# New bit-parallel Montgomery multiplier for trinomials using squaring operation

Yin Li,Yiyang Chen

**Abstract**—In this paper, a new bit-parallel Montgomery multiplier for $GF(2^m)$ is presented, where the field is generated with an irreducible trinomial. We first present a slightly generalized version of a newly proposed divide and conquer approach. Then, by combining this approach and a carefully chosen Montgomery factor, the Montgomery multiplication can be transformed into a composition of small polynomial multiplications and Montgomery squarings, which are simpler and more efficient. Explicit complexity formulae in terms of gate counts and time delay of our architecture are investigated. As a result, the proposed multiplier has generally 25% lower space complexity than the fastest multipliers, with time complexity as good as or better than previous Karatsuba-based multipliers for the same class of fields. Among the five irreducible polynomials recommended by NIST for the ECDSA (Elliptic Curve Digital Signature Algorithm), there are two trinomials which are available for our architecture. We show that our proposal outperforms the previous best known results if the space and time complexity are both considered.

**Index Terms**—Montgomery multiplication, squaring, bit-parallel, trinomials.

## 1 Introduction

Efficient hardware implementation of multiplication over $GF(2^m)$ is very important in many areas such as coding theory, computer algebra and public key cryptosystems [1], [2]. Nowadays, more and more circuit gates can be located on a single chip which make the bit-parallel architectures possible and reasonable. During recent years, a number of bit-parallel $GF(2^m)$ multiplier schemes and architectures have been proposed to achieve the higher computation speed or lower area complexity. They have covered extensive cases with respect to different bases representation [4], [5] and generating polynomials [6], [7], [8].

Montgomery multiplication is an important algorithm which was originally used for fast modular integer multiplication [9] and then extended to the field multiplication over $GF(2^m)$ [10] and $GF(p^m)$ with $p > 2$ [11]. In [10], Koç and Acar have introduced a class of algorithms for software implementation of Montgomery multiplication. They argued that Montgomery multiplication can be implemented efficiently if the Montgomery factor is chosen properly. The hardware implementation of the Montgomery multiplication is investigated in [12], [13]. The Montgomery factor in these literatures are selected as $x^m$. In [14], Wu has proposed a new bit-parallel Montgomery multiplier for irreducible trinomials using a different factor. His scheme is based on the a slightly generalized method proposed in [10] and showed that the Montgomery factor is chosen as the middle term of the trinomial $x^m + x^k + 1$ can result in efficient bit-parallel multiplier and squarer which are at least as good as previous proposals. Also in the literatures, some systolic architectures are proposed for the Montgomery multiplication for trinomials, e.g., [15], [16], [17]. Hariri and Reyhani-Masoleh [18] have further improved Wu's proposal. Besides new recommendation of the Montgomery factor, fast bit-serial and bit-parallel multiplier architectures are also given for irreducible trinomials and pentanomials. It is argued that their scheme matches the best known result reported in the


- *Yin Li is with Department of Computer Science and Technology, Xinyang Normal University, Henan, China. E-mail: yunfeiyangli@gmail.com.*
- *Yiyang Chen is with Department of Computer Science and operational Research, Montreal University, Montreal, Canada. E-mail: chenyiyanginfo@gmail.com.*

literatures.

Hariri and Reyhani-Masolehs scheme is very fast, but their architecture costs about $2m^2$ circuit gates. In this paper, our work is devoted to designing a bit-parallel non-systolic Montgomery multiplier for trinomials, which obtains a trade-off between the space and time complexity. We start describing a slightly extended Park et al. algorithm [19], referred as PCHS algorithm. Then, by combining this algorithm with Montgomery squaring operations, new bit-parallel multiplier architecture is proposed. The main contributions of our work are as follows:

- The space complexity of our proposal is about 25% less than any other Montgomery or Mastrovito multipliers for trinomials, and matches the Karatsuba multiplier proposed by Elia [23].
- Besides, the time complexity of our proposal is slightly higher than the fastest multipliers, but no more than $2T_X$.
- For the range $m \in [100, 1203]$ and $k \leq m/2$, there are 1405 irreducible trinomials. For 1061 trinomials, its time complexity is equal to $T_A + (2 + \lceil \log_2 m \rceil)T_X$, for other trinomials, it is only $1T_X$ more.

The remainder of this paper is organized as follows: In Section 2, we briefly review the PCHS algorithm and the Montgomery squaring operation over $GF(2^m)$. Then we describe a slightly extended PCHS algorithm. Based on it, a new bit-parallel Montgomery multiplier is developed in Section 3. In Section 4, we further analyze its complexity and present a comparison between our proposal and some others. Finally, some conclusions are drawn.

## 2 PRELIMINARY

In this section, we briefly introduce the basic ingredients used in our algorithm, including the PCHS algorithm and Montgomery squaring over $GF(2^m)$.

### 2.1 The PCHS Algorithm

Recently, Park et al. [19] proposed a new divide and conquer approach for odd degree polynomial multiplication. This approach is analogous to Karatsuba algorithm but divides a polynomial according to exponent parity of the indeterminate. Assume that $A = \sum_{i=0}^{m-1} a_i x^i$ and $B = \sum_{i=0}^{m-1} b_i x^i$ be two polynomials over $\mathbb{F}_2[x]$ such that $m$ is an odd integer. $A, B$ can be partitioned into:

$$A = A_1^2 + xA_2^2 \text{ and } B = x^{-1}B_1^2 + B_2^2$$

respectively, where

$$A_1 = \sum_{i=0}^{(m-1)/2} a_{2i}x^i,\ A_2 = \sum_{i=0}^{(m-3)/2} a_{2i+1}x^i,$$
$$B_1 = \sum_{i=1}^{(m-1)/2} b_{2i-1}x^i,\ B_2 = \sum_{i=0}^{(m-1)/2} b_{2i}x^i.$$

Then the polynomial multiplication can be rewritten as:

$$\begin{aligned} AB &= (A_1^2 + xA_2^2)(x^{-1}B_1^2 + B_2^2) \\ &= x^{-1}(A_1B_1)^2 + x(A_2B_2)^2 + (A_1B_2)^2 + (A_2B_1)^2 \\ &= x^{-1}(A_1B_1)^2 + x(A_2B_2)^2 + (A_1B_1)^2 + (A_2B_2)^2 \\ &\quad + [(A_1 + A_2)(B_1 + B_2)]^2 . \end{aligned} \tag{1}$$

It is clear that Equation (1) converts the $m$-term polynomial multiplication into three $\frac{m+1}{2}$ (or $\frac{m-1}{2}$)-term polynomial multiplications and squarings at the cost of three extra additions. For finite field multiplications, this formula can be combined with fast squaring operation together to construct efficient multiplier.

The authors utilized the fast squaring formulae for two types of special pentanomials. However, their scheme is a little complicated as the related squaring is built on transformations between weak dual basis (WDB) and polynomial basis (PB) [20]. Actually, the Montgomery squaring for trinomial is very simple and efficient. In the following sections, we will describe new bit-parallel multiplier for irreducible trinomials based on the PCHS algorithm and Montgomery squaring operation.

### 2.2 Montgomery Squaring over $GF(2^m)$

The Montgomery squaring operation derives from Montgomery multiplication and is defined by $A^2(x)R^{-1}(x) \bmod f(x)$, where $f(x)$ is an irreducible polynomial generating $GF(2^m)$,

$A(x), R(x) \in GF(2^m)$ and $R(x)$ is a fixed element named as Montgomery factor. The general algorithm for Montgomery squaring is studied in [10] where $f(x)$ is an arbitrary irreducible polynomial and $R(x)$ is selected as $x^m$. An optimized Montgomery squaring is proposed by Wu [14] for irreducible trinomials $x^m+x^k+1$. This squaring is designed using $R(x) = x^k$ and the corresponding circuit delay is $T_X$, whereas the squaring in polynomial basis costs more circuit gates and has a delay of as most $2T_X$. The main reason is the factor $R^{-1}(x) = x^{-k}$ could simplify the modular reduction related to $x^m + x^k + 1$. Similar trick is also applied in some special types of irreducible pentanomial in [22].

## 3 New field multiplication using Montgomery squaring operation

In this section, we present a new Montgomery multiplication formula for irreducible trinomials using a slightly generalized PCHS algorithm.

Suppose that the field $GF(2^m)$ is defined by an irreducible trinomial $f(x) = x^m+x^k+1$ with a root $x$, and the field elements are represented using polynomial basis $\{1, x, \cdots, x^{m-1}\}$. From now on, we only take account of $f(x) = x^m + x^k+1$ where $1 \le k \le m/2$, as there always exist irreducible trinomial $f(x) = x^m + x^{m-k} + 1$ by the reciprocal property [3]. Let $A, B \in GF(2^m)$ be two arbitrary elements in polynomial basis representation:

$$A = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0,$$
$$B = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1x + b_0,$$

where $a_i,\ b_i \in \mathbb{F}_2$.

Denoted by $x^h$ $(1 \le h < m)$ the Montgomery factor, the Montgomery multiplication (MM) over $GF(2^m)$ is given by:

$$A \cdot B \cdot x^{-h} \bmod f(x). \tag{2}$$

### 3.1 Extended PCHS Algorithm

According to the parity of $m$, we consider two following cases.

*$m$ is odd.* Let

$$A = A_1^2 + xA_2^2, \quad B = x^{-1}B_1^2 + B_2^2,$$

where

$$\begin{cases} A_1 = \sum_{i=0}^{\frac{m-1}{2}} a_{2i}x^i,\ A_2 = \sum_{i=0}^{\frac{m-3}{2}} a_{2i+1}x^i, \\ B_1 = \sum_{i=1}^{\frac{m-1}{2}} b_{2i-1}x^i,\ B_2 = \sum_{i=0}^{\frac{m-1}{2}} b_{2i}x^i. \end{cases}$$

Then (2) can be rewritten as

$$\begin{aligned} ABx^{-h} &= \left[(A_1^2 + xA_2^2)(x^{-1}B_1^2 + B_2^2)\right] x^{-h} \\ &= [x^{-1}(A_1B_1)^2 + x(A_2B_2)^2 + (A_1B_2)^2 \\ &\quad + (A_2B_1)^2]x^{-h} \\ &= [x^{-1}(A_1B_1)^2 + x(A_2B_2)^2 + (A_1B_1)^2 \\ &\quad + (A_2B_2)^2 + (CD)^2]x^{-h} \\ &= (A_1B_1)^2x^{-h}(1+x^{-1}) + (CD)^2x^{-h} \\ &\quad + (A_2B_2)^2x^{-h}(1 + x), \end{aligned}$$

where $C = A_1 + A_2, D = B_1 + B_2$.

*$m$ is even.* This case is a little different from the previous case. We partition $A, B$ as follows:

$$A = A_1^2 + xA_2^2, \quad B = B_1^2 + xB_2^2,$$

where

$$\begin{cases} A_1 = \sum_{i=0}^{\frac{m}{2}-1} a_{2i}x^i,\ A_2 = \sum_{i=0}^{\frac{m}{2}-1} a_{2i+1}x^i, \\ B_1 = \sum_{i=0}^{\frac{m}{2}-1} b_{2i}x^i,\ B_2 = \sum_{i=0}^{\frac{m}{2}-1} b_{2i+1}x^i. \end{cases}$$

In this case, (2) is written as

$$\begin{aligned} ABx^{-h} &= \left[(A_1^2 + xA_2^2)(B_1^2 + xB_2^2)\right] x^{-h} \\ &= \left[(A_1^2 + xA_2^2)(x^{-1}B_1^2 + B_2^2)\right] x^{-h+1} \\ &= [x^{-1}(A_1B_1)^2 + x(A_2B_2)^2 + (A_1B_2)^2 \\ &\quad + (A_2B_1)^2]x^{-h+1} \\ &= [x^{-1}(A_1B_1)^2 + x(A_2B_2)^2 + (A_1B_1)^2 \\ &\quad + (A_2B_2)^2 + (CD)^2]x^{-h+1} \\ &= (A_1B_1)^2x^{-h+1}(1+x^{-1}) + (CD)^2x^{-h+1} \\ &\quad + (A_2B_2)^2x^{-h+1}(1 + x), \end{aligned}$$

where $C = A_1 + A_2, D = B_1 + B_2$.

The two expressions as above both transform Montgomery multiplication into the three squaring operations. We can choose suitable factor $x^h$ in order to obtain the simplest implementation. It is argued that the squaring $V^2(x)x^{-k} \bmod x^m + x^k + 1$ and $V^2(x)x^{-k+1} \bmod x^m + x^k + 1$ have the simplest modular reduction [14], [18]. Therefore, we choose the Montgomery factor $x^h$ as follows:

$$h = \begin{cases} k, & m \text{ is odd}, \\ k+1, & m \text{ is even}. \end{cases} \tag{3}$$

As a result, the Montgomery multiplication of two cases have the same transformation.

$$ABx^{-h} = (A_1B_1)^2x^{-k}(1+x^{-1}) + (A_2B_2)^2x^{-k}(1+x) + (CD)^2x^{-k}. \quad (4)$$

Meanwhile, its Montgomery squarings have one of the best factors. Note that the degrees of $A_1B_1, A_2B_2$ and $CD$ are at most $m-1$. From now on, the following notations are used:

$$A_1B_1 = \sum_{i=0}^{m-1} c_ix^i, A_2B_2 = \sum_{i=0}^{m-1} d_ix^i, CD = \sum_{i=0}^{m-1} e_ix^i,$$

$$S_1 = (A_1B_1)^2x^{-k}(1+x^{-1}) \bmod f(x) = \sum_{i=0}^{m-1} r_ix^i,$$

$$S_2 = (A_2B_2)^2x^{-k}(1+x) \bmod f(x) = \sum_{i=0}^{m-1} s_ix^i,$$

$$S_3 = (CD)^2x^{-k} \bmod f(x) = \sum_{i=0}^{m-1} t_ix^i.$$

Here, notice that if $m$ is odd, $c_0 = 0$ and $d_{m-1} = 0$, if $m$ is even, we have $c_{m-1} = d_{m-1} = 0$. The new Montgomery multiplication can be summarized in following algorithm:

**Algorithm 1** New Bit-parallel MM

**Input:** $A, B \in GF(2^m), f(x)$
**Output:** $ABx^{-h} \bmod f(x)$

1: Partition $A, B$ according to (4)
2: Implement $S_1, S_2, S_3$ in parallel
3: Compute $S_1 + S_2 + S_3$ in parallel

We then consider the detailed computation of $S_1, S_2$ and $S_3$, respectively.

### 3.2 The Complexities of $A_1B_1, A_2B_2$

First we briefly analyze the complexities of the products $A_1B_1$ and $A_2B_2$ which will be used in the computation of $S_1, S_2$. According to previous description, the coefficients $c_i$s of $A_1B_1 = \sum_{i=0}^{m-1} c_ix^i$ are given as follows:

*$m$ is odd.*

$$c_i = \begin{cases} 0, & i = 0, \\ \sum_{j=0}^{i-1} a_{2j}b_{2(i-j)-1}, & 1 \le i \le \frac{m-1}{2}, \\ \sum_{j=i-\frac{m-1}{2}}^{\frac{m-1}{2}} a_{2j}b_{2(i-j)-1}, & \frac{m+1}{2} \le i \le m-1. \end{cases} \quad (5)$$

*$m$ is even.*

$$c_i = \begin{cases} \sum_{j=0}^{i} a_{2j}b_{2(i-j)}, & 0 \le i \le \frac{m}{2}-1, \\ \sum_{j=i-\frac{m}{2}+1}^{\frac{m}{2}-1} a_{2j}b_{2(i-j)}, & \frac{m}{2} \le i \le m-2. \end{cases} \quad (6)$$

More explicitly, the gate count and time delay for implementation of each $c_i$ in (5) are presented in Table 1. It is easy to check that the computation of $c_i$s totally cost $\frac{m^2-1}{4}$ AND and $\frac{m^2-4m+3}{4}$ XOR gates with path delay $T_A + \lceil \log_2 \left(\frac{m-1}{2}\right) \rceil T_X$.

When $m$ is even, it requires $\frac{m^2}{4}$ AND and $\frac{m^2-4m+4}{4}$ XOR gates with path delay $T_A + \lceil \log_2 \left(\frac{m}{2}\right) \rceil T_X$.

Similarly, we can easily obtain the space and time complexity related to $A_2B_2$ which are the same as those of $A_1B_1$.

### 3.3 Different Cases

Then we consider the computations of $S_1, S_2$ and $S_3$. According to previous description, the key computation of $S_1, S_2, S_3$ is the Montgomery squaring related to $A_1B_1$, $A_2B_2$ and $CD$. This operation has been fully studied through the explicit formulations [14] which are varied according to the range of $m$ and $k$. Hence, we consider six cases:

1) $m$ is odd, $k$ is odd, $1 \le k \le \frac{m-3}{2}$,
2) $m$ is odd, $k$ is odd, $k = \frac{m-1}{2}$,
3) $m$ is odd, $k$ is even, $1 \le k \le \frac{m-3}{2}$,
4) $m$ is odd, $k$ is even, $k = \frac{m-1}{2}$,
5) $m$ is even, $k$ is odd, $m > 2k$,
6) $m$ is even, $k$ is odd, $m = 2k$.

The above six cases correspond to different squaring formulae, resulting different expression of $S_1, S_2$ and $S_3$. For the sake of the length of the paper, we only analyze two representative cases, i.e., case 1 and case 5.[1]

### 3.4 The Computation of $S_1, S_2$

*Case 1:* Denote $\sum_{i=0}^{m-1} z_ix^i$ as the Montgomery squaring $(A_1B_1)^2x^{-k} \bmod f(x)$, we have follow-

1. We can follow a similar line of approaches used in case 1 and case 5.

TABLE 1
The computation complexity of $c_i$

| $c_i$ | #AND | #XOR | Delay |
|---|---|---|---|
| $c_0 = 0$ | 0 | 0 | - |
| $c_1 = a_0 b_1$ | 1 | 0 | $T_A$ |
| $c_2 = a_0 b_3 + a_2 b_1$ | 2 | 1 | $T_A + T_X$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $c_{\frac{m-1}{2}} = a_0 b_{m-2} + \cdots + a_{m-3} b_1$ | $\frac{m-1}{2}$ | $\frac{m-3}{2}$ | $T_A + (\lceil \log_2(\frac{m-1}{2}) \rceil) T_X$ |
| $c_{\frac{m+1}{2}} = a_2 b_{m-2} + \cdots + a_{m-1} b_1$ | $\frac{m-1}{2}$ | $\frac{m-3}{2}$ | $T_A + (\lceil \log_2(\frac{m-1}{2}) \rceil) T_X$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $c_{m-1} = a_{m-1} b_{m-2}$ | 1 | 0 | $T_A$ |
| Total | $\frac{m^2-1}{4}$ | $\frac{m^2-4m+3}{4}$ | $T_A + \lceil \log_2 \left(\frac{m-1}{2}\right) \rceil T_X$ |

ing expression using related formula in [14]:

$$z_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{m+k+i}{2}}, & i = 0, 2, \cdots, k-1; \\ c_{\frac{m+k+i}{2}}, & i = k+1, k+3, \cdots, m-k-2; \\ c_{\frac{k-m+i}{2}}, & i = m-k, m-k+2 \cdots, m-1; \\ c_{\frac{k+i}{2}}, & i = 1, 3, \cdots, k-2; \\ c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}}, & i = k, k+2, \cdots, m-2. \end{cases} \tag{7}$$

Since $x^m + x^k = 1$, we have $x^{-1} = x^{m-1} + x^{k-1}$. It follows that:

$$\begin{aligned} S_1 &= \sum_{i=0}^{m-1} z_i x^i (1 + x^{-1}) \bmod f(x) \\ &= \sum_{i=0, i \neq k-1}^{m-2} (z_i + z_{i+1}) x^i \\ &\quad + (z_{k-1} + z_k + z_0) x^{k-1} + (z_{m-1} + z_0) x^{m-1}. \end{aligned} \tag{8}$$

Then we substitute $z_i$ with the expressions in (7). Note that $c_0 = 0$, the coefficients of $S_1$ are given by:

$$r_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{k+i+1}{2}}, & i = 0, 2, \cdots, k-1; \\ c_{\frac{m+k+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{m+i+1}{2}}, & i = k+1, k+3, \\ & \quad \cdots, m-k-2; \\ c_{\frac{k-m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{m+i+1}{2}}, & i = m-k, m-k+2, \\ & \quad \cdots, m-3; \\ c_{\frac{k-1}{2}} + c_{\frac{m+k}{2}}, & i = m-1; \\ c_{\frac{k+i}{2}} + c_{\frac{i+1}{2}} + c_{\frac{m+k+i+1}{2}}, & i = 1, 3, \cdots, k-2; \\ c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{m+k+i+1}{2}}, & i = k, k+2, \\ & \quad \cdots, m-k-3; \\ c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{k-m+i+1}{2}}, & i = m-k-1, m-k+1, \\ & \quad \cdots, m-2. \end{cases} \tag{9}$$

Obviously, we can utilize the similar strategy to obtain the explicit expression of $S_2$. Let $\sum_{i=0}^{m-1} z'_i x^i$ denote the Montgomery squaring respect to $A_2 B_2$, then we have:

$$\begin{aligned} S_2 &= \sum_{i=0}^{m-1} z'_i x^i (1 + x) \bmod f(x) \\ &= \sum_{i=1, i \neq k}^{m-1} (z'_i + z'_{i-1}) x^i \\ &\quad + (z'_k + z'_{k-1} + z'_{m-1}) x^k + (z'_{m-1} + z'_0). \end{aligned} \tag{10}$$

The explicit formulae for the coefficients of $S_2$ are given in (11).

$$s_i = \begin{cases} d_{\frac{i}{2}} + d_{\frac{m+k+i}{2}} + d_{\frac{k+i-1}{2}}, & i = 0, 2, \cdots, k-1; \\ d_{\frac{m+k+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & i = k+1, k+3, \\ & \quad \cdots, m-k-2; \\ d_{\frac{k-m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & i = m-k, m-k+2, \\ & \quad \cdots, m-1; \\ d_{\frac{k+i}{2}} + d_{\frac{i-1}{2}} + d_{\frac{m+k+i-1}{2}}, & i = 1, 3, \cdots, k-2; \\ d_{\frac{k+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{m+k+i-1}{2}}, & i = k, k+2, \\ & \quad \cdots, m-k-1; \\ d_{\frac{k+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{k-m+i-1}{2}}, & i = m-k+1, m-k+3, \\ & \quad \cdots, m-2. \end{cases} \tag{11}$$

*Case 5:* In this case, it is easy to check that $S_1$ and $S_2$ have the same transformation as case 1 presented in (8) and (10), but the Montgomery

squaring formula is different. We have

$$r_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{m+k+i+1}{2}} + c_{\frac{k+i+1}{2}}, & i = 0, 2, \cdots, k-3; \\ c_0 + c_{\frac{k-1}{2}} + c_{\frac{m+2k}{2}} + c_k, & i = k-1; \\ c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{m+k+i+1}{2}}, & i = k+1, k+3, \\ & \cdots, m-k-3; \\ c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{k-m+i+1}{2}}, & i = m-k-1, m-k+1, \\ & \cdots, m-2; \\ c_{\frac{k-1}{2}} + c_{\frac{m+k-1}{2}} + c_0, & i = m-1; \\ c_{\frac{k+i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{i+1}{2}}, & i = 1, 3, \cdots, k-2; \\ c_{\frac{k+i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{m+k+i+1}{2}}, & i = k, k+2, \\ & \cdots, m-k-2; \\ c_{\frac{k+i}{2}} + c_{\frac{k-m+i}{2}} + c_{\frac{m+i+1}{2}}, & i = m-k, m-k+2, \\ & \cdots, m-3; \end{cases} \tag{12}$$

and

$$s_i = \begin{cases} d_{\frac{i}{2}} + d_{\frac{m+k+i-1}{2}} + d_{\frac{k+i-1}{2}}, & i = 0, 2, \cdots, k-1; \\ d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+k+i-1}{2}}, & i = k+1, k+3, \cdots, \\ & m-k-1; \\ d_{\frac{k-m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & i = m-k+1, m-k+3, \\ & \cdots, m-2; \\ d_{\frac{k+i}{2}} + d_{\frac{m+k+i}{2}} + d_{\frac{i-1}{2}}, & i = 1, 3, \cdots, k-2; \\ d_{\frac{k+i}{2}} + d_{\frac{m+k+i}{2}} + d_{\frac{m+i-1}{2}}, & i = k, k+2, \cdots, \\ & m-k-2; \\ d_{\frac{k+i}{2}} + d_{\frac{k-m+i}{2}} + d_{\frac{m+i-1}{2}}, & i = m-k, m-k+2, \\ & \cdots, m-1. \end{cases} \tag{13}$$

The explicit formulae about $S_1 + S_2$ of case 2-4 and case 6 can be found in the appendix A.

### 3.5 Computation of $S_3$

Since the computation of $C = A_1 + A_2$ and $D = B_1 + B_2$ require one extra $T_X$ gate delay, in order to keep pace with the computations of $S_1$ and $S_2$, we use a different computational strategy for $S_3$ that combines the polynomial multiplication with Montgomery squaring.

*Case 1:* Let $\sum_{i=0}^{\frac{m-1}{2}} u_i x^i = C$ and $\sum_{i=0}^{\frac{m-1}{2}} v_i x^i = D$, then we have

$$e_i = \begin{cases} \sum_{j=0}^{i} u_j v_{i-j}, & 0 \le i \le \frac{m-1}{2}, \\ \sum_{j=i-\frac{m-1}{2}}^{\frac{m-1}{2}} u_j v_{i-j}, & \frac{m+1}{2} \le i \le m-1. \end{cases} \tag{14}$$

and

$$t_i = \begin{cases} e_{\frac{i}{2}} + e_{\frac{m+k+i}{2}}, & i = 0, 2, \cdots, k-1; \\ e_{\frac{m+k+i}{2}}, & i = k+1, k+3, \cdots, m-k-2; \\ e_{\frac{k-m+i}{2}}, & i = m-k, m-k+2 \cdots, m-1; \\ e_{\frac{k+i}{2}}, & i = 1, 3, \cdots, k-2; \\ e_{\frac{k+i}{2}} + e_{\frac{m+i}{2}}, & i = k, k+2, \cdots, m-2. \end{cases} \tag{15}$$

By substituting (14) into (15), we can obtain the explicit expression of $t_i$s summarized in Table 2. Note that it only need $\frac{(m+1)^2}{4}$ AND gates to computed all the $u_i v_j$ for $i, j = 0, 1, \cdots, \frac{m-1}{2}$, here, we only present the required number of XOR gates.

*XOR Gates Reuse trick:* The computation of $t_i$ consists of multiplying $u_i$ with $v_j$ and adding up all these products using a binary XOR tree. In (15), we note that there exist certain overlapped terms among some $t_i$s, thus reusing the intermediated results in binary XOR trees could further reduce the number of required XOR gates.

For example, $t_0 = e_0 + e_{\frac{m+k}{2}}$ and $t_k = e_k + e_{\frac{m+k}{2}}$ contain the same part $e_{\frac{m+k}{2}}$. According to (14), it follows that $e_{\frac{m+k}{2}} = \sum_{j=\frac{k+1}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}-j}$ consisting of $\frac{m-k}{2}$ terms. If $\frac{m-k}{2}$ is an odd number, $t_0$ and $t_k$ are computed in following way:

$$\begin{aligned} t_0 =& \underline{[u_{\frac{k+1}{2}} v_{\frac{m-1}{2}} + u_{\frac{k+3}{2}} v_{\frac{m-3}{2}}]} + \underline{[u_{\frac{k+5}{2}} v_{\frac{m-5}{2}}} \\ & \underline{+ u_{\frac{k+7}{2}} v_{\frac{m-7}{2}}]} + \cdots + \underline{[u_{\frac{m-5}{2}} v_{\frac{k+5}{2}} + u_{\frac{m-3}{2}} v_{\frac{k+3}{2}}]} \\ & + [u_{\frac{m-1}{2}} v_{\frac{k+1}{2}} + u_0 v_0], \end{aligned}$$

$$\begin{aligned} t_k =& \underline{[u_{\frac{k+1}{2}} v_{\frac{m-1}{2}} + u_{\frac{k+3}{2}} v_{\frac{m-3}{2}}]} + \underline{[u_{\frac{k+5}{2}} v_{\frac{m-5}{2}}} \\ & \underline{+ u_{\frac{k+7}{2}} v_{\frac{m-7}{2}}]} + \cdots + \underline{[u_{\frac{m-5}{2}} v_{\frac{k+5}{2}} + u_{\frac{m-3}{2}} v_{\frac{k+3}{2}}]} \\ & + [u_{\frac{m-1}{2}} v_{\frac{k+1}{2}} + u_0 v_k] + [u_1 v_{k-1} + u_2 v_{k-2}] \\ & + \cdots + [u_{k-1} v_1 + u_k v_0]. \end{aligned} \tag{16}$$

In (16), the terms $u_i v_j$ correspond to the XOR tree nodes in depth 0. We then add the nodes pairwisely and repeat this step until adding up all those terms together. This procedure can be depicted in Fig. 1 and Fig. 2.

The black nodes in tree (a) and tree (b) represent the overlapping terms in $t_0$ and $t_k$. Due to parallelism, the additions in the brackets are performed simultaneously. Note that brackets

TABLE 2
The computation complexity of $t_i$ before optimization

| $i$ | $t_i$ | #XOR | $i$ | $t_i$ | #XOR |
|---|---|---|---|---|---|
| 0 | $u_0v_0+\sum_{j=\frac{k+1}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}-j}$ | $\frac{m-k}{2}$ | 1 | $\sum_{j=0}^{\frac{k+1}{2}} u_j v_{\frac{k+1}{2}-j}$ | $\frac{k+1}{2}$ |
| 2 | $u_0v_1+u_1v_0+\sum_{j=\frac{k+3}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}+1-j}$ | $\frac{m-k}{2}$ | 3 | $\sum_{j=0}^{\frac{k+3}{2}} u_j v_{\frac{k+1}{2}+1-j}$ | $\frac{k+3}{2}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $k-1$ | $\sum_{j=0}^{\frac{k-1}{2}} u_j v_{\frac{k-1}{2}-j}+\sum_{j=k}^{\frac{m-1}{2}} u_j v_{\frac{m-1}{2}+k-j}$ | $\frac{m-k}{2}$ | $k-2$ | $\sum_{j=0}^{k-1} u_j v_{k-1-j}$ | $k-1$ |
| $k+1$ | $\sum_{j=k+1}^{\frac{m-1}{2}} u_j v_{\frac{m-1}{2}+k+1-j}$ | $\frac{m-1}{2}-k$ | $k$ | $\sum_{j=0}^{k} u_j v_{k-j}+\sum_{j=\frac{k+1}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}-j}$ | $\frac{m+k}{2}$ |
| $k+3$ | $\sum_{j=k+2}^{\frac{m-1}{2}} u_j v_{\frac{m-1}{2}+k+2-j}$ | $\frac{m-3}{2}-k$ | $k+2$ | $\sum_{j=0}^{k+1} u_j v_{k+1-j}+\sum_{j=\frac{k+3}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}+1-j}$ | $\frac{m+k}{2}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $m-k-2$ | $u_{\frac{m-1}{2}} v_{\frac{m-1}{2}}$ | 0 | $m-k-1$ | $\sum_{j=0}^{\frac{m-1}{2}} u_j v_{\frac{m-1}{2}-j}+\sum_{j=\frac{m-k}{2}}^{\frac{m-1}{2}} u_j v_{m-\frac{k+1}{2}-j}$ | $\frac{m+k}{2}$ |
| $m-k$ | $u_0v_0$ | 0 | $m-k+1$ | $\sum_{j=1}^{\frac{m-1}{2}} u_j v_{\frac{m+1}{2}-j}+\sum_{j=\frac{m-k}{2}+1}^{\frac{m-1}{2}} u_j v_{m-\frac{k-1}{2}-j}$ | $\frac{m+k}{2}-2$ |
| $m-k+2$ | $u_0v_1+u_1v_0$ | 1 | $m-k+3$ | $\sum_{j=2}^{\frac{m-1}{2}} u_j v_{\frac{m+3}{2}-j}+\sum_{j=\frac{m-k}{2}+2}^{\frac{m-1}{2}} u_j v_{m-\frac{k-3}{2}-j}$ | $\frac{m+k}{2}-4$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $m-1$ | $\sum_{j=0}^{\frac{k-1}{2}} u_j v_{\frac{k-1}{2}-j}$ | $\frac{k-1}{2}$ | $m-2$ | $\sum_{j=\frac{k-1}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}-1-j}+u_{\frac{m-1}{2}} v_{\frac{m-1}{2}}$ | $\frac{m-k}{2}+1$ |

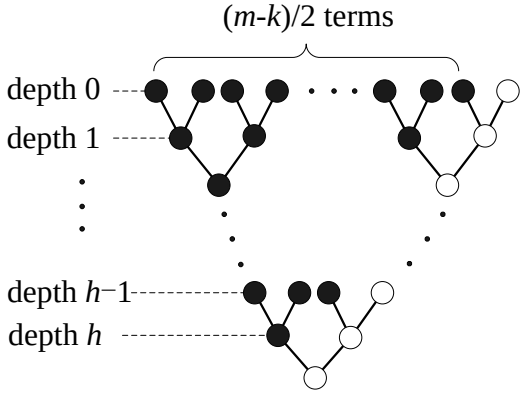


Fig. 1. The binary XOR tree (a) related to $t_0$

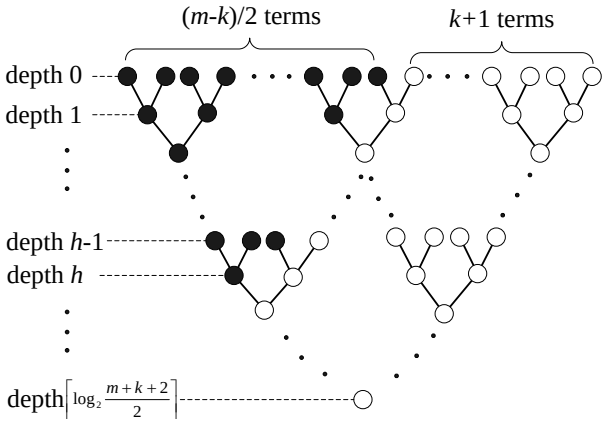


Fig. 2. The binary XOR tree (b) related to $t_k$

with underlines between the two expressions contain the same values, we only need to compute these values in tree (a) and reuse them in tree (b). Therefore, $\lfloor\frac{m-k}{4}\rfloor$ XOR gates will be saved in depth 0. Similarly, it follows that $\lfloor\frac{m-k}{8}\rfloor$ XOR gates can be saved at depth 1, $\lfloor\frac{m-k}{16}\rfloor$ XOR gates saved at depth 2, etc. Let $\lfloor\log_2(\frac{m-k}{2})\rfloor = h$, in depth $h-1$, there exist two nodes thus one XOR gates will be saved. Then the total number of saved XOR gates is

$$\left\lfloor\frac{m-k}{4}\right\rfloor+\left\lfloor\frac{m-k}{8}\right\rfloor+\cdots+\left\lfloor\frac{m-k}{2^{h+1}}\right\rfloor. \quad (17)$$

Here, note that $\lfloor\frac{m-k}{2^{h+1}}\rfloor = 1$. Moreover, above expression can be further simplified to

$$\frac{m-k}{2}-W\left(\frac{m-k}{2}\right),$$

according to certain proposition.[2]

In Table 3, we present the explicit number of XOR gates for all the $t_i$s overlapped with others when we apply this trick. The second column $j$ represents the index of the $t_j$ which overlaps with $t_i$. Consequently, we can obtain

2. The proposition and its proof are presented in appendix B.

TABLE 3
The computation complexity of certain $t_i$s after optimization

| $i$ | overlap with $j$ | #XOR | delay for binary tree |
|---|---|---|---|
| 0 | $m-k$ | $\frac{m-k}{2}-(1-W(1))$ | $\lceil\log_2\frac{m-k}{2}+1\rceil$ |
| 2 | $m-k+2$ | $\frac{m-k}{2}-(2-W(2))$ | $\lceil\log_2\frac{m-k}{2}+1\rceil$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $k-1$ | $m-1$ | $\frac{m-k}{2}-(\frac{k+1}{2}-W(\frac{k+1}{2}))$ | $\lceil\log_2\frac{m-k}{2}+1\rceil$ |
| $k$ | 0 | $\frac{m+k}{2}-(\frac{m-k}{2}-W(\frac{m-k}{2}))$ | $\lceil\log_2\frac{m+k}{2}+1\rceil$ |
| $k+2$ | 2 | $\frac{m+k}{2}-(\frac{m-k}{2}-1-W(\frac{m-k}{2}-1))$ | $\lceil\log_2\frac{m+k}{2}+1\rceil$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $m-k-1$ | $m-2k-1$ | $\frac{m+k}{2}-(\frac{k+1}{2}-W(\frac{k+1}{2}))$ | $\lceil\log_2\frac{m+k}{2}+1\rceil$ |
| $m-k+1$ | $m-2k+1$ | $\frac{m+k}{2}-2-(\frac{k-1}{2}-W(\frac{k-1}{2}))$ | $\lceil\log_2\frac{m+k}{2}\rceil$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $m-2$ | $m-k-2$ | $\frac{m-k}{2}+1-(1-W(1))$ | $\lceil\log_2\frac{m-k}{2}+1\rceil$ |

the complexity related to $S_3$ based on Table 2 and Table 3:

$$\begin{aligned}&\#\text{AND}: \frac{(m+1)^2}{4},\\ &\#\text{XOR}: \frac{(m-1)^2}{4}+\sum_{i=1}^{\frac{k+1}{2}}W(i)+\sum_{i=1}^{\frac{m-k}{2}}W(i),\\ &\text{Delay}: T_A+\lceil\log_2(m+k+2)\rceil T_X.\end{aligned}$$

*Case 5:* Actually, the computation strategies of other cases are nearly the same as that presented in case 1, we can use the *XOR gates reuse trick* to optimize its implementation. Note that the degrees of $C, D$ are at most $\frac{m}{2}-1$, let $\sum_{i=0}^{\frac{m}{2}-1}u_ix^i=C$ and $\sum_{i=0}^{\frac{m}{2}-1}v_ix^i=D$, then the coefficients of $CD$ and its Montgomery squaring are:

$$e_i=\begin{cases}\sum_{j=0}^{i}u_jv_{i-j}, & 0\le i\le\frac{m}{2}-1,\\ \sum_{j=i-\frac{m}{2}+1}^{\frac{m}{2}-1}u_jv_{i-j}, & \frac{m}{2}\le i\le m-2.\end{cases}\quad(18)$$

and

$$t_i=\begin{cases}e_{\frac{i}{2}}, & i=0,2,\cdots,k-1;\\ e_{\frac{m+i}{2}}, & i=k+1,k+3,\cdots,m-2;\\ e_{\frac{k+i}{2}}+e_{\frac{m+k+i}{2}}, & i=1,3,\cdots,m-k-2;\\ e_{\frac{k+i}{2}}+e_{\frac{k-m+i}{2}}, & i=m-k,m-k+2,\cdots,m-1.\end{cases}\quad(19)$$

We observed that when substitute Eq. (18) into Eq. (19), each $t_i$ is the sum of at most $\frac{m}{2}$ terms. Plus the delay of computing $u_iv_j$, the circuit delay for parallel implementation of Eq. (19) is at most $T_A+\lceil\log_2(\frac{m}{2})\rceil T_X$. The space and time complexity of $S_3$ here are given by:

$$\begin{aligned}&\#\text{AND}: \frac{m^2}{4},\\ &\#\text{XOR}: \frac{m^2-4m+4}{4}+\sum_{i=1}^{\frac{k+1}{2}}W(i)+\sum_{i=1}^{\frac{m-k-3}{2}}W(i),\\ &\text{Delay}: T_A+\lceil\log_2 m\rceil T_X.\end{aligned}$$

### 3.6 The Computation Sequence

Ultimately, we add $S_1$, $S_2$ and $S_3$ together to obtain the result. It is crucial to arrange the computation sequence properly to obtain the optical circuit delay. Particularly, note that the computation of $S_3$ requires at least one more $T_X$ gate delay than that of $A_1B_1$ and $A_2B_2$. During this extra XOR gate delay, we can perform one bitwise addition between $S_1, S_2$ in parallel.

*Case 1:* The computation sequence is arranged as follows:

$$\left\{\begin{array}{c|c}\underbrace{C,\ D}_{1T_X} & \underbrace{(CD)^2x^{-t}}_{T_A+\lceil\log_2\frac{m+k+2}{2}\rceil T_X}\\ \hline \underbrace{A_1B_1, A_2B_2}_{T_A+\lceil\log_2\frac{m-1}{2}\rceil T_X} & \underbrace{[S_1+S_2]}_{1T_X}\end{array}\right|\ \underbrace{[S_1+S_2]^*+S_3}_{2T_X}$$

where $[S_1+S_2]$ denotes the parallel bitwise additions in the square brackets indicated in

following equation.

$$
\begin{aligned}
&r_i + s_i = \\
&\begin{cases}
[c_{\frac{i}{2}} + c_{\frac{m+k+i}{2}}] + [c_{\frac{k+i+1}{2}} + d_{\frac{i}{2}}] & i = 0, 2, \cdots, k-1; \\
+[d_{\frac{m+k+i}{2}} + d_{\frac{k+i-1}{2}}], & \\
[c_{\frac{m+k+i}{2}} + c_{\frac{k+i+1}{2}}] + [c_{\frac{m+i+1}{2}} & i = k+1, k+3, \\
+d_{\frac{m+k+i}{2}}] + [d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}], & \cdots, m-k-2; \\
[c_{\frac{k-m+i}{2}} + c_{\frac{k+i+1}{2}}] + [c_{\frac{m+i+1}{2}} & i = m-k, m-k+2, \\
+d_{\frac{k-m+i}{2}}] + [d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}], & \cdots, m-3; \\
[c_{\frac{k-1}{2}} + c_{\frac{m+k}{2}}] + [d_{\frac{k-1}{2}} + d_{\frac{k+m-2}{2}}] & i = m-1; \\
[c_{\frac{k+i}{2}} + c_{\frac{i+1}{2}}] + [c_{\frac{m+k+i+1}{2}} + d_{\frac{k+i}{2}}] & \\
+[d_{\frac{i-1}{2}} + d_{\frac{m+k+i-1}{2}}], & i = 1, 3, \cdots, k-2; \\
[c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}}] + [c_{\frac{m+k+i+1}{2}} & i = k, k+2, \\
+d_{\frac{k+i}{2}}] + [d_{\frac{m+i}{2}} + d_{\frac{m+k+i-1}{2}}], & \cdots, m-k-3; \\
[c_{\frac{m-1}{2}} + c_{\frac{2m-k-1}{2}}] + [d_{\frac{m-1}{2}} & \\
+d_{\frac{2m-k-1}{2}}], & i = m-k-1; \\
[c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}}] + [c_{\frac{k-m+i+1}{2}} & i = m-k+1, \\
+d_{\frac{k+i}{2}}] + [d_{\frac{m+i}{2}} + d_{\frac{k-m+i-1}{2}}], & m-k+3, \cdots, m-2.
\end{cases}
\end{aligned}
\tag{20}
$$

After that, each coefficient of $S_1 + S_2$ consists of at most 3 terms, we denote these results as $[S_1 + S_2]^*$. It follows that there are at most 4 terms constituting to these coefficients of $[S_1 + S_2]^* + S_3$ which can be implemented in $2T_X$ in parallel. Therefore, the total complexity of the proposed multiplier of case 1 will be

$$
\begin{aligned}
&\#\text{AND} : \tfrac{3m^2+2m-1}{4}, \\
&\#\text{XOR} : \tfrac{3m^2}{4} + \tfrac{7m}{2} + \textstyle\sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i) - \tfrac{25}{4}, \\
&\text{Delay} : T_A + (2 + \lceil \log_2(m+k+2) \rceil) T_X.
\end{aligned}
\tag{21}
$$

*Case 5:* According to the coefficient expressions of $S_1$ in (12), the coefficient $r_{k-1}$ consists of four terms which would lead to one more XOR delay for $S_1 + S_2$.

However, we observed that $r_{k-1}$ contains the term $c_0$ that can be obtained with delay $T_A + T_X$. It is possible to "insert" $c_0$ to the binary XOR tree related to $t_{k-1}$ of $S_3$. [3] Particularly, the $k-1$-th coefficient of $S_3$, i.e., $t_i = e_{\frac{k-1}{2}}$ consists of $\frac{k+1}{2}$ terms. The delay of the binary tree related to $t_{k-1} + c_0$ is $\lceil \log_2(\frac{k+1}{2} + 1) \rceil T_X$. Note that here $k < \frac{m}{2}$, we have $\lceil \log_2(\frac{k+1}{2} + 1) \rceil < \lceil \log_2 \frac{m}{2} \rceil$, which indicates that $S_3 + c_0$ has the same delay with that of $S_3$. Therefore, the computation sequence of case 5 is given by:

$$
\left\{
\begin{array}{c|c}
\underbrace{C,\ D}_{1T_X} & \underbrace{(CD)^2 x^{-t} + \{c_0 x^{k-1}\}}_{T_A + \lceil \log_2 \frac{m}{2} \rceil T_X} \\
\hline
\underbrace{A_1 B_1, A_2 B_2}_{T_A + \lceil \log_2 \frac{m}{2} \rceil T_X} & \underbrace{[S_1 + S_2] - \{c_0 x^{k-1}\}}_{1T_X}
\end{array}
\right\}
\underbrace{[S_1 + S_2]^* + S_3}_{2T_X}
$$

The total complexity of the proposed multiplier of case 5 will be

$$
\begin{aligned}
&\#\text{AND} : \tfrac{3m^2}{4}, \\
&\#\text{XOR} : \tfrac{3m^2}{4} + 4m + \textstyle\sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k-3}{2}} W(i), \\
&\text{Delay} : T_A + (2 + \lceil \log_2 m \rceil) T_X.
\end{aligned}
\tag{22}
$$

The computation sequences of other cases are the same as those we presented in (21) and (22). Finally, we summarize the space and time complexity of these corresponding multipliers in the Table 4.

## 4 Comparison and Discussion

*Comments on space complexity*: We note that the expressions for number of XOR gates in Table 4 contain the sum of hamming weights related to certain integer, denoted by $\sum_{i=1}^{\sigma} W(i)$. This expression can be roughly written as $\frac{\sigma}{2} \log_2 \sigma$. [4] Therefore, the number of XOR gates required by our multiplier here is about: $\frac{3m^2}{4} + O(m \log_2 m)$.

*Comments on time complexity*: Denoted by $T(m, k)$ the time complexity of our multiplier, according to (21), (22) and Table 4, one can check that

$$
\begin{cases}
T(m,k) \le T_A + (3 + \lceil \log_2 m \rceil) T_X, & m \text{ odd}, \\
T(m,k) = T_A + (2 + \lceil \log_2 m \rceil) T_X, & m \text{ even}.
\end{cases}
$$

But for odd $m$, it is interesting only if $T(m,k) = T_A + (2 + \lceil \log_2 m \rceil) T_X$. This happens frequently when $m = 2^n + c$ where $c$ is smaller than $2^{n-1}$. For the range $100 < m \le 1023$, there exist 786 trinomials of odd degrees where $k \le \frac{m-1}{2}$. We have checked all these trinomials and found that 442 trinomials satisfies the previous requirement.

In Table 5, we provide the comparison between our proposal and several different bit-parallel multipliers of the same category. For

3. All the nodes of the binary XOR trees related to $t_i$ can be calculated with the same delay $T_A + T_X$.

4. Note that bit length of $\sigma$ is $\lceil \log_2 \sigma \rceil$, the average hamming weight of the number from 1 to $\sigma$ is about $\frac{\sigma}{2}$, which directly obtain the evaluation.

TABLE 4
Complexities of Montgomery multiplier for other cases

| Case | #AND | #XOR | Delay |
|---|---|---|---|
| $m$ odd, $k$ even $0 < k \leq \frac{m-1}{3}$ | $\frac{3m^2+2m-1}{4}$ | $\frac{3m^2}{4} + \frac{7m}{2} + \sum_{i=1}^{\frac{k}{2}} W(i) + \sum_{i=1}^{\frac{m-k-1}{2}} W(i) - \frac{25}{4}$ | $T_A + (2 + \lceil \log_2(2m - 3k - 2) \rceil) T_X$ |
| $m$ odd, $k$ even $\frac{m-1}{3} < k \leq \frac{m-3}{2}$ | | | $T_A + (2 + \lceil \log_2 3k \rceil) T_X$ |
| $m$ odd, $k$ even $k = \frac{m-1}{2}$ | $\frac{3m^2+2m-1}{4}$ | $\frac{3m^2}{4} + \frac{7m}{2} + \sum_{i=1}^{\frac{k}{2}} W(i) + \sum_{i=1}^{\frac{m-k-1}{2}} W(i) - \frac{25}{4}$ | $T_A + (2 + \lceil \log_2 3k \rceil) T_X$ |
| $m$ odd, $k$ odd $k = \frac{m-1}{2}$ | $\frac{3m^2+2m-1}{4}$ | $\frac{3m^2}{4} + \frac{7m}{2} + \sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i) - \frac{25}{4}$ | $T_A + (2 + \lceil \log_2(m + k + 2) \rceil) T_X$ |
| $m$ even, $k$ odd $k = \frac{m}{2}$ | $\frac{3m^2}{4}$ | $\frac{3m^2}{4} + 4m + \sum_{i=1}^{\frac{m+2}{4}} W(i) + \sum_{i=1}^{\frac{m-6}{4}} W(i) - 1$ | $T_A + (2 + \lceil \log_2 m \rceil) T_X$ |

TABLE 5
Comparison of Some Bit-Parallel Multipliers for Irreducible Trinomials

| Multiplier | # AND | # XOR | Time delay |
|---|---|---|---|
| $x^m + x + 1$ | | | |
| [6] [7] [8] | $m^2$ | $m^2 - 1$ | $T_A + (1 + \lceil \log_2 m \rceil) T_X$ |
| Wu [14] | $m^2$ | $m^2 - 1$ | $T_A + (2 + \lceil \log_2(m - 2) \rceil) T_X$ |
| [4] [18] [25] | $m^2$ | $m^2 - 1$ | $T_A + \lceil \log_2(2m - 1) \rceil T_X$ |
| Cho [21] | $m^2 - 1$ | $m^2 - 1$ | $T_A + (2 + \lceil \log_2(m - 4) \rceil) T_X$ |
| Proposed | $\frac{3m^2+2m-1}{4}$ | $\frac{3m^2}{4} + O(m \log_2 m)$ | $T_A + (2 + \lceil \log_2(m + 3) \rceil) T_X$ |
| $x^m + x^k + 1$ $(1 < k < \frac{m}{2})$ | | | |
| [6] [7] [8] [14] | $m^2$ | $m^2 - 1$ | $T_A + (2 + \lceil \log_2 m \rceil) T_X$ |
| Petra [25] | $m^2$ | $m^2 - 1$ | $T_A + (\lceil \log_2(2m + 2k - 3) \rceil) T_X$ |
| Fan [4] | $m^2$ | $m^2 - 1$ | $T_A + \lceil \log_2(2m - k - 1) \rceil T_X$ |
| Hariri [18] | $m^2$ | $m^2 - 1$ | $T_A + \lceil \log_2(2m - k - 1) \rceil T_X$ |
| Elia [23] | $\frac{3m^2+2m-1}{4}$ | $\frac{3m^2}{4} + 4m + k - \frac{23}{4}$ ($m$ odd) | $T_A + (3 + \lceil \log_2(m - 1) \rceil) T_X$ |
| | $\frac{3m^2}{4}$ | $\frac{3m^2}{4} + \frac{5m}{2} + k - 4$ ($m$ even) | |
| Li [24] | $\frac{m^2}{2} + (m - k)^2$ | $\frac{m^2}{2} + (m - k)^2 + 2k$ | $T_A + (2 + \lceil \log_2(m - 1) \rceil) T_X$ |
| Cho [21] | $m^2 - k^2$ | $m^2 + k - k^2 - 1 (1 < k < \frac{m}{3})$ | $\leq T_A + (2 + \lceil \log_2 m \rceil) T_X$ |
| | | $m^2 + 4k - k^2 - m - 1 (\frac{m}{3} \leq k < \frac{m-1}{2})$ | |
| | | $m^2 + 2k - k^2 (k = \frac{m-1}{2})$ | |
| Proposed | $\frac{3m^2+2m-1}{4}$ | $\frac{3m^2}{4} + O(m \log_2 m)$ ($m$ odd) | $\leq T_A + (3 + \lceil \log_2 m \rceil) T_X$ |
| | $\frac{3m^2}{4}$ | $\frac{3m^2}{4} + O(m \log_2 m)$ ($m$ even) | $T_A + (2 + \lceil \log_2 m \rceil) T_X$ |
| $x^m + x^{\frac{m}{2}} + 1$ | | | |
| [6] [7] [8] [14] | $m^2$ | $m^2 - \frac{m}{2}$ | $T_A + (1 + \lceil \log_2(m - 1) \rceil) T_X$ |
| [4] [18] [25] | $m^2$ | $m^2 - \frac{m}{2}$ | $T_A + \lceil \log_2 \frac{3m}{2} \rceil T_X$ |
| Shen [26] | $\frac{3m^2}{4}$ | $\frac{3m^2}{4} + m + 1$ | $T_A + (1 + \lceil \log_2(m - 1) \rceil) T_X$ |
| Shou [27] | $\frac{3m^2}{4}$ | $\frac{3m^2}{4} + m + 1$ | $T_A + (3 + \lceil \log_2(m - 1) \rceil) T_X$ |
| Cho [21] | $\frac{3m^2}{4}$ | $\frac{3m^2}{4} + m + 1$ | $T_A + (1 + \lceil \log_2(m - 2) \rceil) T_X$ |
| Proposed | $\frac{3m^2}{4}$ | $\frac{3m^2}{4} + O(m \log_2 m)$ | $T_A + (2 + \lceil \log_2 m \rceil) T_X$ |

good fields, our proposal obtains the same time complexity as the Mastrovito multiplier [6], [7], [8] and the Montgomery multiplier [14], but has a gain of roughly 25% space complexity. In addition, even we compare the time complexity presented in (21), (22) and Table 4 with those of the fastest multipliers [4], [18], it is found that our proposal is at most $2T_X$ slower (see Appendix C). Comparing with other Karatsuba-based multipliers [23], [24], it still maintains relatively low space complexity with the same or less time complexity.

In 2012, Cho et al. [21] proposed a variant Karatsuba algorithm and constructed a fast bit-parallel Karatsuba multiplier. They speeded up their multiplier at the cost of increasing the space complexity. Compared with their scheme, our proposal is slightly slower but has lower space complexity. In fact, there are 1405 irreducible trinomials with the degree $m$ between $[100, 1023]$ and $k \leq m/2$, we have checked all these trinomials and found that about 25% trinomials where our proposal has the same delay as [21], otherwise our scheme has at most $2T_X$ extra circuit delay. Particularly, among the five irreducible polynomials suggested for ECDSA (Elliptic Curve Digital Signature Algorithm) by NIST [28], there are two trinomials which are available for our architecture. We found our multipliers built on the two trinomials have the same time complexity with [21] and only one $T_X$ more than the fastest proposals [4] [18], while the space complexity is as good as the classic Karatsuba multiplier [23].

TABLE 6
Complexities of multiplier for trinomials recommended by NIST

| | # AND | # XOR | Time delay |
|---|---|---|---|
| $x^{233}+x^{74}+1$ | | | |
| [4] [18] | 54289 | 54288 | $T_A + 9T_X$ |
| [23] | 40833 | 41717 | $T_A + 11T_X$ |
| [21] | 48813 | 48886 | $T_A + 10T_X$ |
| This paper | 40833 | 41859 | $T_A + 10T_X$ |
| $x^{409}+x^{87}+1$ | | | |
| [4] [18] | 167281 | 167280 | $T_A + 10T_X$ |
| [23] | 125665 | 127178 | $T_A + 12T_X$ |
| [21] | 159712 | 159798 | $T_A + 11T_X$ |
| This paper | 125665 | 127566 | $T_A + 11T_X$ |

## 5 Conclusion

In this paper, we proposed a new bit-parallel Montgomery multiplier architecture for a class of irreducible trinomials. Our architecture is based on the extended PCHS algorithm and Montgomery squaring operations. It is argued that the space complexity of our proposal is about the same as those of the previous Karatsuba multipliers, while time complexity can match some Mastrovito multipliers and Montgomery multipliers, which are developed without any divide and conquer algorithm.

Since the PCHS algorithm usually relies on efficient squaring operations, the possible future work in this line should include Montgomery multiplier for pentanomials based on GPB squaring operations proposed by Xiong and Fan [22].

## Acknowledgements

The authors sincerely thank the anonymous reviewers for their insightful comments to improve this paper. The first author is supported by the National Natural Science Foundation of China (Grant No. 61402393).

## Appendix A

In the following, we give the explicit formulae about $S_1 + S_2$ in other cases.

Case 2: When both $m$ and $k$ are odd, $k = \frac{m-1}{2}$, we have:

$$r_i + s_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{k+i+1}{2}} + d_{\frac{i}{2}} & i = 0, 2, \cdots, k-1; \\ +d_{\frac{m+k+i}{2}} + d_{\frac{k+i-1}{2}}, & \\ c_{\frac{k-m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{m+i+1}{2}} & i = k+1, k+3, \\ +d_{\frac{k-m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & \cdots, m-3; \\ c_{\frac{k-1}{2}} + c_{\frac{m+k}{2}} + d_{\frac{k-1}{2}} & i = m-1; \\ +d_{\frac{k+m-2}{2}}, & \\ c_{\frac{k+i}{2}} + c_{\frac{i+1}{2}} + c_{\frac{m+k+i+1}{2}} & \\ +d_{\frac{k+i}{2}} + d_{\frac{i-1}{2}} + d_{\frac{m+k+i-1}{2}}, & i = 1, 3, \cdots, k-2; \\ c_k + c_{\frac{m+k}{2}} + d_k + d_{\frac{m+k}{2}}, & i = k; \\ c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{k-m+i+1}{2}} & i = k+2, k+4, \\ +d_{\frac{k+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{k-m+i-1}{2}}, & \cdots, m-2. \end{cases}$$

Case 3: When $m$ is odd and $k$ is even, $0 < k \leq \frac{m-3}{2}$, we have:

$$r_i + s_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{k+i}{2}} + c_{\frac{m+k+i+1}{2}} & i = 0, 2, \cdots, k-2; \\ +d_{\frac{i}{2}} + d_{\frac{k+i}{2}} + d_{\frac{m+k+i-1}{2}}, & \\ c_{\frac{k+i}{2}} + c_{\frac{m+k+i+1}{2}} + c_{\frac{m+i+1}{2}} & i = k, k+2, \\ +d_{\frac{k+i}{2}} + d_{\frac{m+k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & \cdots, m-k-3; \\ c_{\frac{m-1}{2}} + c_{m-\frac{k}{2}} + d_{\frac{m-1}{2}} + d_{m-\frac{k}{2}-1}, & i = m-k-1; \\ c_{\frac{k+i}{2}} + c_{\frac{k-m+i+1}{2}} + c_{\frac{m+i+1}{2}} & i = m-k+1, m-k+3, \\ +d_{\frac{k+i}{2}} + d_{\frac{k-m+i-1}{2}} + d_{\frac{m+i-1}{2}}, & \cdots, m-3; \\ c_{\frac{m+k+i}{2}} + c_{\frac{i+1}{2}} + c_{\frac{k+i+1}{2}} & \\ +d_{\frac{m+k+i}{2}} + d_{\frac{i-1}{2}} + d_{\frac{k+i-1}{2}}, & i = 1, 3, \cdots, k-1; \\ c_{\frac{m+k+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} & i = k+1, k+3, \\ +d_{\frac{m+k+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}}, & \cdots, m-k-2; \\ c_{\frac{k-m+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} & i = m-k, m-k+2, \\ +d_{\frac{k-m+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}}, & \cdots, m-2; \\ c_{\frac{m+k-1}{2}} + c_{\frac{k}{2}} + d_{\frac{m+k+1}{2}} + d_{\frac{k}{2}-1}, & i = m-1. \end{cases}$$

Case 4: When $m$ is odd and $k$ is even, $k = \frac{m-1}{2}$, we have:

$$r_i + s_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{k+i}{2}} + c_{\frac{m+k+i+1}{2}} & i = 0, 2, \cdots, k-2; \\ +d_{\frac{i}{2}} + d_{\frac{k+i}{2}} + d_{\frac{m+k+i-1}{2}}, & \\ c_k + c_{\frac{m+k+1}{2}} + d_k + d_{\frac{m+k-1}{2}} & i = k; \\ c_{\frac{k+i}{2}} + c_{\frac{k-m+i+1}{2}} + c_{\frac{m+i+1}{2}} & i = k+2, k+4, \\ +d_{\frac{k+i}{2}} + d_{\frac{k-m+i-1}{2}} + d_{\frac{m+i-1}{2}}, & \cdots, m-3; \\ c_{\frac{m+k-1}{2}} + c_{\frac{k}{2}} + d_{\frac{m+k-1}{2}-1} + d_{\frac{k}{2}-1}, & i = m-1; \\ c_{\frac{m+k+i}{2}} + c_{\frac{i+1}{2}} + c_{\frac{k+i+1}{2}} & \\ +d_{\frac{m+k+i}{2}} + d_{\frac{i-1}{2}} + d_{\frac{k+i-1}{2}}, & i = 1, 3, \cdots, k-1; \\ c_{\frac{k-m+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} & i = k+1, k+3, \\ +d_{\frac{k-m+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}}, & \cdots, m-2. \end{cases}$$

Case 6: When $m$ is even and $k$ is odd, $m = 2k$

$$r_i + s_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{m+k+i+1}{2}} + d_{\frac{i}{2}} & i = 0, 2, \cdots, \frac{m}{2}-3; \\ +d_{\frac{m+k+i-1}{2}} + d_{\frac{k+i-1}{2}} & \\ c_k + c_{\frac{k-1}{2}} + d_{k-1} + d_{\frac{k-1}{2}} & i = \frac{m}{2}-1; \\ c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{k-m+i+1}{2}} & i = k+1, k+3, \\ +d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{k-m+i-1}{2}}, & \cdots, m-2; \\ c_k + c_{\frac{m+k+1}{2}} + c_0 + d_k & i = \frac{m}{2}; \\ +d_0 + d_{\frac{k+m-1}{2}} & \\ c_{\frac{k+i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{i+1}{2}} & \\ +d_{\frac{k+i}{2}} + d_{\frac{k-m+i}{2}} + d_{\frac{m+i-1}{2}}, & i = 1, 3, \cdots, \frac{m}{2}-2; \\ c_{\frac{k+i}{2}} + c_{\frac{k-m+i}{2}} + c_{\frac{m+i+1}{2}} & i = \frac{m}{2}+2, k+4, \\ +d_{\frac{k+i}{2}} + d_{\frac{k-m+i}{2}} + d_{\frac{m+i-1}{2}}, & \cdots, m-3; \\ c_{\frac{k-1}{2}} + c_{\frac{m+k-1}{2}} + d_{\frac{m}{2}} + d_{\frac{k-1}{2}}, & i = m-1; \end{cases}$$

## APPENDIX B

**Proposition 1** *Let $W(i)$ be the hamming weight of an integer $i$, then $i$ can always be written as $i = 2^{n_1} + 2^{n_2} + \cdots + 2^{n_t}$ where $n_1 > n_2 > \cdots > n_t \geq 0$. Note that $\lfloor \log_2 i \rfloor = n_1$, then*

$$\left\lfloor \frac{i}{2} \right\rfloor + \left\lfloor \frac{i}{4} \right\rfloor + \cdots + \left\lfloor \frac{i}{2^{n_1}} \right\rfloor = i - W(i)$$

*Proof:* Firstly, it is clear that

$$\begin{aligned} \left\lfloor \frac{i}{2} \right\rfloor &= 2^{n_1-1} + 2^{n_2-1} + \cdots + 2^{n_t-1}, \\ \left\lfloor \frac{i}{4} \right\rfloor &= 2^{n_1-2} + 2^{n_2-2} + \cdots + 2^{n_t-2}, \\ &\vdots \\ \left\lfloor \frac{i}{2^{n_t}} \right\rfloor &= 2^{n_1-n_t} + 2^{n_2-n_t} + \cdots + 2^{n_{t-1}-n_t} + 2^0, \\ &\vdots \\ \left\lfloor \frac{i}{2^{n_1}} \right\rfloor &= 1. \end{aligned}$$

When we rearrange these terms of previous expressions and add them up, we have:

$$\begin{aligned} 2^{n_1-1} + 2^{n_1-2} + \cdots + 1 &= 2^{n_1} - 1, \\ 2^{n_2-1} + 2^{n_2-2} + \cdots + 1 &= 2^{n_2} - 1, \\ &\vdots \\ 2^{n_t-1} + 2^{n_t-2} + \cdots + 1 &= 2^{n_t} - 1. \end{aligned}$$

Obviously,

$$2^{n_1} - 1 + 2^{n_2} - 1 + \cdots + 2^{n_t} - 1 = i - W(i),$$

which conclude the proposition. □

## APPENDIX C

**Proposition 2** *The time complexity of our multiplier is at most $2T_X$ higher than those of Fan [4] and Hariri [18] scheme.*

*Proof:* According to Table 5, the time complexity of [4] and [18] are the same, which is equal to

$$T_A + \lceil \log_2(2m - k - 1) \rceil T_X,$$

if $0 < k \leq \frac{m-1}{2}$, or

$$T_A + \lceil \log_2(m + k) \rceil T_X,$$

if $k = \frac{m}{2}$. When we compare it with the formulae with respect to time delay presented in (21), (22) and Table 4, we have:

1) $m$ odd, $k$ odd, $1 \leq k \leq \frac{m-3}{2}$
$2m-k-1-(m+k+2) = m-2k-3 \geq 0$;

2) $m$ odd, $k$ odd, $k = \frac{m-1}{2}$,
$2m-k-1-(m+k+2) = -2$;

3) a) $m$ odd, $k$ even, $1 \leq k \leq \frac{m-1}{3}$,
$2m-k-1-(2m-3k-2) = 2k-1 > 0$;
b) $m$ odd, $k$ even, $\frac{m-1}{3} < k \leq \frac{m-3}{2}$,
$2m-k-1-3k = 2m-4k-1 > 0$;

4) $m$ odd, $k$ even, $k = \frac{m-1}{2}$,
$2m-k-1-3k = 2m-4k-1 = 1 > 0$;

5) $m$ even, $k$ odd, $m > 2k$,
$2m-k-1-m = m-k-1 > 0$;

6) $m$ even, $k$ odd, $m = 2k$,
$m+k-m = k > 0$.

Therefore, except case 2, all the formulae related to the time delay have at most 2 more $T_X$ than those of Fan [4] and Hariri [18] scheme.

In addition, in case 2, we note that the difference between $m+k+2$ and $2m-k-1$ is only 2. There is a high probability that the formula $\lceil \log_2(m+\frac{m-1}{2}+2) \rceil$ is equal to $\lceil \log_2(2m-\frac{m-1}{2}-1) \rceil$. Only if $m+\frac{m-1}{2}+2 = 2^\ell+1$ or $2^\ell+2$, the two formulae are unequal, where $\ell > 0$ is an integer. But we found that there is no such irreducible trinomial for $m \in [100, 2048]$ for cryptography interests.

Therefore, in case 2, our multiplier is also $2T_X$ slower than Fan [4] and Hariri [18] scheme, which conclude the proposition.

□

## REFERENCES


[1] A.J. Menezes, I.F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone, T. Yaghoobian, *Applications of Finite Fields*, Kluwer Academic, Norwell, Massachusetts, USA, 1993.

[2] I. Blake, G. Seroussi, N. Smart, *Elliptic Curves in Cryptography*, Lond. Math. Soc. Lect. Note Ser., vol. 265, Cambridge University Press, 1999.

[3] R. Lidl, H. Niederreiter, *Finite Fields*, Cambridge University Press, New York, NY, USA, 1996.

[4] H. Fan, M.A. Hasan, "Fast Bit Parallel-Shifted Polynomial Basis Multipliers in $GF(2^n)$," *IEEE Trans. Circuits and Systems I: Fundamental Theory and Applications*, vol. 53, no. 12, pp. 2606–2615, Dec. 2006.

[5] A. Cilardo, "Fast Parallel $GF(2^m)$ Polynomial Multiplication for All Degrees," *IEEE Trans. Computers*, vol. 62, no. 5, pp.929–943, May 2013.

[6] B. Sunar and Ç.K. Koç, "Mastrovito multiplier for all trinomials," *IEEE Trans. Computers*, vol. 48, no. 5, pp. 522–527, 1999.

[7] A. Halbutogullari and Ç.K. Koç, "Mastrovito multiplier for general irreducible polynomials," *IEEE Trans. Computers*, vol. 49, no. 5, pp. 503–518, May 2000.

[8] T. Zhang and K.K. Parhi, "Systematic design of original and modified mastrovito multipliers for general irreducible polynomials," *Computers, IEEE Transactions on*, vol. 50, no. 7, pp. 734–749, July 2001.

[9] Peter L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[10] Ç.K. Koç and T. Acar, "Montgomery multiplication in $GF(2^k)$," *Designs, Codes and Cryptography*, vol. 14, no. 1, pp. 57–69, 1998.

[11] J.C. Bajard, L. Imbert, and C. Negre, "Arithmetic operations in finite fields of medium prime characteristic using the lagrange representation," *IEEE Trans. Computers*, vol. 55, no. 9, pp. 1167–1177, 2006.

[12] C. Chiou, C. Lee, A. Deng, and J. Lin, "Concurrent Error Detection in Montgomery Multiplication over $GF(2^m)$," *IEICE Trans. Fundamentals of Electronics, Comm. and Computer Sciences*, vol. 89, no. 2, pp. 566-574, 2006.

[13] M. Morales-Sandoval, C. Feregrino-Uribe, P. Kitsos, and R. Cumplido, "Area/performance trade-off analysis of an FPGA digit-serial GF(2m) Montgomery multiplier based on LFSR," *Comput. Electr. Eng.* vol. 39, no. 2, pp. 542-549, February 2013.

[14] H. Wu, "Montgomery multiplier and squarer for a class of finite fields," *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 521–529, 2002.

[15] C.-Y. Lee, C.-C. Chen; E.-H. Lu, "Compact Bit-Parallel Systolic Montgomery Multiplication Over $GF(2^m)$ Generated by Trinomials," *TENCON 2006. 2006 IEEE Region 10 Conference*, pp. 1-4, November 2006.

[16] C.-Y. Lee, C.W. Chiou, J.-M. Lin, C.-C. Chang, "Scalable and systolic Montgomery multiplier over $GF(2^m)$ generated by trinomials," *IET Circuits, Devices & Systems*, vol. 1, no. 6, pp. 477-484, December 2007.

[17] C.-Y. Lee, J.-S. Horng, I.-C. Jou, E.-H. Lu, "Low-Complexity Bit-Parallel Systolic Montgomery Multipliers for Special Classes of $GF(2^m)$," *IEEE Trans. Comput.*, vol. 54, no. 9, pp. 1061-1070, September 2005.

[18] A. Hariri and A. Reyhani-Masoleh, "Bit-serial and bit-parallel Montgomery multiplication and squaring over $GF(2^m)$," *IEEE Trans. Computers*, vol. 58, no. 10, pp. 1332–1345, October 2009.

[19] S. Park, K, Chang, D. Hong, and C. Seo, "New efficient bit-parallel polynomial basis multiplier for special pentanomials," *Integration, the VLSI Journal*, vol. 47, no. 1, pp. 130 – 139, 2014.

[20] S. Park, "Explicit formulae of polynomial basis squarer for pentanomials using weakly dual basis Integration," *Integration, the VLSI Journal*, vol. 45, pp. 205–210, 2012.

[21] Y. Cho, N. Chang, C. Kim, Y. Park, and S. Hong, "New bit parallel multiplier with low space complexity for all irreducible trinomials over $GF(2^n)$," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 10, pp. 1903–1908, October 2012.

[22] X. Xiong, H. Fan, "$GF(2^n)$ bit-parallel squarer using generalised polynomial basis for new class of irreducible pentanomials," *Electronics Letters* , vol. 50, no. 9, pp. 655–657, April 2014.

[23] M. Elia, M. Leone, and C. Visentin, "Low complexity bit-parallel multipliers for $GF(2^m)$ with generator polynomial $x^m + x^k + 1$," *Electronic Letters*, vol. 35, no. 7, pp. 551–552, 1999.

[24] Y. Li, G. Chen, and J. Li, "Speedup of bit-parallel karatsuba multiplier in $GF(2^m)$ generated by trinomials," *Inf. Process. Lett.*, vo. 111, no. 8, pp. 390–394, March 2011.

[25] N. Petra, D. De Caro, and A.G.M. Strollo, "A novel architecture for galois fields $GF(2^m)$ multipliers based on mastrovito scheme," *IEEE Trans. Computers*, vol. 56, no. 11, pp. 1470–1483, November 2007.

[26] H. Shen, Y. Jin. 2008, "Low complexity bit parallel multiplier for $GF(2^m)$ generated by equally-spaced trinomials," *Inf. Process. Lett.*, vol. 107, no. 6, pp. 211-215, August 2008.

[27] G. Shou, Z. Mao, Y. Hu, Z. Guo, Z. Qian, "Low complexity architecture of bit parallel multipliers for $GF(2^m)$," *Electronics Letters*, vol. 46, no. 19, pp. 1326-1327, September 2010.

[28] Recommended Elliptic Curves for Federal Government Use, http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf, July, 1999.