A low complexity bit-parallel Montgomery multiplier based on squaring for trinomials

Yin Li,Yiyang Chen

Abstract—In this paper, we present a new bit-parallel Montgomery multiplier for $GF(2^m)$ generated with an irreducible trinomial. A newly proposed divide-and-conquer approach is applied to simplify the polynomial multiplication while the Montgomery squaring is induced to optimize the modular reduction. Meanwhile, this design effectively exploits the overlapped elements in squaring and reduction operations to reduce the space complexity. As a result, the proposed multiplier has about 25% reduced space complexity compared with previous multipliers, with a slight increase of time complexity. Among the five binary fields recommended by NIST for the ECDSA (Elliptic Curve Digital Signature Algorithm), there exist two fields, i.e., $GF(2^{409})$, $GF(2^{233})$, defined by trinomials. For these two fields, we show that our proposal outperforms the previous best known results if the space and time complexities 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 makes 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 algorithm 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 among these literatures is selected as x^m. In [14], Wu has proposed a new bit-parallel Montgomery multiplier for irreducible trinomials using 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. Hariri and Reyhani-Masoleh [15] have further improved Wu's proposal. Besides new recommendation of the Montgomery factor, fast bit-serial and bit-parallel multiplier architectures for irreducible trinomials and pentanomials are also given. It is argued that their scheme matches the best known result reported in the literatures.

Hariri and Reyhani-Masolehs architecture is really fast, but the space complexity of their multiplier costs about $O(m^2)$ circuit gates. Our work is devoted to reduce the space complexity of Montgomery multiplier so as to perform the scalar multiplication of Elliptic Curve

- *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.*

Cryptography (ECC) in some area constraint devices. We achieve this goal by inducing a new divide-and-conquer algorithm which is recently proposed by Park et al. [16]. Other than the frequently used Karatsuba algorithm which partitions polynomials into two halves, Park et al. approach, referred as PCHS algorithm, splits a polynomial $A = \sum_{i=0}^{m-1} a_i x^i, a_i \in \mathbb{F}_2$ according to the parity of x's exponent and utilized the squaring operation. This algorithm is originally applied in the multiplier for irreducible pentanomials, in this paper we will show that such approach is also applicable for the multiplier for trinomials. Combining with the PCHS algorithm and Montgomery squaring, a new bit-parallel Montgomery multiplication architecture is proposed. Explicit formulae about complexity analysis are given. As a result, the space complexity of our proposal is about 25% less than any other Montgomery or Mastrovito multipliers for irreducible trinomials, with the time complexity is slightly higher than the best known results.

The remainder of this paper is organized as follows: In Section 2, we briefly review the Montgomery multiplication over $GF(2^m)$ and the PCHS algorithm. Then a new bit-parallel Montgomery multiplier based on PCHS algorithm is described 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 some concepts about the Montgomery multiplication over $GF(2^m)$ and the PCHS algorithm.

2.1 Montgomery multiplication in $GF(2^m)$

Let $f(x)$ be an irreducible polynomials which defines the finite field $GF(2^m)$ and $r(x)$ be a fixed polynomial with $\deg(r(x)) \leq m$. It is clear that $gcd(f(x), r(x)) = 1$, therefore, there exist $\tilde{f}(x)$ and $\tilde{r}(x)$ such that

$$r(x)\tilde{r}(x) + f(x)\tilde{f}(x) = 1. \qquad (1)$$

Obviously, $\tilde{r}(x) = r^{-1}(x)$ is the inverse of $r(x)$ modulo $f(x)$. Given two field elements $a(x), b(x) \in GF(2^m)$, the Montgomery multiplication over $GF(2^m)$ is given by:

$$c(x) = a(x)b(x)r^{-1}(x) \bmod f(x).$$

The fixed polynomial $r(x)$ is denoted as Montgomery factor and can be chosen according to $f(x)$ to construct efficient multiplier architecture. In [10], $r(x)$ is chosen as x^m, because the modular operation and a final division regarding to x^m are very simple. The former can ignore the terms whose powers of x are great than or equal to m and the latter requires m-bit shift of its operand. Hariri and Reyhani-Masoleh [15] have proved that $r(x) = x^{m-1}$ is more suitable for efficient bit-serial Montgomery multiplier. When $f(x)$ is an irreducible trinomial or pentanomial, the bit-parallel Montgomery multiplier architecture is investigated in [15], [14]. In this case, $r(x)$ is chosen as x^k or x^{k-1} which can simplify the modular reduction. Therefore, this choice is very suitable for bit-parallel architecture that can result in short circuit delay.

2.2 The PCHS algorithm

Let $A = \sum_{i=0}^{m-1} \alpha x^i$ and $B = \sum_{i=0}^{m-1} \beta x^i$ be two elements in $GF(2^m)$ using polynomial basis. Assume that m is an odd integer. We can partition A, B into

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

respectively, where $A_1 = \sum_{i=0}^{(m-1)/2} \alpha_{2i} x^i$, $A_2 = \sum_{i=0}^{(m-3)/2} \alpha_{2i+1} x^i$, $B_1 = \sum_{i=1}^{(m-1)/2} \beta_{2i-1} x^i$, and $B_2 = \sum_{i=0}^{(m-1)/2} \beta_{2i} x^i$. Then we have

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

Analogous with Karatsuba algorithm, Equation (2) modified the m-term polynomial multiplication into three $\frac{m+1}{2}$ (or $\frac{m-1}{2}$)-term polynomial multiplications at the cost of three extra additions. Please note that this formula also utilize squaring operations, thus the PCHS algorithm should be combined with efficient squaring operation together in order to construct efficient multiplication. In the following section, we will describe a new bit-parallel multiplier based on Park et al. algorithm and Montgomery squaring operation for irreducible trinomials.

3 New field multiplication using Montgomery squaring operation

In this section, we present a new Montgomery multiplication formula for irreducible trinomials using the PCHS algorithm. Suppose that the finite 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 $1 \leq k \leq m/2$ and m is odd, because there always exist irreducible trinomial $f(x) = x^m + x^{m-k} + 1$ by the reciprocal property [3] and this type of trinomial is practically used in cryptography [21].

Assume that $A, B \in GF(2^m)$ are two arbitrary elements in polynomial basis representation:

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

where $a_i,\ b_i \in \mathbb{F}_2$. We use x^u $(1 \leq u < m)$ as the Montgomery factor, consider the Montgomery multiplication over $GF(2^m)$:

$$C = A \cdot B \cdot x^{-u} \bmod f(x). \tag{3}$$

According to the PCHS approach, we split A, B into two parts as follows:

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

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 the Montgomery multiplication can be rewritten as

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

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

Form the above expression, it is clear that the Montgomery multiplication is transferred into the Montgomery squaring in terms of A_1B_1, A_2B_2, CD. Please note that the choice of the Montgomery factor x^u could highly influence these squaring operations. Hariri and Reyhani-Masoleh [15] have proposed the optimal x^u which can be summarized in following lemma:

Lemma 1 *Let $f(x) = x^m + x^k + 1$ be an irreducible trinomial over $\mathbb{F}_2$ and x be the root of $f(x)$. Then, the Montgomery factor x^u is obtained from following in order to construct the simplest modular reduction.*

$$u = \begin{cases} 1, & k = 1, \\ k \text{ or } k-1, & k > 1. \end{cases} \tag{4}$$

Proof See section 5 in [15]. □

Hence, in this paper we use the same Montgomery factor as Wu's argument, where $x^u = x^k$. Note that the degrees of A_1B_1, A_2B_2 and CD are at most $m-1$. From now on, the following notions are used:

$$\begin{aligned} &A_1B_1 = \sum_{i=0}^{m-1} c_ix^i, A_2B_2 = \sum_{i=0}^{m-1} d_ix^i, CD = \sum_{i=0}^{m-1} e_ix^i, \\ &S_1 = (A_1B_1)^2x^{-k}(1+x^{-1}) \bmod f(x) = \sum_{i=0}^{m-1} s_ix^i, \\ &S_2 = (A_2B_2)^2x^{-k}(1+x) \bmod f(x) = \sum_{i=0}^{m-1} t_ix^i, \\ &S_3 = (CD)^2x^{-k} \bmod f(x) = \sum_{i=0}^{m-1} r_ix^i. \end{aligned}$$

Here, notice that $c_0 = 0$ and $d_{m-1} = 0$. Then the Montgomery multiplication can be expressed as

$$ABx^{-k} \bmod f(x) = S_1 + S_2 + S_3.$$

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

3.1 The complexities of A_1B_1, A_2B_2

We first 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_is of $A_1B_1 = \sum_{i=0}^{m-1} c_i x^i$ are given by

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

As an example, the gate count and time delay for implementation of each c_i in (5) are presented in Table 1. It is easy to see that the computation of c_is for $0 \leq i < m$ cost $\frac{m^2-1}{4}$ AND and $\frac{m^2-4m+3}{4}$ XOR gates with path delay $T_A + \lceil \log_2\left(\frac{m-1}{2}\right) \rceil T_X$.

Similarly, we can easily obtain the space and time complexity related to A_2B_2 in following formulae:

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

3.2 The computation of S_1, S_2

According to previous description, we know that the key computation step of S_1, S_2 is the Montgomery squaring operation related to A_1B_1 and A_2B_2. The Montgomery squaring has been fully studied through the explicit formulation [14] which are varied according to the range of m and k. Hence, we consider four cases:

- m is odd, k is odd, $1 \leq k \leq \frac{m-3}{2}$,
- m is odd, k is odd, $k = \frac{m-1}{2}$,
- m is odd, k is even, $1 \leq k \leq \frac{m-3}{2}$,
- m is odd, k is even, $k = \frac{m-1}{2}$.

The four cases as above correspond to different squaring formulae, hence the result of S_1, S_2 is expressed differently. For example, suppose that m and k satisfy case 1, denote $\sum_{i=0}^{m-1} z_i x^i$ as the Montgomery squaring $(A_1B_1)^2 x^{-k}$, we have following expression:

$$z_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{m+k+i}{2}}, & i = 0, 2, \cdots, k-1; \\ c_{\frac{m+k+i}{2}}, & i = k+1, k+3, \cdots, m-k-2; \\ c_{\frac{k-m+i}{2}}, & i = m-k, m-k+2 \cdots, m-1; \\ c_{\frac{k+i}{2}}, & i = 1, 3, \cdots, k-2; \\ c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}}, & i = k, k+2, \cdots, m-2. \end{cases} \tag{6}$$

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

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

Then we substitute z_i with the above expressions in (6). Note that $c_0 = 0$, the coefficients of S_1 are given by:

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

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_2B_2, then we have:

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

The explicit formulae for the coefficients of S_2 are given in (8). It follows that each coefficient of $S_1 + S_2$ consists of at most six terms which can be implemented with the gate delay at

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	$\frac{m^2-1}{4}$	$\frac{m^2-4m+3}{4}$	$T_A + \lceil \log_2 (\frac{m-1}{2}) \rceil T_X$

most $3T_X$ in parallel. The detailed computation sequence will be investigated in Section 3.4.

$$t_i = \begin{cases} d_{\frac{i}{2}} + d_{\frac{m+k+i}{2}} + d_{\frac{k+i-1}{2}}, & i = 0, 2, \cdots, k-1; \\ d_{\frac{m+k+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & i = k+1, k+3, \\ & \quad \cdots, m-k-2; \\ d_{\frac{k-m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & i = m-k, m-k+2, \\ & \quad \cdots, m-1; \\ d_{\frac{k+i}{2}} + d_{\frac{i-1}{2}} + d_{\frac{m+k+i-1}{2}}, & i = 1, 3, \cdots, k-2; \\ d_{\frac{k+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{m+k+i-1}{2}}, & i = k, k+2, \\ & \quad \cdots, m-k-1; \\ d_{\frac{k+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{k-m+i-1}{2}}, & i = m-k+1, m-k+3, \\ & \quad \cdots, m-2; \end{cases} \tag{8}$$

For simplicity, we do not present detailed computational procedures of S_1+S_2 for other cases. The detailed results of other cases can be found in the appendix.

3.3 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 computation of S_1+S_2, we use different computational strategy which combines the polynomial multiplication with Montgomery squaring.

Let $\sum_{i=0}^{\frac{m-1}{2}} u_i = A_1+A_2$ and $\sum_{i=0}^{\frac{m-1}{2}} v_i = B_1+B_2$, then

$$e_i = \begin{cases} \sum_{j=0}^{i} u_j v_{i-j} & 0 \le i \le \frac{m-1}{2}, \\ \sum_{j=i-\frac{m-1}{2}}^{\frac{m-1}{2}} u_j v_{i-j} & \frac{m+1}{2} \le i \le m-1. \end{cases} \tag{9}$$

and

$$r_i = \begin{cases} e_{\frac{i}{2}} + e_{\frac{m+k+i}{2}}, & i = 0, 2, \cdots, k-1; \\ e_{\frac{m+k+i}{2}}, & i = k+1, k+3, \cdots, m-k-2; \\ e_{\frac{k-m+i}{2}}, & i = m-k, m-k+2 \cdots, m-1; \\ e_{\frac{k+i}{2}}, & i = 1, 3, \cdots, k-2; \\ e_{\frac{k+i}{2}} + e_{\frac{m+i}{2}}, & i = k, k+2, \cdots, m-2. \end{cases} \tag{10}$$

By substituting (9) into (10), we can obtain the explicit expression of r_is. Note that it only need $\frac{(m+1)^2}{4}$ AND gates to computed all the $u_i v_j$ for $i, j = 0, 1, \cdots, \frac{m-1}{2}$, we only present the number of XOR gates required by each r_i which are summarized in Table 2.

Generally, the computation of r_i presented in Table 2 consists of multiplying u_i with v_j and adding up all these products using a binary XOR tree. However, note that there exist certain overlapped terms among some r_is, we will show that the number of required XOR gates can be further reduced by reusing the intermediated results in these binary trees.

For example, one can check that $r_0 = e_0 + e_{\frac{m+k}{2}}$ contains a same part $e_{\frac{m+k}{2}}$ with $r_k = e_k + e_{\frac{m+k}{2}}$. According to (9), it follows that $e_{\frac{m+k}{2}} = \sum_{j=\frac{k+1}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}-j}$ which consists of $\frac{m-k}{2}$ terms. Provided that $\frac{m-k}{2}$ is odd and then r_0, r_k are calculated using a binary XOR tree. The detailed computation of r_0 and r_k are performed

TABLE 2
The computation complexity of r_i.

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

as follows:

$$\begin{aligned} r_0 =&\big[u_{\frac{k+1}{2}}v_{\frac{m-1}{2}} + u_{\frac{k+3}{2}}v_{\frac{m-3}{2}}\big] + \big[u_{\frac{k+5}{2}}v_{\frac{m-5}{2}} \\ &+ u_{\frac{k+7}{2}}v_{\frac{m-7}{2}}\big] + \cdots + \big[u_{\frac{m-5}{2}}v_{\frac{k+5}{2}} + u_{\frac{m-3}{2}}v_{\frac{k+3}{2}}\big] \\ &+ \big[u_{\frac{m-1}{2}}v_{\frac{k+1}{2}} + u_0v_0\big], \end{aligned}$$

$$\begin{aligned} r_k =&\big[u_{\frac{k+1}{2}}v_{\frac{m-1}{2}} + u_{\frac{k+3}{2}}v_{\frac{m-3}{2}}\big] + \big[u_{\frac{k+5}{2}}v_{\frac{m-5}{2}} \\ &+ u_{\frac{k+7}{2}}v_{\frac{m-7}{2}}\big] + \cdots + \big[u_{\frac{m-5}{2}}v_{\frac{k+5}{2}} + u_{\frac{m-3}{2}}v_{\frac{k+3}{2}}\big] \\ &+ \big[u_{\frac{m-1}{2}}v_{\frac{k+1}{2}} + u_0v_k\big] + \big[u_1v_{k-1} + u_2v_{k-2}\big] \\ &+ \cdots + \big[u_{k-1}v_1 + u_kv_0\big]. \end{aligned} \tag{11}$$

In (11), the terms u_iv_j represent the XOR tree nodes in depth 0 and the additions in the brackets are computed simultaneously. The results of the brackets correspond to the XOR tree nodes in depth 1. We then follow the same line to add the nodes pairwisely and repeat those steps until adding up all those terms together. This procedure can be depicted in Fig. 1 and Fig. 2.

Due to parallelism, each layer of the XOR trees related to r_0 and r_k cost a gate delay T_X. Meanwhile, note that the first $\lfloor\frac{m-k}{4}\rfloor$ brackets of the two expressions in (11) have the same results, hence we only need to compute these

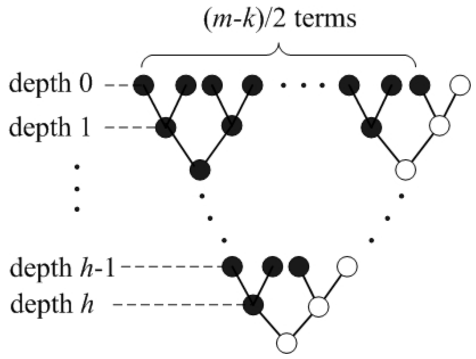

Fig. 1. The binary XOR tree (a) related to r_0

results in tree (a) and $\lfloor\frac{m-k}{4}\rfloor$ XOR gates will be saved in tree (b). Similarly, it follows that $\lfloor\frac{m-k}{8}\rfloor$ XOR gates can be saved at depth 2, $\lfloor\frac{m-k}{2^{t+1}}\rfloor$ XOR gates saved at depth t, etc. The black nodes in tree (a) and tree (b) represent the overlapping terms in r_0 and r_k. Denote $\lfloor\log_2(\frac{m-k}{2})\rfloor = h$, then from depth 0 to depth $h-1$, the number of saving XOR gates related to the computation of r_k is

$$\left\lfloor\frac{m-k}{4}\right\rfloor + \left\lfloor\frac{m-k}{8}\right\rfloor + \cdots + \left\lfloor\frac{m-k}{2^{h+1}}\right\rfloor. \tag{12}$$

Moreover, (12) can be simplified even further according to following proposition.

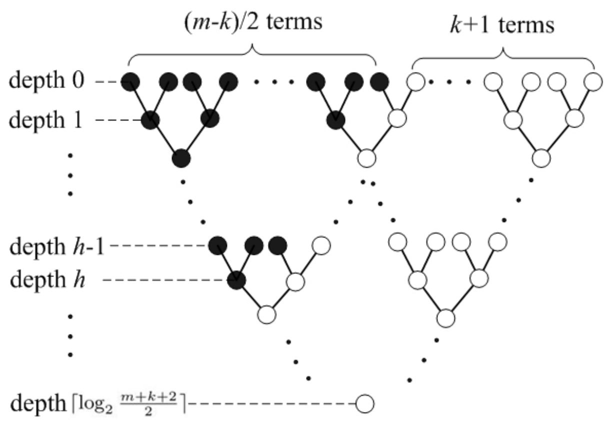

Fig. 2. The binary XOR tree (b) related to r_k

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:

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

Obviously,

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

which conclude the proposition. □

Based on proposition 1, (12) can be rewritten as

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

Consequently, the explicit number of XOR gates for overlapped r_is are given in Table 3.

In Table 3, the second column j represent the index of the r_j which overlap with r_i. Notice that it requires 1 T_X to calculate $C = A_1 + A_2$ and $D = B_1 + B_2$ in parallel. As a result, we can directly obtain the space the time complexity related to S_3 based on Table 2 and Table 3:

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

3.4 The computation sequence

Ultimately, we add up S_1, S_2 and S_3 to obtain the final result. 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 time, we can perform one addition between the coefficients of $S_1 + S_2$ in parallel, namely,

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

Due to parallelism, terms in the square brackets of (13) are computed concurrently. Consequently, the computation sequence of the whole

TABLE 3
The computation complexity of r_i 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	$\frac{m+k}{2}-(\frac{m-k}{2}-1-W(\frac{m-k}{2}-1))$	$\lceil\log_2\frac{m+k}{2}+1\rceil$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m-k-1$	$m-2k-1$	$\frac{m+k}{2}-(\frac{k+1}{2}-W(\frac{k+1}{2}))$	$\lceil\log_2\frac{m+k}{2}+1\rceil$
$m-k+1$	$m-2k+1$	$\frac{m+k}{2}-2-(\frac{k-1}{2}-W(\frac{k-1}{2}))$	$\lceil\log_2\frac{m+k}{2}\rceil$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m-2$	$m-k-2$	$\frac{m-k}{2}+1-(1-W(1))$	$\lceil\log_2\frac{m-k}{2}+1\rceil$

algorithm is arranged as follows:

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

The expression $[S_1+S_2]$ means computing one addition in the square brackets of (13) simultaneously. After the parallel addition in (13), each coefficient of S_1+S_2 consists of at most 3 terms. It follows that there are at most 4 terms constituting to these coefficients plus r_i which can be performed in 2 T_X in parallel. Therefore, the total complexity of the proposed multiplier will be

$$\begin{aligned} &\#\text{AND}: \tfrac{3m^2+2m-1}{4}, \\ &\#\text{XOR}: \tfrac{3m^2}{4}+\tfrac{7m}{2}+\textstyle\sum_{i=1}^{\frac{k+1}{2}}W(i)+\sum_{i=1}^{\frac{m-k}{2}}W(i)-\tfrac{25}{4}, \\ &\text{Delay}: T_A+(2+\lceil\log_2(m+k+2)\rceil)\,T_X. \end{aligned} \qquad (14)$$

For simplicity, we do not present the detailed computation for the other cases of m and k. One can utilize the same strategy as above to develop corresponding multipliers. The space and time complexity of the multipliers with respect to other cases are summarized in the Table 4.

4 Comparison and Discussion

Notice that the time complexity of our multiplier can be recognized as a function of m and k which is denoted as $T(m,k)$. According to (14) and Table 4, one can check that

$$T(m,k)\le T_A+(3+\lceil\log_2 m\rceil)T_X.$$

In table 5, we provide the comparison between our proposal and several different bit-parallel multipliers of the same category. On the one hand, it is obvious that our scheme requires at most $3T_X$ than the fastest multiplier but saves about 25% logic gates. On the other hand, it costs less circuit gates compared with other Karatsuba-based multiplier with the same or less time complexity.

Particularly, there exist a number of m and k satisfying

$$T(m,k)=T_A+(2+\lceil\log_2 m\rceil)T_X. \qquad (15)$$

For the range $100<m\le 1023$, there exist 786 trinomials with odd degrees. We have searched all these trinomials and found that about 56% trinomials where the time complexity is equal to $T_A+(2+\lceil\log_2 m\rceil)T_X$. Specially, among the five irreducible polynomials suggested for ECDSA (Elliptic Curve Digital Signature Algorithm) by NIST [21], the two trinomials, namely, $x^{409}+x^{87}+1$ and $x^{233}+x^{74}+1$, are all satisfying the (15). At this time, the time complexity of our proposal the same as some Mastrovito multiplier [6] [7] [8] or Montgomery multiplier [14], but requires roughly 25% fewer logic gates.

TABLE 4
Complexities of Montgomery multiplier for other cases

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

TABLE 5
Comparison of Some bit-parallel Multipliers for $x^m + x^k + 1 (1 < k \leq \frac{m-1}{2})$

Multiplier	# AND	# XOR	Time delay
[6] [7] [8] [14]	m^2	$m^2 - 1$	$T_A + (2 + \lceil \log_2 m \rceil) T_X$
Elia [18]	$\frac{3m^2+2m-1}{4}$	$\frac{3m^2}{4} + 4m + k - \frac{23}{4}$	$T_A + (3 + \lceil \log_2(m-1) \rceil) T_X$
Fan [4]	m^2	$m^2 - 1$	$T_A + \lceil \log_2(2m - k - 1) \rceil T_X$
Hariri [15]	m^2	$m^2 - 1$	$T_A + \lceil \log_2(2m - k - 1) \rceil T_X$
Li [19]	$\frac{m^2}{2} + (m-k)^2$	$\frac{m^2}{2} + (m-k)^2 + 2k$	$T_A + (2 + \lceil \log_2(m-1) \rceil) T_X$
Petra [20]	m^2	$m^2 - 1$	$T_A + (\lceil \log_2(2m + 2k - 3) \rceil) T_X$
Cho [17]	$m^2 - k^2$	$m^2 + k - k^2 - 1 (1 < k < \frac{m}{3})$	$\leq T_A + (2 + \lceil \log_2 m \rceil) T_X$
		$m^2 + 4k - k^2 - m - 1 (\frac{m}{3} \leq k < \frac{m-1}{2})$	
		$m^2 + 2k - k^2 (k = \frac{m-1}{2})$	
Proposed	$\frac{3m^2+2m-1}{4}$	$\frac{3m^2}{4} + O(m)$	$\leq T_A + (3 + \lceil \log_2 m \rceil) T_X$

In [17], Cho et al. proposed a variant Karatsuba algorithm and constructing an efficient Karatsuba based bit-parallel multiplier. However, theirs approach speed up the Karatsuba multiplier at the cost of increasing its space complexity. Compared with their scheme, our proposal requires even lower space complexity while maintains nearly the same time complexity. In table 6, we give two small examples to illustrate the efficiency of our multiplier. To facilitate description, we only compared our proposal with three typical results presented in [15], [18] and [17]. It is obvious that our multiplier outperforms others if the space and time complexities are both considered. This merit enables our multiplier to meet performance requirements under area constraint, such as smartcard, RFID tags, etc, but maintain considerable implementation efficiency as well.

TABLE 6
Complexities of multiplier for trinomials recommended by NIST

	# AND	# XOR	Time delay
$x^{233} + x^{74} + 1$			
[15]	54289	54288	$T_A + 9T_X$
[18]	40833	41717	$T_A + 11T_X$
[17]	48813	48886	$T_A + 10T_X$
This paper	40833	41859	$T_A + 10T_X$
$x^{409} + x^{87} + 1$			
[15]	167281	167280	$T_A + 10T_X$
[18]	125665	127178	$T_A + 12T_X$
[17]	159712	159798	$T_A + 11T_X$
This paper	125665	127566	$T_A + 11T_X$

5 Conclusion

In this paper, a new bit-parallel Montgomery multiplier architecture based on squaring operations is proposed. We show that the PCHS algorithm is not only applicable for pentanomials, but also applicable for trinomials. Explicit formulae of the Montgomery multipliers for the family of trinomials are also induced. The theoretical complexity analysis shows that our multipliers have the significant lower space complexity compared with other best known multipliers for these types of trinomials, but maintains a relatively low time delay. Our work is especially interesting for ECDSA as its efficiency over the two finite field proposed by NIST.

Appendix

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

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

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

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

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

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

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

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, Jul 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] 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.

[16] 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.

[17] 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, Oct 2012.

[18] 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.

[19] 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.

[20] 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*, vo. 56, no. 11, pp. 1470–1483, Nov 2007.

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