatory parameter used to correct the final result, we know that (12) is equal to the inversion of ω^*. Then, the result is direct.

Algorithm 1 Exponentiation Based on SPB Squaring and Asymptotic Square Root

Input: $A \in GF(2^m), f(x), e = (e_{m-1}, e_{m-2}, \cdots, e_1, e_0)_2$
Output: $B = A^e \bmod f(x)$

1: $B = C = 1$;
2: $e_m = 0$;
3: $n = \lfloor \frac{m}{2} \rfloor$;
4: **for** $i = n-1$ **down to** 0 **do** | **for** $j = n$ **to** m **do**
5: $B = B^2 \cdot r$; | $C = C^{\frac{1}{2}} \cdot \omega$;
6: **if** $e_i == 1$ **then** | **if** $e_j == 1$ **then**
7: $B = B \cdot A \bmod f(x)$; | $C = C \cdot A \bmod f(x)$;
8: **end if** | **end if**
9: **end for** | **end for**
10: $B = B \cdot C$;
11: $B = B \cdot \omega^*$;
12: **return** B;

Description: The parameter ω is the asymptotic factor while r is the SPB factor. The two procedures presented in steps 4-9 are running in parallel. Also notice that the choice of n in step 3 is slightly different from [8], as we found that this selection can make the number of squaring and square root almost equal, which can save the whole algorithm delay. Since we utilize the SPB squaring and asymptotic square root operator instead of the original ones, ω^* in step 11 is a compensatory parameter which is used to correct the final exponentiation. The form of ω^* is given in following proposition.

Proposition 3: Algorithm 1 is correct and it returns the exponentiation A^e.

Proof: According to the description in Algorithm 1 and Proposition 1, since the compensatory parameter ω^* will correct the final result influenced by adding factors to the squaring and square root, we can omit all these parameters here. On top of that, we know that this algorithm recursively computes A^{2^i}, $i = 0, 1, \cdot, n-1$ and $A^{2^{-i}}$, $i = 1, 2, \cdots, m-n$. Note that $n = \lfloor \frac{m}{2} \rfloor$ and $m-n = \lceil \frac{m}{2} \rceil$. Plus the final multiplication in Step 10, one can check that Algorithm 1 does compute all the operations contained in (10). We immediately obtain the conclusion.

Additionally, based on proposition 1, it is clear that the explicit formula of ω^* relies on the forms of ω and r, which vary according to the explicit form of $f(x)$. For example, Type C.1 pentanomial $x^{24} + x^{23} + x^{14} + x + 1$ defines the finite field $GF(2^{24})$. Please note that there is no irreducible trinomials nor Type-II pentanomials for this degree. We have $\omega^* = x^{21} + x^{20} + x^{19} + x^{16} + x^{15} + x^{11} + x^{10} + x^8 + x^2$.

Furthermore, the constant multiplication $B \cdot \omega^*$ can be performed using Mastrovito approach. Since ω^* is constant, the corresponding product matrix is fixed, no AND gate is needed. In the former example, ω^* only contains 9 non-zero terms, which lead to a little sparse Mastrovito matrix for this constant multiplication. Anyhow, such a multiplication requires no more than $m^2 - m$ XOR gates with at most $\lceil \log_2 m \rceil$ XOR gate delay.

APPENDIX A: THE ω AND ε OF ASYMPTOTIC SQUARE ROOT FOR TYPE C.2 PENTANOMIALS

TABLE 4. Asymptotic factor ω and ε for different cases of Type C.2 pentanomial.

Cases	ω	ε
m odd, k_2 odd, k_1 odd	$1 + x^{\frac{m-k_1}{2}}$	$x^{\frac{m+1}{2}} + x^{\frac{k_2+1}{2}} + x^{\frac{k_1+1}{2}}$
m even, k_2 even, k_1 odd	$1 + x^{\frac{k_2}{2}} + x^{\frac{m}{2}}$	$x^{\frac{k_1+1}{2}} + x^{\frac{m-k_1+1}{2}}$
m odd, k_2 odd, k_1 even	$1 + x^{\frac{k_1}{2}}$	$x^{\frac{k_2+1}{2}} + x^{\frac{m-k_1+1}{2}} + x^{\frac{m+1}{2}}$
m odd, k_2 even, k_1 odd	$1 + x^{\frac{k_2}{2}} + x^{\frac{m-k_1}{2}}$	$x^{\frac{k_1+1}{2}} + x^{\frac{m+1}{2}}$
m even, k_2 odd, k_1 odd	$1 + x^{\frac{m}{2}}$	$x^{\frac{k_1+1}{2}} + x^{\frac{k_2+1}{2}} + x^{\frac{m-k_1+1}{2}}$
m even, k_2 odd, k_1 even	$1 + x^{\frac{k_1}{2}} + x^{\frac{m-k_1}{2}} + x^{\frac{m}{2}}$	$x^{\frac{k_2+1}{2}}$
m odd, k_2 even, k_1 even	$1 + x^{\frac{k_1}{2}} + x^{\frac{k_2}{2}}$	$x^{\frac{m-k_1+1}{2}} + x^{\frac{m+1}{2}}$

APPENDIX B: ASYMPTOTIC SQUARE ROOT FORMULAE FOR TYPE C.2 PENTANOMIAL

Case 2: m is even, k_2 is even, k_1 is odd.

$$
\begin{aligned}
d_i = \begin{cases}
a_{2i}, \quad i = 0, 1, \ldots, \dfrac{k_1 - 1}{2}, \\
a_{2i} + a_{2i-k_1}, \\
\qquad i = \dfrac{k_1+1}{2}, \dfrac{k_1+3}{2}, \ldots, \dfrac{k_2-2}{2}, \\
a_{2i} + a_{2i-k_2} + a_{2i-k_1}, \\
\qquad i = \dfrac{k_2}{2}, \dfrac{k_2+2}{2}, \ldots, \dfrac{m-k_1-1}{2}, \\
a_{2i} + a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1}, \\
\qquad i = \dfrac{m-k_1+1}{2}, \dfrac{m-k_1+3}{2}, \ldots, \dfrac{m-2}{2}, \\
a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1} + a_{2i-m}, \\
\qquad i = \dfrac{m}{2}, \dfrac{m+2}{2}, \ldots, \dfrac{m+k_1-1}{2}, \\
a_{2i-k_2} + a_{2i-m+k_1} + a_{2i-m}, \\
\qquad i = \dfrac{m+k_1+1}{2}, \dfrac{m+k_1+3}{2}, \ldots, \dfrac{m+k_2-2}{2}, \\
a_{2i-m+k_1} + a_{2i-m}, \\
\qquad i = \dfrac{m+k_2}{2}, \dfrac{m+k_2+2}{2}, \ldots, \dfrac{2m-k_1-1}{2}, \\
a_{2i-m}, \\
\qquad i = \dfrac{2m-k_1+1}{2}, \dfrac{2m-k_1+3}{2}, \ldots, m-1.
\end{cases}
\end{aligned}
\tag{13}
$$

Case 3: m is odd, k_2 is odd, k_1 is even.

$$
\begin{aligned}
d_i = \begin{cases}
a_{2i}, \quad i = 0, 1, \ldots, \dfrac{k_1 - 2}{2}, \\
a_{2i} + a_{2i-k_1}, \\
\qquad i = \dfrac{k_1}{2}, \dfrac{k_1+2}{2}, \ldots, \dfrac{k_2-1}{2}, \\
a_{2i} + a_{2i-k_2} + a_{2i-k_1}, \\
\qquad i = \dfrac{k_2+1}{2}, \dfrac{k_2+3}{2}, \ldots, \dfrac{m-k_1-1}{2}, \\
a_{2i} + a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1}, \\
\qquad i = \dfrac{m-k_1+1}{2}, \dfrac{m-k_1+3}{2}, \ldots, \dfrac{m-1}{2}, \\
a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1} + a_{2i-m}, \\
\qquad i = \dfrac{m+1}{2}, \dfrac{m+3}{2}, \ldots, \dfrac{m+k_1-1}{2}, \\
a_{2i-k_2} + a_{2i-m+k_1} + a_{2i-m}, \\
\qquad i = \dfrac{m+k_1+1}{2}, \dfrac{m+k_1+3}{2}, \ldots, \dfrac{m+k_2-2}{2}, \\
a_{2i-m+k_1} + a_{2i-m}, \\
\qquad i = \dfrac{m+k_2}{2}, \dfrac{m+k_2+2}{2}, \ldots, \dfrac{2m-k_1-2}{2}, \\
a_{2i-m}, \\
\qquad i = \dfrac{2m-k_1}{2}, \dfrac{2m-k_1+2}{2}, \ldots, m-1.
\end{cases}
\end{aligned}
\tag{14}
$$

Case 4: m is odd, k_2 is even, k_1 is odd.

$$d_i = \begin{cases} a_{2i}, \quad i = 0, 1, \ldots, \dfrac{k_1-1}{2}, \\ a_{2i} + a_{2i-k_1}, \\ \qquad i = \dfrac{k_1+1}{2}, \dfrac{k_1+3}{2}, \ldots, \dfrac{k_2-2}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1}, \\ \qquad i = \dfrac{k_2}{2}, \dfrac{k_2+2}{2}, \ldots, \dfrac{m-k_1-2}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1}, \\ \qquad i = \dfrac{m-k_1}{2}, \dfrac{m-k_1+2}{2}, \ldots, \dfrac{m-1}{2}, \\ a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+1}{2}, \dfrac{m+3}{2}, \ldots, \dfrac{m+k_1-2}{2}, \\ a_{2i-k_2} + a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+k_1}{2}, \dfrac{m+k_1+2}{2}, \ldots, \dfrac{m+k_2-1}{2}, \\ a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+k_2+1}{2}, \dfrac{m+k_2+3}{2}, \ldots, \dfrac{2m-k_1-1}{2}, \\ a_{2i-m}, \\ \qquad i = \dfrac{2m-k_1+1}{2}, \dfrac{2m-k_1+3}{2}, \ldots, m-1. \end{cases} \tag{15}$$

Case 5: m is even, k_2 is odd, k_1 is odd.

$$d_i = \begin{cases} a_{2i}, \quad i = 0, 1, \ldots, \dfrac{k_1-1}{2}, \\ a_{2i} + a_{2i-k_1}, \\ \qquad i = \dfrac{k_1+1}{2}, \dfrac{k_1+3}{2}, \ldots, \dfrac{k_2-1}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1}, \\ \qquad i = \dfrac{k_2+1}{2}, \dfrac{k_2+3}{2}, \ldots, \dfrac{m-k_1-1}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1}, \\ \qquad i = \dfrac{m-k_1+1}{2}, \dfrac{m-k_1+3}{2}, \ldots, \dfrac{m-2}{2}, \\ a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m}{2}, \dfrac{m+2}{2}, \ldots, \dfrac{m+k_1-1}{2}, \\ a_{2i-k_2} + a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+k_1+1}{2}, \dfrac{m+k_1+3}{2}, \ldots, \dfrac{m+k_2-1}{2}, \\ a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+k_2+1}{2}, \dfrac{m+k_2+3}{2}, \ldots, \dfrac{2m-k_1-1}{2}, \\ a_{2i-m}, \\ \qquad i = \dfrac{2m-k_1+1}{2}, \dfrac{2m-k_1+3}{2}, \ldots, m-1. \end{cases} \tag{16}$$

Case 6: m is even, k_2 is odd, k_1 is even.

$$d_i = \begin{cases} a_{2i}, \quad i = 0, 1, \ldots, \dfrac{k_1-2}{2}, \\ a_{2i} + a_{2i-k_1}, \\ \qquad i = \dfrac{k_1}{2}, \dfrac{k_1+2}{2}, \ldots, \dfrac{k_2-1}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1}, \\ \qquad i = \dfrac{k_2+1}{2}, \dfrac{k_2+3}{2}, \ldots, \dfrac{m-k_1-2}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1}, \\ \qquad i = \dfrac{m-k_1}{2}, \dfrac{m-k_1+2}{2}, \ldots, \dfrac{m-2}{2}, \\ a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m}{2}, \dfrac{m+2}{2}, \ldots, \dfrac{m+k_1-2}{2}, \\ a_{2i-k_2} + a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+k_1}{2}, \dfrac{m+k_1+2}{2}, \ldots, \dfrac{m+k_2-1}{2}, \\ a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+k_2+1}{2}, \dfrac{m+k_2+3}{2}, \ldots, \dfrac{2m-k_1-2}{2}, \\ a_{2i-m}, \\ \qquad i = \dfrac{2m-k_1}{2}, \dfrac{2m-k_1+2}{2}, \ldots, m-1. \end{cases} \tag{17}$$

Case 7: m is odd, k_2 is even, k_1 is even.

$$d_i = \begin{cases} a_{2i}, \quad i = 0, 1, \ldots, \dfrac{k_1-2}{2}, \\ a_{2i} + a_{2i-k_1}, \\ \qquad i = \dfrac{k_1}{2}, \dfrac{k_1+2}{2}, \ldots, \dfrac{k_2-2}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1}, \\ \qquad i = \dfrac{k_2}{2}, \dfrac{k_2+2}{2}, \ldots, \dfrac{m-k_1-1}{2}, \\ a_{2i} + a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1}, \\ \qquad i = \dfrac{m-k_1+1}{2}, \dfrac{m-k_1+3}{2}, \ldots, \dfrac{m-1}{2}, \\ a_{2i-k_2} + a_{2i-k_1} + a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+1}{2}, \dfrac{m+3}{2}, \ldots, \dfrac{m+k_1-1}{2}, \\ a_{2i-k_2} + a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+k_1+1}{2}, \dfrac{m+k_1+3}{2}, \ldots, \dfrac{m+k_2-1}{2}, \\ a_{2i-m+k_1} + a_{2i-m}, \\ \qquad i = \dfrac{m+k_2+1}{2}, \dfrac{m+k_2+3}{2}, \ldots, \dfrac{2m-k_1-2}{2}, \\ a_{2i-m}, \\ \qquad i = \dfrac{2m-k_1}{2}, \dfrac{2m-k_1+2}{2}, \ldots, m-1. \end{cases} \tag{18}$$

VI. CONCLUSION

In this paper, we introduce the notion of asymptotic square root for all polynomials. Particularly, we have proposed a new type of asymptotic square root operation for two classes of irreducible pentanomials. By choosing a proper factor, the proposed scheme has only 2 T_X delays and its space complexity matches the best squaring of the same kind. As an important application, we show that this type of square root combined with GPB squaring can speed up parallel exponentiation algorithm, which is based on ordinary squaring and square root.

REFERENCES

[1] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. New York, NY, USA: Cambridge Univ. Press, 1994.
[2] R. Lidl and H. Niederreiter, *Finite Fields*. New York, NY, USA: Cambridge Univ. Press, 1996.
[3] J. von Zur Gathen and J. Gerhard, *Modern Computer Algebra*, 3rd ed. New York, NY, USA: Cambridge Univ. Press, 2013.
[4] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. New York, NY, USA: Springer-Verlag, 2004.
[5] R. Schroeppel, C. Beaver, R. Gonzales, R. Miller, and T. Draelos, "A low-power design for an elliptic curve digital signature chip," in *Proc. 4th Int. Workshop Cryptograph. Hardware Embedded Syst.*, in Lecture Notes in Computer Science, vol. 2523, Feb. 2003, pp. 366–380.
[6] F. Rodríguez-Henríquez, G. Morales-Luna, N. A. Saqib, and N. Cruz-Cortés, "Parallel Itoh–Tsujii multiplicative inversion algorithm for a special class of trinomials," *Des. Codes Cryptogr.*, vol. 45, no. 1, pp. 19–37, Jun. 2007.
[7] K. Fong, D. Hankerson, J. López, and A. Menezes, "Field inversion and point halving revisited," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 1047–1059, Aug. 2004.
[8] F. Rodríguez-Henríquez, G. Morales-Luna, and J. López, "low-complexity bit-parallel square root computation over $GF(2^m)$ for all trinomials," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 472–480, Apr. 2008.
[9] B. Sunar and C. K. Koc, "Mastrovito multiplier for all trinomials," *IEEE Trans. Comput.*, vol. 48, no. 5, pp. 522–527, May 1999.
[10] T. Zhang and K. K. Parhi, "Systematic design of original and modified Mastrovito multipliers for general irreducible polynomials," *IEEE Trans. Comput.*, vol. 50, no. 7, pp. 734–749, Jul. 2001.
[11] A. Cilardo, "Fast parallel $GF(2^m)$ polynomial multiplication for all degrees," *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 929–943, May 2013.
[12] H. Wu, "Montgomery multiplier and squarer for a class of finite fields," *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 521–529, May 2002.
[13] R. Schoof, "Elliptic curves over finite fields and the computation of square roots p," *Math. Comput.*, vol. 44, no. 170, pp. 483–494, Apr. 1985.
[14] S. Müller, "On the computation of square roots in finite fields," *Des. Codes Cryptogr.*, vol. 31, no. 3, pp. 301–312, Mar. 2004.
[15] E. Ozdemir, "Computing square roots in finite fields," *IEEE Trans. Inf. Theory*, vol. 59, no. 9, pp. 5613–5615, Sep. 2013.
[16] P. Nguyen, "A montgomery-like square root for the number field sieve," in *Proc. Int. Algorithmic Number Theory Symp. (ANTS)*, in Lecture Notes in Computer Science, vol. 1423. Berlin, Germany: Springer, May 2006, pp. 151–168.
[17] Y. Li, Y. Zhang, and X. Guo, "Fast montgomery-like square root computation for all trinomials," *IEICE Trans. Fundamentals*,vol. E102.A, no. 1, pp. 307–309, Jan. 2019.
[18] X. Xiong and H. Fan, "$GF(2^n)$ bit-parallel squarer using generalised polynomial basis for new class of irreducible pentanomials," *Electron. Lett.*, vol. 50, no. 9, pp. 655–657, Apr. 2014.
[19] Q. Chen, Y. Li, and C. Qi, "Efficient $GF(2^m)$ squarer for type C.2 pentanomial," *Electron. Lett.*, vol. 54, no. 13, pp. 829–831, May 2018.
[20] F. Rodríguez-Henrìquez, and Ç. K. Koç, "Parallel multipliers based on special irreducible pentanomials," *IEEE Trans. Comput.*, vol. 52, no. 12, pp. 1535–1542, Dec. 2003.
[21] S.-M. Park, "Explicit formulae of polynomial basis squarer for pentanomials using weakly dual basis," *Integration*, vol. 45, pp. 205–210, Mar. 2012.

YU ZHANG received the B.Sc. degree from the Henan University of Economics and Law, in 2008, and the Ph.D. degree from the Huazhong University of Science and Technology, in 2015. Since 2016, he has been with the Department of Computer Science and Information Technology, Xinyang Normal University, where he is currently a Lecturer. His major interests include information security, cryptography, and information retrieval.

YIN LI received the B.Sc. degree in information engineering and the M.Sc. degree in cryptography from Information Engineering University, Zhenzhou, in 2004 and 2007, respectively, and the Ph.D. degree in computer science from Shanghai Jiaotong University (SJTU), Shanghai, in 2011. He held a postdoctoral position with the Department of Computer Science, Ben-Gurion University of the Negev, Israel. He is currently a Lecturer with the Department of Computer Science and Technology, Xinyang Normal University, Henan, China. His current research interests include algorithm and architectures for computation in finite field, computer algebra, secure cloud computing.

QING CHEN received the B.Sc. degree in mathematics from Xinyang Normal University, in 2016, where she is currently pursuing the M.D. degree with the Department of Mathematics. Her research interests include computer algebra and cryptography.