Efficient Algorithms and Architectures for Field Multiplication Using Gaussian Normal Bases

Arash Reyhani-Masoleh, *Member*, *IEEE*

Abstract—Recently, implementations of normal basis multiplication over the extended binary field $GF(2^m)$ have received considerable attention. A class of low complexity normal bases called Gaussian normal bases has been included in a number of standards, such as IEEE [1] and NIST [2] for an elliptic curve digital signature algorithm. The multiplication algorithms presented there are slow in software since they rely on bit-wise inner product operations. In this paper, we present two vector-level software algorithms which essentially eliminate such bit-wise operations for Gaussian normal bases. Our analysis and timing results show that the software implementation of the proposed algorithm is faster than previously reported normal basis multiplication algorithms. The proposed algorithm is also more memory efficient compared with its look-up table-based counterpart. Moreover, two new digit-level multiplier architectures are proposed and it is shown that they outperform the existing normal basis multiplier structures. As compared with similar digit-level normal basis multipliers, the proposed multiplier with serial output requires the fewest number of XOR gates and the one with parallel output is the fastest multiplier.

Index Terms—Finite field multiplication, normal basis, Gaussian normal basis, software algorithms, ECDSA.

1 Introduction

Arithmetic operations in the finite field $GF(2^m)$ are extensively used in elliptic curve cryptographic protocols and discrete-log-based cryptosystems [1]. The binary field $GF(2^m)$ is a set of 2^m elements, where each of these elements can be represented by a bit string of length m. The field can also be considered as an m-dimensional vector space with m linearly independent elements over $GF(2)$ known as a basis. There are two common bases, polynomial basis and normal basis, to represent field elements of $GF(2^m)$ [1]. Field element representations and arithmetic operations (except addition/subtraction) using these two bases are quite different. In normal basis representation, squaring of a field element is done by a cyclic shift of its coordinates and it is free in hardware. In the literature, a number of hardware structures for $GF(2^m)$ multiplication using normal bases have been proposed. For example, one can see [3], [4], [5], [6], [7] for bit-level, [8], [9], [10] for digit-level, and [11], [12], [13] for parallel normal basis multiplier architectures. For software implementation, multiplication using polynomial bases is considered simpler than that using normal bases. However, in some situations, such as those where compatibility with other bases and/or efficient squaring is important, one may need to implement normal basis multiplication in software [14], [15], [16], [17], [18].

A Gaussian normal basis (GNB) is a special class of normal basis (NB) which has received considerable attention for efficient implementation of field multiplication [19]. The GNBs have been included in a number of standards, such as IEEE [1] and NIST (National Institute of Standards and Technology) [2] for the elliptic curve digital signature algorithm (ECDSA). It is well-known that, for $GF(2^m)$, a normal basis exists for every positive integer m [20]. Also, a GNB exists for every positive integer m that is not divisible by eight [19]. A GNB for $GF(2^m)$ is determined by an integer T and is referred to as type T Gaussian normal basis. When there is more than one GNB for a given m, those with smaller values of T yield efficient implementation of field multiplication. This is because the complexity of a type T GNB multiplier is proportional to T [19]. A table of GNBs for $GF(2^m)$ for $2 \leq m \leq 1,000$ is given in [1].

In this paper, we consider GNBs for $GF(2^m)$, where m is an odd number. This implies that the type T of such a GNB for $GF(2^m)$ is an even integer. These classes of GNBs are important for cryptographic applications and include the five $GF(2^m)$ fields recommended by NIST, i.e., $m \in \{163, 233, 283, 409, 571\}$, for ECDSA [2]. In this paper, two efficient algorithms for $GF(2^m)$ multiplication using Gaussian normal bases are proposed. Moreover, two new digit-level multiplier architectures are proposed. For software and hardware implementations, they are compared with the best algorithms and structures available in the open literature. It is shown that the proposed GNB multiplication algorithms and structures outperform their counterparts available in the open literature. It is noted that this paper is an extended version of [21].

The organization of this paper is as follows: In Section 2, bit-level multiplication algorithms presented in IEEE [1] and NIST [2] as well as a vector-level multiplication algorithm proposed by Ning and Yin [14], [15] are briefly reviewed. Then, in Section 3, we modify the Ning-Yin software algorithm for multiplication using GNBs. Another algorithm which is more efficient than Ning-Yin's is proposed in Section 4. Also, the implementation results of these algorithms for the five binary field recommended by NIST for ECDSA are given in this section. The multiplication formulations are modified

• *The author is with the Department of Electrical and Computer Engineering, University of Western Ontario, London, Ontario, Canada N6A 5B9. E-mail: areyhani@uwo.ca.*

Manuscript received 16 May 2005; revised 3 Sept. 2005; accepted 7 Sept. 2005; published online 22 Nov. 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0160-0505.

for digit-level architectures and associated structures with serial and parallel outputs are proposed in Sections 5 and 6, respectively. Finally, conclusions are made in Section 7.

2 Multiplication Algorithms

In this section, we briefly review multiplication algorithms using an arbitrary normal basis as well as a GNB.

2.1 Multiplication Using an Arbitrary Normal Basis

A normal basis of $GF(2^m)$ over $GF(2)$ is a set $N = \{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ whose elements are linearly independent. In $GF(2^m)$, any element $A = (a_0, a_1, \cdots, a_{m-1})$ can be represented as a linear combination of the elements in N, i.e., $A = \sum_{i=0}^{m-1} a_i \beta^{2^i} = a_0\beta + a_1\beta^2 + \cdots + a_{m-1}\beta^{2^{m-1}}$, whose coefficient $a_i \in GF(2)$, $0 \le i \le m-1$, denotes the ith coordinate of A. Let us denote a field element $\mu_{i,j} = \beta^{2^i + 2^j} \in GF(2^m)$, $0 \le i, j \le m-1$, which is represented with respect to N as $\mu_{i,j} = \sum_{l=0}^{m-1} \mu_{i,j}^{(l)} \beta^{2^l}$. Let $\underline{a} = [a_0, a_1, \cdots, a_{m-1}]$ and $\underline{b} = [b_0, b_1, \cdots, b_{m-1}]$ be the row vectors which correspond to the field elements $A = (a_0, a_1, \cdots, a_{m-1})$ and $B = (b_0, b_1, \cdots, b_{m-1})$, respectively, and $C \in GF(2^m)$ denote their product, i.e.,

$$C = (c_0, c_1, \cdots, c_{m-1}) = AB = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \mu_{i,j}.$$

Then, the lth coordinate of C can be written as

$$c_l = \underline{a}\mathbf{M}^{(l)}\underline{b}^{tr}, \; 0 \le l \le m-1, \tag{1}$$

where $\mathbf{M}^{(l)} = [\mu_{i,j}^{(l)}]_{i,j=0}^{m-1}$, $\mu_{i,j}^{(l)} \in GF(2)$, $0 \le i, j \le m-1$, and $\underline{b}^{tr}$ denotes the matrix transpose of row vector $\underline{b}$. In (1), $\mathbf{M}^{(l)}$ is obtained from the l-fold right and down circular shifts of the *multiplication matrix* $\mathbf{M} = \mathbf{M}^{(0)}$. Thus, the entries of $\mathbf{M}^{(l)}$, i.e., $\mu_{i,j}^{(l)}$, can be obtained from the entries of $\mathbf{M}$, i.e., $M(i,j)$. As

$$\mu_{i,j}^{(l)} = M(i-l, j-l), \; 0 \le i, j, l \le m-1. \tag{2}$$

To compute the multiplication matrix $\mathbf{M}$ for an arbitrary NB, one can see [1]. Also, it is shown in [22] that the nonzero entries of $\mathbf{M}$ for an arbitrary NB can be found from the set $\Phi = \{(m-1, m-1)\} \cup \Phi_0 \cup \Phi_0'$, where

$$\begin{aligned}&\Phi_0 = \{(-\omega_{j,k} \bmod m, \; j - \omega_{j,k} \bmod m) : \\ &1 \le j \le \tfrac{m-1}{2}, \; 1 \le k \le h_j\}, \\ &\Phi_0' = \{(j - \omega_{j,k} \bmod m, \; -\omega_{j,k} \bmod m) : \\ &1 \le j \le \tfrac{m-1}{2}, \; 1 \le k \le h_j\},\end{aligned}$$

and $\mu_{0,j} = \beta^{1+2^j} = \sum_{k=1}^{h_j} \beta^{2^{\omega_{j,k}}}$. Thus,

$$M(i,j) = \begin{cases} 1, & \text{if } (i,j) \in \Phi \\ 0, & \text{otherwise.} \end{cases}$$

Having found the multiplication matrix $\mathbf{M}$ (and, hence, c_0), one can use the following algorithm to determine the remaining coordinates of C [1].

Algorithm 1. [1] (Multiplication using an arbitrary NB)

```
Input:
$A = (a_0, a_1, \cdots, a_{m-1}), \; B = (b_0, b_1, \cdots, b_{m-1}) \in GF(2^m)$,
and $\mathbf{M}$
Output: $C = (c_0, c_1, \cdots, c_{m-1}) = AB$
1.   Initialize $X := A$ and $Y := B$
2.   For $k = 0$ to $m-1$ {
3.     Compute: $c_k = \underline{x}\mathbf{M}\underline{y}^{tr}$
4.     $X := X \ll 1$ and $Y := Y \ll 1$}
5.   Output $C = (c_0, c_1, \cdots, c_{m-1})$.
```

In this algorithm, $X \ll 1 = (x_1, \cdots, x_{m-1}, x_0)$ denotes a left cyclic shift of the coordinates of $X = (x_0, x_1, \cdots, x_{m-1})$. Algorithm 1 is indeed an algorithmic version of the architecture of the bit-serial NB multiplier proposed by Massey and Omura in [3]. In that architecture, the number of 1s in the multiplication matrix $\mathbf{M}$ is a good measure of the number of logic gates needed for hardware implementation. This measure is referred to as the *complexity* of the normal basis and is denoted as C_N.

Unlike hardware implementation, the software implementation of Algorithm 1 is not very efficient. This is because, in each iteration of the *for* loop of the algorithm, it generates only one coordinate of the product C and requires one matrix-by-vector multiplication, i.e., $\mathbf{M}\underline{y}^{tr}$, and one vector-by-vector multiplication, i.e., $\underline{x}(\mathbf{M}\underline{y}^{tr})$. Let w denote the data path width of the processor. Then, the coordinates of the field elements and each column of $\mathbf{M}$ can be divided into $\lceil \frac{m}{w} \rceil$ words. Thus, for the purpose of storage, the multiplication matrix will require $\lceil \frac{m}{w} \rceil m$ words in total.

In [14], [15], Ning and Yin proposed software algorithms to use the full data path of the processor on which their algorithms are implemented. The idea behind the algorithm is to precompute and store the different cyclic shifts of A and B into a look-up table (i.e., memory). Then, instead of using cyclic shifts (as shown in Step 4 of Algorithm 1 above), the shifted values are read from the memory. This is shown below.

Algorithm 2. [14], [15] (Ning-Yin multiplication for arbitrary NBs)

```
Input:
$A = (a_0, a_1, \cdots, a_{m-1}), \; B = (b_0, b_1, \cdots, b_{m-1}) \in GF(2^m)$,
and $\mathbf{M}$
Output: $C = (c_0, c_1, \cdots, c_{m-1}) = AB$
1.   Precompute arrays for $A_i$ and $B_i$
2.   Initialize $C := 0$
3.   For $i = 0$ to $m-1$ {
4.     Initialize $S := 0$
5.     For $j = 0$ to $m-1$ {
6.       If $M(i,j) = 1$ then $S := S \oplus B_j$}
7.     $C := C \oplus (A_i \odot S)$
8.   Output $C = (c_0, c_1, \cdots, c_{m-1})$.
```

In this algorithm $M(i,j)$ represents the (i,j)th entry of the multiplication matrix $\mathbf{M}$. Also, $X \odot Y = (x_0y_0, x_1y_1, \cdots, x_{m-1}y_{m-1})$ and $X \oplus Y = (x_0 + y_0, x_1 + y_1, \cdots, x_{m-1} + y_{m-1})$ denote the bit-wise AND and the bit-wise XOR (field addition) operations between coordinates of $X = (x_0, x_1, \cdots, x_{m-1})$ and $Y = (y_0, y_1, \cdots, y_{m-1})$, respectively. In Step 1 of Algorithm 2, each unit of w bits of the cyclic shifts of A (and, similarly, B) are stored into two arrays as follows:

$$A[i] = (a_i, \; \cdots, \; a_{i+w-1 \bmod m}), \; 0 \le i \le m-1. \tag{3}$$

Thus, the i-fold left cyclic shifts of A can be obtained by reading the following units from the array as

$A_i = (A[i],\ A[i+w],\ \cdots,\ A[i+(\lceil \frac{m}{w} \rceil - 1)w])$, where the additions inside the square brackets are reduced modulo m.

2.2 Conventional GNB Multiplication

It was shown in [23] that, for any normal basis of $GF(2^m)$, the number of 1s in the multiplication matrix is always greater than or equal to $2m-1$, i.e., $C_N \geq 2m-1$. In order to have an efficient and simple implementation, one needs to choose a normal basis such that C_N is as small as possible. The best choice for C_N is obviously $2m-1$ and the corresponding NB is called an optimal normal basis (ONB). Two types of ONBs, i.e., type I and type II, were constructed by Mullin et al. [23]. However, such ONBs do not exist for all m. As an extension of optimal normal bases, Gaussian normal bases have been constructed by Ash et al. in [19]. Like ONBs, GNBs are special classes of normal bases. Although GNBs are generally less efficient than ONBs, when an ONB does not exist, GNB is considered to be the best alternative.

Definition 1. *A type T GNB for $GF(2^m)$ exists if and only if $p = Tm+1$ is prime and $\gcd(\frac{Tm}{k},\ m) = 1$, where k is the multiplicative order of 2 modulo p [24].*

It is noted that, when $T = 1$ and 2, we have type 1 and type 2 GNBs which are the same as type I and type II ONBs, respectively. It is also noted that, for $GF(2^m)$ when m is not a multiple of eight and no ONB exists, at least one GNB exists.

Instead of using $\mathbf{M}$ as an input in Algorithm 1, the multiplication algorithm presented in [1], [2] for GNB requires the explicit formula for the first coordinate of the product C, i.e.,

$$c_0 = \sum_{k=1}^{p-2} a_{F(k+1)} b_{F(p-k)}. \tag{4}$$

In (4), the sequence $F(1),\ F(2),\ \cdots,\ F(p-1)$ needs to be precomputed using

$$F(2^i u^j \mod p) = i,\ 0 \leq i \leq m-1,\ 0 \leq j < T, \tag{5}$$

where u is an integer of order $T \mod p$ and $p = Tm+1$. It is noted that each term in (4) corresponds to a nonzero entry of $\mathbf{M}$. Thus, one can easily obtain the multiplication matrix $\mathbf{M}$ for GNBs from (4).

In the following two sections, we discuss algorithms for efficient software implementation of Gaussian normal basis multiplication. Unlike the algorithms presented in the IEEE and NIST standards [1], [2], which use bit-wise operations, the proposed algorithms use vectors of bits. This enables us to use the full data path of the processor on which the software is executed.

3 Modified Ning-Yin Multiplication Algorithm

In this section, we modify Algorithm 2 for GNBs. This modification eliminates the inner *for* loop of Algorithm 2 and increases the speed of the algorithm. We first introduce the following lemmas which are subsequently used in our formulation of the modified multiplication algorithm (i.e., Algorithm 3).

Lemma 1 [25]. *For a self-dual normal basis*[1] *over $GF(2^m)$, when m is odd, the entries of $\mathbf{M}^{(m-1)}$ have the following properties:*

1. $\mu^{(m-1)}_{i,m-1} = \mu^{(m-1)}_{m-1,i} = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise.} \end{cases}$
2. $\mu^{(m-1)}_{i,j} = \mu^{(m-1)}_{(m-1+i-j),(m-j-2)} = \mu^{(m-1)}_{(j-i-1),(m-i-2)}$, *for $i < j$ and $0 \leq i, j \leq m-1$.*

Lemma 2 [16]. *For type T GNBs, where T is an even integer $F(k) = F(p-k),\ 1 \leq k \leq p-1$, where $F(k)$ is defined in (5).*

3.1 Formulation

It is well-known that the multiplication matrix is symmetric, i.e., $\mathbf{M} = \mathbf{M}^{tr}$. It is also known that all diagonal entries of $\mathbf{M}$ are zero except the last one, i.e.,

$$M(i,i) = \begin{cases} 1, & \text{if } i = m-1 \\ 0, & \text{otherwise.} \end{cases}$$

Now, we state the following lemma from [21] to present certain properties of $\mathbf{M}$ that the modified Ning-Yin algorithm relies on.

Lemma 3. *For type T Gaussian normal bases over $GF(2^m)$, where m is an odd integer (hence, T is even), the multiplication matrix $\mathbf{M}$ has the following properties:*

1. *There is only one nonzero entry in row 0 (or column 0), i.e.,*

$$M(0,j) = M(j,0) = \begin{cases} 1, & if\ j = 1 \\ 0, & otherwise. \end{cases}$$

2. *For other rows (or columns), the number of 1s in each row (or column) of $\mathbf{M}$ is even and less than or equal to T.*

Proof. Using (2) with $l = j = m-1$ into part 1 of Lemma 1, one can see that the first property exists for all self-dual normal bases. Also, it is proven in [26] that all GNBs of even type are self-dual normal basis. Thus, the proof of the first part is complete.

In order to prove the second part, first we use the above-mentioned property of $\mathbf{M}$ and Lemma 2 to simplify (4) as follows:

$$\begin{aligned} c_0 &= a_0 b_1 + \sum_{k=2}^{p-2} a_{F(k)} b_{F(k+1)} \\ &= a_0 b_1 + \sum_{i=1}^{m-1} a_i \left(\sum_{F(k)=i} b_{F(k+1)} \right), \text{ for } 2 \leq k \leq p-2, \end{aligned} \tag{6}$$

where

$$k = 2^i u^j \mod p,\ 1 \leq i \leq m-1,\ 0 \leq j < T. \tag{7}$$

Since u is an integer of order $T \mod p$ and, using (7), one can find that, for a given value of i, $1 \leq i \leq m-1$,

$$F(k) = i \tag{8}$$

1. A self-dual normal basis is a basis whose dual basis is itself.

has at most T unique solutions for k, $2 \leq k \leq p-2$, say $k_1, k_2, \cdots, k_T$. For each i in (6), if $F(k_u+1) \neq F(k_v+1)$, $u \neq v$, $1 \leq u, v \leq T$, then row i has exactly T number of 1s in the multiplication matrix. Otherwise, the number of 1s will be an even integer and it will be less than T. This is because T is even and, in (6), when the terms labeled as $b_{F(k+1)}$ are summed for $F(k) = i$, any two of these terms with the same subscript value would cancel each other.□

Remark 1. For $T = 2$, all rows of $\mathbf{M}$ have two nonzero entries except for row 0.

Using Lemma 3 and by counting the number of ones in the multiplication matrix, we can obtain the following:

Corollary 1. *The complexity of type T GNB over $GF(2^m)$ is $C_N \leq Tm - T + 1$, where m is an odd integer.*

Remark 2. It has been proven in [19] that $C_N \leq Tm - 1$. Thus, the new upper bound in Corollary 1 is a tighter upper bound for all even integers $T \geq 2$.

Remark 3. For $T = 4$, it has been shown in [19] that $C_N = 4m - 7$. This implies that only two rows of $\mathbf{M}$ should have two nonzero entries.

In order to efficiently implement GNB multiplication in software, it is better to store the indices of nonzero columns of $\mathbf{M}$ instead of storing the whole $\mathbf{M}$. Since the location of the single 1 of row 0 of $\mathbf{M}$ is fixed, we need to store those in rows 1 up to $m-1$. This eliminates the *if* condition in Step 6 of Algorithm 2. In an attempt to modify Algorithm 2 (more specifically, Steps 5 to 7 of the algorithm), let us denote an $(m-1) \times T$ matrix $\mathbf{R}$ whose entry $R(i,j)$, $0 \leq R(i,j) \leq m-1$, $1 \leq i \leq m-1$, $1 \leq j \leq T$, contains the column indices of 1s in row i of $\mathbf{M}$. If the number of 1s in row i of $\mathbf{M}$ is T, then all $R(i,j)$, $1 \leq j \leq T$, contain an integer in $[0, m-1]$. Otherwise, we initialize the remaining entries of $\mathbf{R}$, whose number is even, with a constant value, say 0.

Then, one can write c_0 from (6) as

$$c_0 = a_0 b_1 + \sum_{i=1}^{m-1} a_i \left(\sum_{j=1}^{T} b_{R(i,j)} \right). \tag{9}$$

This is because (9) has an even number of $b_{R(i,j)}$ redundant terms for a given i, $1 \leq i \leq m-1$. Since their associated entries of $R(i,j)$ are the same, say 0, the modulo 2 addition of such terms will be zero.

Using (9), one can obtain c_l by adding l modulo m to all indices in (9), i.e.,

$$c_l = a_l b_{l+1 \mathrm{mod} m} + \sum_{i=1}^{m-1} a_{l+i \bmod m} \left(\sum_{j=1}^{T} b_{l+R(i,j) \bmod m} \right),$$
$$0 \leq l \leq m-1.$$

Thus, we can state the following [21]:

Theorem 1. *Let C be the product of A and B in $GF(2^m)$. Then,*

$$C = (A \odot B_1) \oplus \sum_{i=1}^{m-1} A_i \odot (B_{R(i,1)} \oplus B_{R(i,2)} \oplus \cdots \oplus B_{R(i,T)}), \tag{10}$$

where $X \odot Y = (x_0 y_0,\ x_1 y_1,\ \cdots,\ x_{m-1} y_{m-1})$ and $X \oplus Y = (x_0 + y_0,\ x_1 + y_1,\ \cdots,\ x_{m-1} + y_{m-1})$ denote the bit-wise AND and XOR operations between coordinates of $X = (x_0,\ x_1,\ \cdots,\ x_{m-1})$ and $Y = (y_0,\ y_1,\ \cdots,\ y_{m-1})$, respectively, and $X_i = (x_i,\ x_{i+1},\ \cdots,\ x_{i-1})$ is the i-fold left cyclic shift of X.

3.2 Modified Algorithm

Based on Theorem 1, we can present the following algorithm for efficient software implementation of type T GNB multiplication for $GF(2^m)$. As seen earlier, Algorithm 2 has the second level of loop with m iterations, i.e., Steps 5-7. However, the following algorithm does not have such a loop and this helps to increase the speed of its execution when implemented in software. For the smallest nonzero positive even integer (i.e., $T = 2$), we have type 2 GNBs which are identical to type II ONBs and, for this particular value of T, the following algorithm is similar to the type II ONB multiplication algorithm presented in [14].

Algorithm 3. (Modified Ning-Yin multiplication for m odd)

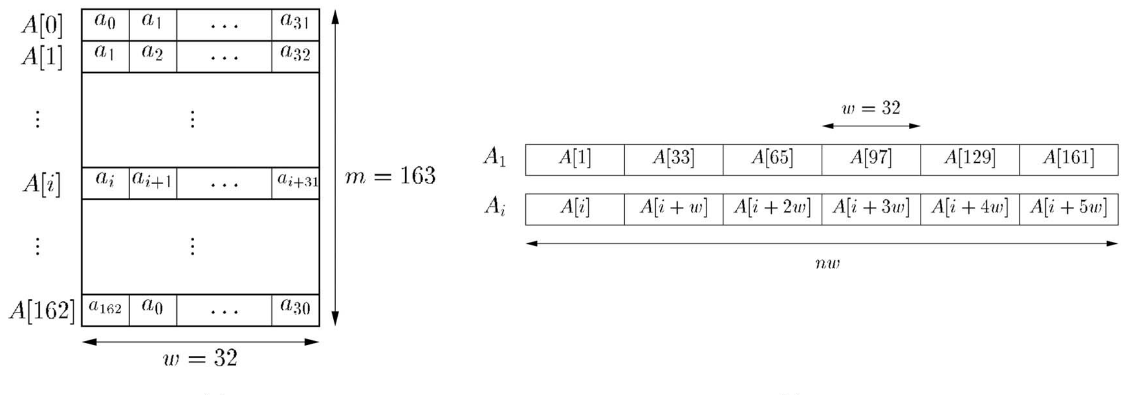
```
Input:
$A = (a_0,\ a_1,\ \cdots,\ a_{m-1}),\ B = (b_0,\ b_1,\ \cdots, b_{m-1}) \in GF(2^m)$,
and $m-1 \times T$ matrix $\mathbf{R}$
Output: $C = (c_0,\ c_1,\ \cdots,\ c_{m-1}) = AB$
1.    Precompute arrays for $A_i$ and $B_i$
2.    Initialize $C := A \odot B_1$
3.    For $i = 1$ to $m-1$ {
4.        $C := C \oplus (A_i \odot (B_{R(i,1)} \oplus B_{R(i,2)} \oplus \cdots \oplus B_{R(i,T)}))$}
5.    Output $C = (c_0,\ c_1,\ \cdots,\ c_{m-1})$.
```

In the above algorithm, three different operations, namely, bit-wise AND ($\odot$), bit-wise XOR ($\oplus$), and cyclic shifts, are used. As seen in Step 2 and Step 4 of Algorithm 3, the number of two former operations are m and $T(m-1)$, respectively. In a programming language like C, bit-wise AND and bit-wise XOR operations can be easily implemented by $\lceil \frac{m}{w} \rceil$ AND and XOR instructions. As a result, Algorithm 3 requires $m\lceil \frac{m}{w} \rceil$ AND instructions and $T\lceil \frac{m}{w} \rceil (m-1)$ XOR instructions.

Although the cyclic shift operation is essentially free in hardware, it is costly in software. One way to reduce the number of cyclic shift operations is to store cyclically shifted versions of A and B into memory and read them from memory. In [27], all m different cyclic shifts of field elements of A and B are stored into two m arrays, each of which contains one cyclic shift of A and B. Thus, its memory requirement for each field element is about $2m^2$ bits. We use another efficient method which requires relatively less memory and is used in Algorithm 2 [14], [15]. This method, as stated in Step 1 of Algorithms 2 and 3, stores two arrays of

$$\begin{matrix} A[i] = (a_i,\ \cdots,\ a_{i+w-1 \bmod m}) \\ B[i] = (b_i,\ \cdots,\ b_{i+w-1 \bmod m}) \end{matrix},\ 0 \leq i \leq m-1,$$

and then, by reading $\lceil \frac{m}{w} \rceil$ w-bit units from the memory for generating each cyclic shifts of the elements as



A[0]
A[1]
A[i]
A[162]
a_0 a_1 ... a_{31}
a_1 a_2 ... a_{32}
a_i a_{i+1} ... a_{i+31}
a_{162} a_0 ... a_{30}
$m = 163$
$w = 32$
(a)
$w = 32$
A_1
A[1] A[33] A[65] A[97] A[129] A[161]
A_i
A[i] A[i + w] A[i + 2w] A[i + 3w] A[i + 4w] A[i + 5w]
nw
(b)

Fig. 1. (a) Precompute arrays for A_i ($m = 163$ and $w = 32$). (b) Generating A_1 and A_i using the arrays.

$$A_i = (A[i],\ A[i + w \bmod m],\ \cdots,\ A[i + (\lceil \tfrac{m}{w} \rceil - 1)w \bmod m]),$$
$$B_i = (B[i],\ B[i + w \bmod m],\ \cdots,\ B[i + (\lceil \tfrac{m}{w} \rceil - 1)w \bmod m]),$$
$$0 \le i \le m - 1.$$

This is shown in Fig. 1 for $m = 163$ and $w = 32$. To precompute each unit of arrays for A_i and B_i, ρ assorted instructions are required. The typical value of ρ for programming languages like C is four. Thus, the total number of assorted instructions needed for Step 1 of the algorithm is $2\rho m$.

For the memory requirement, we assume that each entry of $\mathbf{R}$ in Algorithm 3 is stored in at most $\lceil \log_2 m \rceil$ bits with a total of $T(m-1)\lceil \log_2 m \rceil$ bits. Also in Step 1 of this algorithm, another $2wm$ bits are required. Thus, the total memory requirement of the algorithm is $T(m-1)\lceil \log_2 m \rceil + 2wm$ bits, which includes the memory for storing the input matrix $\mathbf{R}$ and arrays used in Step 1 of the algorithm. For the number of memory accesses needed in Algorithm 3, one needs to consider accesses to each memory unit. This number is determined by taking into account the accesses due to $\mathbf{R}$, A_is and B_is which are $T(m-1)$, $\lceil \frac{m}{w} \rceil m$, and $T(m-1)\lceil \frac{m}{w} \rceil + \lceil \frac{m}{w} \rceil$, respectively. Now, we can conclude the above complexity analysis as follows:

Proposition 1. *The number of AND, XOR, and assorted instructions for Algorithm 3 are mn, $T(m-1)n$, and $2\rho m$, respectively. Also, Algorithm 3 requires $\approx T(m-1)\lceil \log_2 m \rceil + 2wm$ bits of memory with $\approx T(m-1)(n+1) + n(m+1)$ number of memory access, where $n = \lceil \frac{m}{w} \rceil$.*

4 A More Efficient Algorithm

Let us introduce the following lemma which helps us obtain the multiplication matrix $\mathbf{M}$ for GNBs [21].

Lemma 4. *For the type T GNB over $GF(2^m)$, when m is odd, the (i,j)th entry of the multiplication matrix $\mathbf{M}$ is*

$$M(i,j) = \mu_{i,0}^{(j)},\ 0 \le i,j \le m-1, \tag{11}$$

where $\mu_{i,0} = \beta^{2^i+1} = \sum_{j=0}^{m-1} \mu_{i,0}^{(j)} \beta^{2^j}$.

Proof. Since $\mathbf{M} = \mathbf{M}^{tr}$, it is sufficient to prove it only for $i < j$ and $0 \le i \le m-1$. For $i = 0$, (11) is obtained directly from part 1 of Lemma 3. Using (2) into (11), we need to show that

$$M(i,j) = M(m+i-j, m-j),\ 0 < i < j \le m-1. \tag{12}$$

Using part 2 of Lemma 1 and (2), one can see that (12) holds for self-dual normal bases. Thus, the proof is complete because all GNBs of even type are self-dual normal bases [26]. □

It is noted that the proof of Lemma 4 with another approach can be found in [28].

4.1 New Software Algorithm

The actual timing of software implementation of an algorithm depends not only on the total number of instructions, but also on the memory requirement and the number of memory accesses. The memory issue plays an important role in the analysis of algorithms that are to be implemented in software. This is especially important for resource constrained cryptosystems such as smart cards, where the storage memory is not only limited but also slower than what one would find in many other environments. Thus, in addition to reducing the number of instructions of an algorithm, we need to minimize its memory requirement and the number of memory accesses. To do so, in this section, we develop another algorithm which is based on the following property of the multiplication matrix $\mathbf{M}$ for GNBs [21]:

Corollary 2. *In the multiplication matrix $\mathbf{M}$ of GNBs for $GF(2^m)$, when m is odd, row $m-i$, $1 \le i \le \frac{m-1}{2}$, is the i-fold left cyclic shift of row i, i.e.,*

$$M(m-i,j) = M(i, j+i \bmod m),\ 1 \le i \le \frac{m-1}{2},\ 0 \le j \le m-1. \tag{13}$$

Proof. The proof follows from (11) and the fact that $\mu_{i,0} = \mu_{m-i,0}^{2^i}$ for $i > \frac{m-1}{2}$. □

Based on the above property, we can obtain the following relationship among the entries of matrix $\mathbf{R}$:

$$R(m-i,j) = R(i,j+i) - i \bmod m,\quad 1 \le i \le \frac{m-1}{2},\ 1 \le j \le T. \tag{14}$$

By applying (14) for a given i and all values of j, we then have

$$\begin{aligned} &B_{R(m-i,1)} \oplus B_{R(m-i,2)} \oplus \cdots \oplus B_{R(m-i,T)} \\ &\quad = \left(B_{R(i,1)} \oplus B_{R(i,2)} \oplus \cdots \oplus B_{R(i,T)}\right) \gg i. \end{aligned} \tag{15}$$

Let $\mathbf{R}'$ be an $\frac{m-1}{2} \times T$ matrix which contains the first half of matrix $\mathbf{R}$. Then, using (15), Theorem 1, and the fact that $A_{m-i} \odot (S \gg i) = (A \gg i) \odot (S \gg i) = (A \odot S) \gg i$, for any $A, S \in GF(2^m)$, $1 \leq i \leq \frac{m-1}{2}$, one can conclude the following, which is the key equation to our next multiplication algorithm.

Theorem 2. *For GNBs in $GF(2^m)$, when m is an odd number, the product of A and B is*

$$C = (A \odot B_1) \oplus \sum_{i=1}^{\frac{m-1}{2}} (((A \ll i) \odot S) \oplus ((A \odot S) \gg i)), \tag{16}$$

where

$$S = S(i) = B_{R'(i,1)} \oplus B_{R'(i,2)} \oplus \cdots \oplus B_{R'(i,T)}. \tag{17}$$

In (16), i-fold left cyclic shifts of A, i.e., $A \ll i$, are used instead of A_i, as appeared in (10). This is because, in the software implementation of Theorem 2 which is given below, we only store the cyclic shifts of one operand, namely, B. This will help us reduce the memory requirement to one half of that of Algorithm 3. The algorithm presented below is based on Theorem 2 and is given for odd m only. For m even, one can use the algorithms presented in [16].

Algorithm 4. (Efficient GNB Multiplication for $GF(2^m)$ when m is odd)

```
Input:
$A = (a_0, a_1, \cdots, a_{m-1}),\ B = (b_0, b_1, \cdots, b_{m-1}) \in GF(2^m)$,
and $\frac{m-1}{2} \times T$ matrix $\mathbf{R}'$
Output: $C = (c_0, c_1, \cdots, c_{m-1}) = AB$
1.   Precompute array for $B_i$
2.   Initialize $L := A$, $C := A \odot B_1$
3.   For $i = 1$ to $\frac{m-1}{2}$ {
4.       $L := L \ll 1$,
         $S := B_{R'(i,1)} \oplus B_{R'(i,2)} \oplus \cdots \oplus B_{R'(i,T)}$
5.       $C := C \oplus (L \odot S)$
6.       $R := (A \odot S) \gg i$
7.       $C := C \oplus R$ }
8.   Output $C = (c_0, c_1, \cdots, c_{m-1})$.
```

For the efficient software implementation of the above algorithm, one important issue is to minimize the number of CPU instructions that need to be executed. Compared to Algorithm 3, in Algorithm 4, the number of loop iterations has become half, which results in not only the number of XOR instructions being reduced, but also the number of memory accesses for reading B_i has become half. This will decrease the computational time of the algorithm when it is implemented on resource constrained cryptosystems. For this algorithm, one can perform analyses similar to the one presented for Algorithm 3. The results are given in the following:

TABLE 1
The Sequence of F for Type 4 GNB over $GF(2^7)$

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$F(k)$	0	1	5	2	1	6	5	3	3	2	4	0	4	6
k	15	16	17	18	19	20	21	22	23	24	25	26	27	28
$F(k)$	6	4	0	4	2	3	3	5	6	1	2	5	1	0

Proposition 2. *The number of AND, XOR, and assorted instructions for Algorithm 4 are mn, $\frac{T+1}{2}(m-1)n$, and $\rho m + \rho n(m-1)$, respectively. Also, Algorithm 4 requires $\approx \frac{T(m-1)}{2}\lceil \log_2 m \rceil + wm$ bits of memory with $\approx \frac{T(m-1)}{2}(n+1)$ number of memory access, where $n = \lceil \frac{m}{w} \rceil$.*

4.2 An Example

For type 4 GNB over $GF(2^7)$, one has $p = 29$ and $u = 12$ or 17. Then, the sequence of F is shown in Table 1, which is obtained from [2].

Thus, using (4), we can obtain

$$\begin{aligned} c_0 = {} & a_0 b_1 + a_1(b_0 + b_2 + b_5 + b_6) + a_2(b_1 + b_3 + b_4 + b_5) \\ & + a_3(b_2 + b_5) + a_4(b_2 + b_6) + a_5(b_1 + b_2 + b_3 + b_6) \\ & + a_6(b_1 + b_4 + b_5 + b_6), \end{aligned} \tag{18}$$

and corresponding $\mathbf{M}$, $\mathbf{R}$, and $\mathbf{R}'$ matrices, respectively, are

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}, \tag{19}$$

$$\mathbf{R} = \begin{pmatrix} 0 & 2 & 5 & 6 \\ 1 & 3 & 4 & 5 \\ 2 & 5 & 0 & 0 \\ 2 & 6 & 0 & 0 \\ 1 & 2 & 3 & 6 \\ 1 & 4 & 5 & 6 \end{pmatrix}, \quad \mathbf{R}' = \begin{pmatrix} 0 & 2 & 5 & 6 \\ 1 & 3 & 4 & 5 \\ 2 & 5 & 0 & 0 \end{pmatrix}.$$

It is seen from $\mathbf{M}$ that $C_N = 21$. Since $b_0 + b_0 = 0$, one can obtain c_0 from (18) as

$$\begin{aligned} c_0 = {} & a_0 b_1 + a_1(b_0 + b_2 + b_5 + b_6) + a_2(b_1 + b_3 + b_4 + b_5) \\ & + a_3(b_2 + b_5 + b_0 + b_0) + a_4(b_2 + b_6 + b_0 + b_0) \\ & + a_5(b_1 + b_2 + b_3 + b_6) + a_6(b_1 + b_4 + b_5 + b_6). \end{aligned} \tag{20}$$

Thus, the resultant multiplication is

TABLE 2
Generic Comparison of Multiplication Algorithms in Terms of Number of Instructions and Memory Requirements, $n = \lceil \frac{m}{w} \rceil$

Algorithms	# Instructions			Memory	
	XOR	AND	Assorted	Size in bits	# Accesses
Alg. 3 of [16]	$\frac{n}{2}(C_N + m - 2)$	mn	$\frac{\rho n}{2}(C_N + 2m - 1)$	$\frac{C_N - 1}{2} \lceil \log_2 m \rceil$	$\frac{C_N - 1}{2}$
Algorithm 3	$Tn(m-1)$	mn	$2\rho m$	$T(m-1)\lceil \log_2 m \rceil + 2wm$	$T(m-1) + T(m-1)n$
Algorithm 4	$\frac{T+1}{2} n(m-1)$	mn	$\rho m + \rho n(m-1)$	$\frac{T(m-1)}{2} \lceil \log_2 m \rceil + wm$	$\frac{T(m-1)}{2} + \frac{T(m-1)n}{2}$

Note that $2m - 1 \leq C_N \leq Tm - T + 1$.

TABLE 3
Comparison of Software Implementations of the GNB Multiplication for the Five Binary Fields Recommended by NIST for ECDSA

Platform	m	Gaussian Normal Basis				Polynomial Basis
		T	Alg. 3 of [16]	Algorithm 3	Algorithm 4	[29]
Sun Blade 100	163	4	4.73	2.95	2.12	1
	233	2	5.74	3.37	2.79	1.93
	283	6	15.3	10.66	6.71	2.13
	409	4	24.25	15.76	10.63	4.46
	571	10	90.45	73.17	38.79	7.32
Pentium 4	163	4	8.12	4.62	4.50	1
	233	2	10	5.25	6.13	1.88
	283	6	27.38	17.25	13.50	2.00
	409	4	42.38	24.50	22.12	4.13
	571	10	160.50	107.38	75.38	5.75

$$\begin{aligned} C = (A \odot B_1) &\oplus (A_1 \odot (B_0 \oplus B_2 \oplus B_5 \oplus B_6)) \\ &\oplus (A_2 \odot (B_1 \oplus B_3 \oplus B_4 \oplus B_5)) \\ &\oplus (A_3 \odot (B_2 \oplus B_5 \oplus B_0 \oplus B_0)) \\ &\oplus (A_4 \odot (B_2 \oplus B_6 \oplus B_0 \oplus B_0)) \\ &\oplus (A_5 \odot (B_1 \oplus B_2 \oplus B_3 \oplus B_6)) \\ &\oplus (A_6 \odot (B_1 \oplus B_4 \oplus B_5 \oplus B_6)), \end{aligned}$$

which simplifies to

$$\begin{aligned} C = (A \odot B_1) &\oplus ((A \ll 1) \odot (B_0 \oplus B_2 \oplus B_5 \oplus B_6)) \\ &\oplus ((A \ll 2) \odot (B_1 \oplus B_3 \oplus B_4 \oplus B_5)) \\ &\oplus ((A \ll 3) \odot (B_2 \oplus B_5 \oplus B_0 \oplus B_0)) \\ &\oplus ((A \odot (B_5 \oplus B_2 \oplus B_0 \oplus B_0)) \gg 3) \\ &\oplus ((A \odot (B_3 \oplus B_4 \oplus B_5 \oplus B_1)) \gg 2) \\ &\oplus ((A \odot (B_2 \oplus B_5 \oplus B_6 \oplus B_0)) \gg 1). \end{aligned}$$

4.3 Comparison

Table 2 compares the multiplication algorithms of Reyhani-Hasan (Algorithm 3 in [16]) and the modified Ning-Yin (Algorithm 3) and the proposed algorithm (Algorithm 4) in terms of the number of processor or CPU instructions and memory requirements. As seen in the table, our proposed Algorithm 4 requires the least amount of memory after Algorithm 3 of [16].

During the preparation of this paper, independent work from Dahab et al. appeared [18]. It is noted that the key equation of their new algorithm, i.e., Algorithm 4 of [18], is similar to (16). Then, Algorithm 4 (and, hence, Algorithm 7) in [18] requires the same number of operations and memory requirements as the one proposed here, i.e., Algorithm 4 of this paper.

We have coded normal basis algorithms given in Table 2 and the best polynomial basis multiplication algorithm presented in [29]. Table 3 compares the multiplication algorithms for the five binary fields recommended by NIST for ECDSA applications. The codes have been written using the C programming language and have been executed on two platforms—a Sun Blade 100 with SparcV9 processor clocked at 502MHz, 512MB of RAM, and a PC with Pentium 4 clocked at 3.00 GHz, 1.00 GB of RAM. It is noted that our codes have not been optimized using techniques such as loop unrolling and reducing register consumption as compared with the ones used in [18]. Therefore, we have reported the ratios of the obtained timings to the timing of the $GF(2^{163})$ polynomial basis multiplication in Table 3. As seen in this table, although the proposed algorithm (Algorithm 4) is faster than the previously reported algorithms using GNB, it is still slow as compared with the multiplication algorithm using polynomial basis.

In the following two sections, two new digit-level architectures for multiplication using GNBs are proposed. The multiplication operation in both architectures require $\lceil \frac{m}{d} \rceil$ clock cycles. Based on the available space for a given multiplier, the parameter d, $1 \leq d \leq m$, can be chosen by the designer to meet the gate counts and timing requirements. It is noted that, throughout the following two sections, $A_{m-i} = A \gg i = A^{2^i}$ can be obtained by i-fold right cyclic shifts of the coordinates A and it is free in hardware implementation.

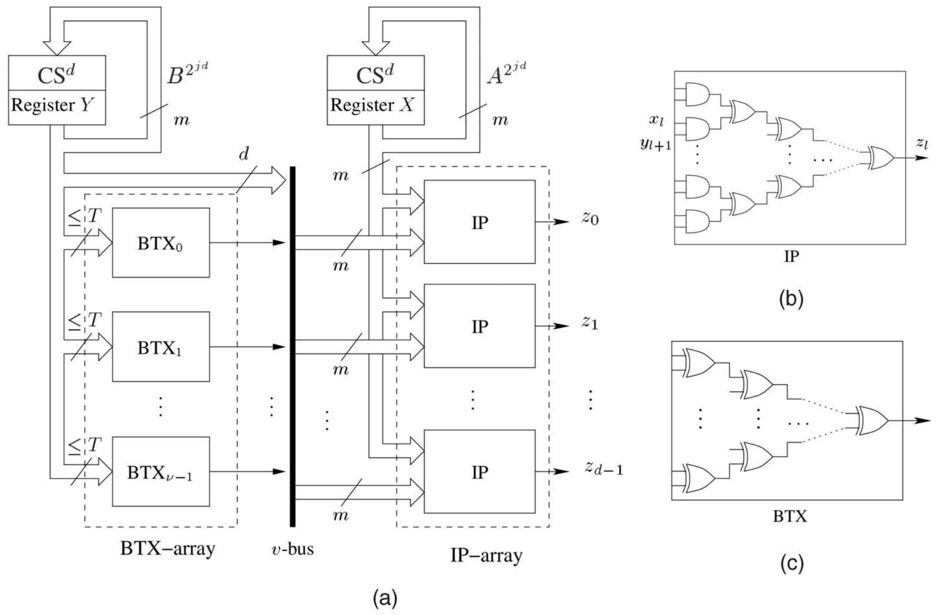


Fig. 2. (a) The architecture of digit-level GNB multiplier with serial output (DLGM_S). (b) and (c) Details of the IP (inner product) and the BTX (binary tree of XOR gates) blocks. CS^d is the d-fold right cyclic shift.

5 New Digit-Level Multiplier with Serial Output

We now present a new digit-level multiplier which requires $\lceil\frac{m}{d}\rceil$ clock cycles and generates d, $1 \leq d \leq m$, coordinates of $C = AB$ in each clock cycle.

Let us denote

$$z_l = \underline{x}\mathbf{M}^{(l)}\underline{y}^{tr},\ 0 \leq l \leq d-1, \tag{21}$$

where $\mathbf{M}^{(l)}$, as mentioned earlier, is the l-fold right and down circular shifts of the multiplication matrix $\mathbf{M} = \mathbf{M}^{(0)}$. Using (1), one can verify that the first d coordinates of C can be obtained from (21), i.e., $c_l = z_l,\ 0 \leq l \leq d-1$, if $\underline{x} = \underline{a}$ and $\underline{y} = \underline{b}$. Consecutive d coordinates of C, i.e.,

$$c_{-jd \bmod m}, \cdots, c_{-jd+d-1 \bmod m}, 1 \leq j \leq \left\lceil\frac{m}{d}\right\rceil - 1,$$

can be found from (21) if $\underline{x}$ and $\underline{y}$ are to be replaced with their consecutive d-fold right cyclic shifts, i.e.,

$$[a_{-jd \bmod m}, \cdots, a_{-jd+m-1 \bmod m}]$$

and

$$[b_{-jd \bmod m}, \cdots, b_{-jd+m-1 \bmod m}],$$

respectively.

5.1 Architecture

To realize d equations of (21), the architecture of Fig. 2a, which is hereafter referred to as digit-level GNB multiplier with serial output (DLGM_S), is proposed. In this structure, two m-bit registers of X and Y, which correspond to $\underline{x}$ and $\underline{y}$ in (21), are initialized $(j = 0)$ with the coordinates of $A = (a_0, a_1, \cdots, a_{m-1})$ and $B = (b_0, b_1, \cdots, b_{m-1})$, respectively. The d-fold right cyclic shifts are realized by two CS^d blocks in Fig. 2a. Thus, after the jth, $1 \leq j \leq \lceil\frac{m}{d}\rceil - 1$, clock cycle, the registers X and Y contain, respectively, the coordinates of $A^{2^{(jd)}} = (a_{-jd \bmod m}, \cdots, a_{-jd+m-1 \bmod m})$ and $B^{2^{(jd)}} = (b_{-jd \bmod m}, \cdots, b_{-jd+m-1 \bmod m})$ and the multiplier generates d coordinates of C, i.e., $c_{l-jd \bmod m} = z_l,\ 0 \leq l \leq d-1$, $0 \leq j \leq \lceil\frac{m}{d}\rceil - 1$.

The multiplier structure of Fig. 2a consists of two blocks of arrays, namely, BTX-array and IP-array. The BTX-array generates all entries of d vectors of

$$\mathbf{v}^{(l)} = [v_0^{(l)},\ v_1^{(l)},\ \cdots,\ v_{m-1}^{(l)}]^{tr} = \mathbf{M}^{(l)}\underline{y}^{tr},\ 0 \leq l \leq d-1 \tag{22}$$

using ν binary tree of XOR ($\mathrm{BTX}_{0\cdots\nu-1}$) gates, where the value of ν will be determined in the end of this section. After generating all entries of (22), the IP-array block of Fig. 2a realizes d equations of $z_l = \underline{x}\mathbf{v}^{(l)} = \sum_{i=0}^{m-1} x_i v_i^{(l)},\ 0 \leq l \leq d-1$, using d inner product (IP) blocks. Each IP block, as shown in Fig. 2b, requires m AND gates and $m-1$ XOR gates. Thus, the IP-array consists of dm AND gates and $d(m-1)$ XOR gates.

To implement a single entry $v_i^{(l)},\ 0 \leq i \leq m-1$, $0 \leq l \leq d-1$, in (22), we need to consider row i of $\mathbf{M}^{(l)}$. Using Lemma 3 and the fact that $\mathbf{M}^{(l)}$ is the l-fold right and down circular shifts of $\mathbf{M} = \mathbf{M}^{(0)}$, we can see that 1) row l of $\mathbf{M}^{(l)}$ has one nonzero entry (which is in column $l+1$) and 2) all other rows have at most T 1s. Thus, there is no gate needed for realizing the d entries of $v_l^{(l)} = y_{l+1}$, $0 \leq l \leq d-1$. This corresponds to the d lines on the v-bus in Fig. 2a which are connected to $y_1, y_2, \cdots, y_d$ of the register Y. Also, one can verify that each entry of $v_i^{(l)}$, $i \neq l$, in (22), which is realized by a BTX block (see Fig. 2c), requires at most $T-1$ two-input XOR gates. As a result, the total number of XOR gates in the BTX-array of the multiplier is $\leq \nu(T-1)$. In order to determine the number of XOR gates in the multiplier, we need to obtain ν, which is the total number of different rows with more than one nonzero entry in $\mathbf{M}^{(l)}$s for $0 \leq l \leq d-1$.

It is noted that ν is upper bounded by $d(m-1)$. However, the following helps us to obtain the exact value of ν.

Lemma 5. *For any $0 \le k, j \le m-1$, $k \ne j$, the row k of $\mathbf{M}^{(j)}$ (denoted as $\mu^{(j)}_{k,*}$) is the same as the row j of $\mathbf{M}^{(k)}$ (denoted as $\mu^{(k)}_{j,*}$), i.e., $\mu^{(j)}_{k,*} = \mu^{(k)}_{j,*}$.*

Proof. Without loss of generality, we can assume that, $1 \le (j-k) \bmod m \le \frac{m-1}{2}$, for given values of k and j. Since the matrix $\mathbf{M}^{(j)}$ is the $(j-k)$-fold right and down circular shifts of the matrix $\mathbf{M}^{(k)}$, the row k of $\mathbf{M}^{(j)}$ is the $(j-k)$-fold right cyclic shifts of the row $k-(j-k) = 2k-j$ of $\mathbf{M}^{(k)}$, i.e.,

$$\mu^{(j)}_{k,*} = \mu^{(k)}_{2k-j,*} \gg (j-k). \tag{23}$$

Since the matrix $\mathbf{M}^{(k)}$ is also the k-fold right and down circular shifts of the matrix $\mathbf{M}$, we can have

$$\mu^{(k)}_{i+k,*} = \mu^{(0)}_{i,*} \gg k, \; 0 \le i \le m-1. \tag{24}$$

Using Corollary 2, one can obtain the row $j-k$ of $\mathbf{M}$ by the $j-k$-fold right cyclic shifts of the row $m-(j-k)$, i.e.,

$$\mu^{(0)}_{j-k,*} = \mu^{(0)}_{m-(j-k),*} \gg (j-k). \tag{25}$$

Also, using (24) with $i = j-k$ and (25), we can obtain

$$\mu^{(k)}_{j,*} = (\mu^{(0)}_{m-j+k,*} \gg (j-k)) \gg k = (\mu^{(0)}_{m-j+k,*} \gg k) \gg (j-k). \tag{26}$$

Substituting $i = m-j+k$ into (24), we have

$$\mu^{(k)}_{m-j+2k,*} = \mu^{(0)}_{m-j+k,*} \gg k. \tag{27}$$

From (26) and (27), one can obtain $\mu^{(k)}_{j,*} = \mu^{(k)}_{m-j+2k,*} \gg (j-k)$ and then the proof is complete by using (23). □

5.2 Complexity

Using the above lemma, one can conclude the following:

Corollary 3. *The total number of different rows with more than one nonzero entry in $\mathbf{M}, \mathbf{M}^{(1)}, \cdots,$ and $\mathbf{M}^{(d-1)}$ is $\nu = d(m-1) - \frac{d(d-1)}{2}$.*

Proof. Since each $\mathbf{M}^{(l)}$, $0 \le l \le d-1$, has $m-1$ rows with more than one 1, the total number of rows in all $\mathbf{M}^{(l)}$s is $d(m-1)$. Using Lemma 5, every two $\mathbf{M}^{(i)}$ and $\mathbf{M}^{(j)}$ have a common row with the total of $\binom{d}{2} = \frac{d(d-1)}{2}$ common rows in all $\mathbf{M}, \mathbf{M}^{(1)}, \cdots,$ and $\mathbf{M}^{(d-1)}$. Thus, the proof is complete by removing the number of common rows from the total ones, i.e., $\nu = d(m-1) - \frac{d(d-1)}{2}$. □

Based on the above discussions, we can state the gate counts and time delay of the proposed digit-level multiplier as follows:

Proposition 3. *For type T GNB over $GF(2^m)$, the proposed digit-level GNB multiplier with serial output (DLGM$_S$) requires dm two-input AND gates, $\le d((m - \frac{d+1}{2})T + \frac{d-1}{2})$ two-input XOR gates, and $2m$ one-bit latches. Also, the critical path delay of the multiplier is $\le T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil)T_X$, where T_A and T_X are the time delay of a two-input AND gate and a two-input XOR gate, respectively.*

Proof. As seen in Fig. 2, the number of AND gates and latches are dm and $2m$, respectively. The total number of XOR gates in the BTX-array and the IP-array are $\le \nu(T-1) = d(m-1-\frac{d-1}{2})(T-1)$ and $d(m-1)$, respectively. Thus, the total XOR gate count is $\le d((m-\frac{d+1}{2})T + \frac{d-1}{2})$. The time delay can be obtained by adding the delay of the BTX-array, i.e., $\le \lceil \log_2 T \rceil T_X$, and the delay of the IP-array, i.e., $T_A + \lceil \log_2 m \rceil T_X$. □

It is noted that the number of XOR gates in the BTX-array can be reduced if some common terms among BTX blocks are reused. This is shown in the following example.

5.3 An Example

For type 4 GNB over $GF(2^7)$, let $d = 2$. Using (19), we have

$$\mathbf{M} = \begin{pmatrix} 0&1&0&0&0&0&0 \\ 1&0&1&0&0&1&1 \\ 0&1&0&1&1&1&0 \\ 0&0&1&0&0&1&0 \\ 0&0&1&0&0&0&1 \\ 0&1&1&1&0&0&1 \\ 0&1&0&0&1&1&1 \end{pmatrix},$$

$$\mathbf{M}^{(1)} = \begin{pmatrix} 1&0&1&0&0&1&1 \\ 0&0&1&0&0&0&0 \\ 1&1&0&1&0&0&1 \\ 0&0&1&0&1&1&1 \\ 0&0&0&1&0&0&1 \\ 1&0&0&1&0&0&0 \\ 1&0&1&1&1&0&0 \end{pmatrix}.$$

Thus, the resultant lines on the v-bus of Fig. 2a which are implemented using 16 XOR gates are

$$\begin{aligned} \mathbf{v}^{(0)} &= [y_1,\ (y_2+y_5)+(y_0+y_6),\ (y_4+y_5)+(y_1+y_3), \\ &\quad y_{25},\ y_2+y_6,\ y_{13}+y_{26},\ (y_1+y_6)+y_{45}]^{tr}, \\ \mathbf{v}^{(1)} &= [v^{(0)}_1,\ y_2,\ (y_0+y_3)+y_{16},\ y_{26}+y_{45},\ y_3+y_6,\ y_{03},\ y_{03} \\ &\quad + (y_2+y_4)]^{tr}. \end{aligned} \tag{28}$$

In (28), $v^{(0)}_1 = (y_2+y_5)+(y_0+y_6)$ and $y_{ij} = y_i + y_j$s are reused terms in the BTX-array, where y_i, $0 \le i \le m-1$, is the ith bit of the register $Y = (y_0, y_1, \cdots, y_{m-1})$. Also, the IP-array requires $2 \times (7-1) = 12$ XOR gates and 14 AND gates for the implementation of $z_l = \sum_{i=0}^{6} x_i v^{(l)}_i$, $l = 0, 1$. As a result, the multiplier requires 28 XOR gates and 14 AND gates with the time delay of $T_A + 5T_X$.

6 New Digit-Level Multiplier with Parallel Output

Here, we present another architecture for multiplication of two field elements of A and B in such a way that all m coordinates of $C = AB$ are available simultaneously at the end of final clock cycle. Similar to the previous architecture, the total number of clock cycles needed for the multiplication is $q = \lceil \frac{m}{d} \rceil$, where d, $1 \le d \le m$, is denoted as the number of bits in each digit.

6.1 Formulation

Let h_i be the number of 1s in the row i, $1 \le i \le m-1$, of the multiplication matrix $\mathbf{M}$. It was shown that h_i is an even number, where $2 \le h_i \le T$. Let us define

$$S'(k,B) = (B_{R(2k,1)} \oplus B_{R(2k,2)} \oplus \cdots \oplus B_{R(2k,h_{2k})}) \gg k, \quad 1 \le k \le \frac{m-1}{2}, \tag{29}$$

where $R(i,j)$, $1 \le i \le m-1$, is the (i,j)th entry of the $m-1 \times T$ matrix $\mathbf{R}$ used in Theorem 1. Then, we can state the following:

Lemma 6. *Let $C = AB$, $A, B, C \in GF(2^m)$. Then,*

$$C = (A \odot B_1) \oplus \sum_{k=1}^{\frac{m-1}{2}} (((A_k \odot S'(k,B)) \ll k) \oplus ((A_{m-k} \odot S'(k,B)) \gg k)), \tag{30}$$

where $S'(k,B) \in GF(2^m)$ has been defined in (29).

Proof. Using (10) and the fact that $\sum_{j=h_i+1}^{T} B_{R(i,j)} = 0$, we have

$$C = (A \odot B_1) \oplus \sum_{i=1}^{m-1} A_i \odot S_i, \tag{31}$$

where $S_i = B_{R(i,1)} \oplus B_{R(i,2)} \oplus \cdots \oplus B_{R(i,h_i)}$. Since m is odd, $m-2k$ is odd. Then, we can write the term in the summation of (31) into two terms over even and odd values of i, i.e., $i = 2k$ and $i = m-2k$, as

$$\sum_{i=1}^{m-1} A_i \odot S_i = \sum_{k=1}^{\frac{m-1}{2}} ((A_{2k} \odot S_{2k}) \oplus (A_{m-2k} \odot S_{m-2k})) \tag{32}$$

$$= \sum_{k=1}^{\frac{m-1}{2}} (((A_k \ll k) \odot S_{2k}) \oplus ((A_{m-k} \gg k) \odot S_{m-2k})). \tag{33}$$

Using (15) and (29), S_{m-2k} in (33) can be written as

$$S_{m-2k} = S_{2k} \gg 2k = (S_{2k} \gg k) \gg k = S'(k,B) \gg k. \tag{34}$$

Also, S_{2k} in (33) can be written as

$$S_{2k} = (S_{2k} \gg k) \ll k = S'(k,B) \ll k. \tag{35}$$

Substituting (34) and (35) into (33) and using the following:

$$(A_k \ll k) \odot (S'(k,B) \ll k) = (A_k \odot S'(k,B)) \ll k$$
$$(A_{m-k} \gg k) \odot (S'(k,B) \gg k) = (A_{m-k} \odot S'(k,B)) \gg k,$$

the proof is complete. □

Let us denote that $S'(m-k,B) = S'(k,B)$, $1 \le k \le \frac{m-1}{2}$. Then, (30) can be stated as follows if we use $(A_k \odot S'(k,B)) \ll k = (A_k \odot S'(k,B))^{2^{m-k}}$ and

$$(A_{m-k} \odot S'(k,B)) \gg k = (A_{m-k} \odot S'(k,B))^{2^k}.$$

Corollary 4. *Given $A, B \in GF(2^m)$,*

$$C = AB = \sum_{k=0}^{m-1} (A_{m-k} \odot S'(k,B))^{2^k}, \tag{36}$$

where $S'(0,B) = B_1$.

Let $S'(k,B) \in GF(2^m)$ be represented with respect to the GNB as $S'(k,B) = \sum_{l=0}^{m-1} s'_l(k,B)\beta^{2^l}$, where $s'_l(k,B) \in \{0,1\}$. We now define a field element,

$$J(X,Y) = \sum_{k=0}^{m-1} x_{m-k} s'_0(k,Y) \beta^{2^k}, \tag{37}$$

as a function of two field elements of $X = (x_0, x_1, \cdots, x_{m-1})$ and $Y = (y_0, y_1, \cdots, y_{m-1})$, where $s'_0(0,Y) = y_1$ and $s'_0(k,Y) = s'_0(m-k,Y)$ for $1 \le k \le \frac{m-1}{2}$. Then, one can write (36) as

$$C = \sum_{k=0}^{m-1} \left(\sum_{l=0}^{m-1} a_{m-k+l} s'_l(k,B) \beta^{2^l} \right)^{2^k} = \sum_{l=0}^{m-1} \left(\sum_{k=0}^{m-1} a_{m-k+l} s'_l(k,B) \beta^{2^k} \right)^{2^l} \tag{38}$$

$$= \sum_{l=0}^{m-1} J^{2^l}(A^{2^{m-l}}, B^{2^{m-l}}). \tag{39}$$

Let $q = \lceil \frac{m}{d} \rceil$ be the number of clock cycles needed for the multiplication. Then, we can write $m = qd - r$, where $0 \le r \le d-1$. Let us define

$$Z(X,Y) = (\cdots((L^{2^d}(X,Y) \oplus L(X^{2^d}, Y^{2^d}))^{2^d} \oplus \cdots \oplus L(X^{2^{(q-2)d}}, Y^{2^{(q-2)d}}))^{2^d} \oplus L_0(X^{2^{(q-1)d}}, Y^{2^{(q-1)d}}), \tag{40}$$

where

$$L(X,Y) = \sum_{i=0}^{d-1} J^{2^{d-1-i}}(X^{2^i}, Y^{2^i}) \tag{41}$$

and

$$L_0(X,Y) = \sum_{i=0}^{d-r-1} J^{2^{d-1-i}}(X^{2^i}, Y^{2^i}). \tag{42}$$

By comparing (39) with (40) for the values of $X = A^2$ and $Y = B^2$, one can verify that $Z(A^2, B^2) = C^{2^r}$. Thus, $C = AB$ can be obtained from (40) if $X = A^{2^{1-r}}$ and $Y = B^{2^{1-r}}$, i.e., $Z(A^{2^{1-r}}, B^{2^{1-r}}) = C$.

6.2 Architecture

In order to realize (40), the structure of Fig. 3a is proposed. This multiplier is hereafter referred to as the digit-level GNB multiplier with parallel output (DLGM$_P$). Let $Z_{(j)}$ denote the content of the output register Z at the jth, $0 \le j \le q-1$, clock cycle. In the initialization step of this multiplier ($j = 0$), the output register Z should be cleared, i.e., $Z_{(0)} = 0$, and the input registers of X and Y should be loaded by the coordinates of $X_{(0)} = A^{2^{1-r}}$ and $Y_{(0)} = B^{2^{1-r}}$,

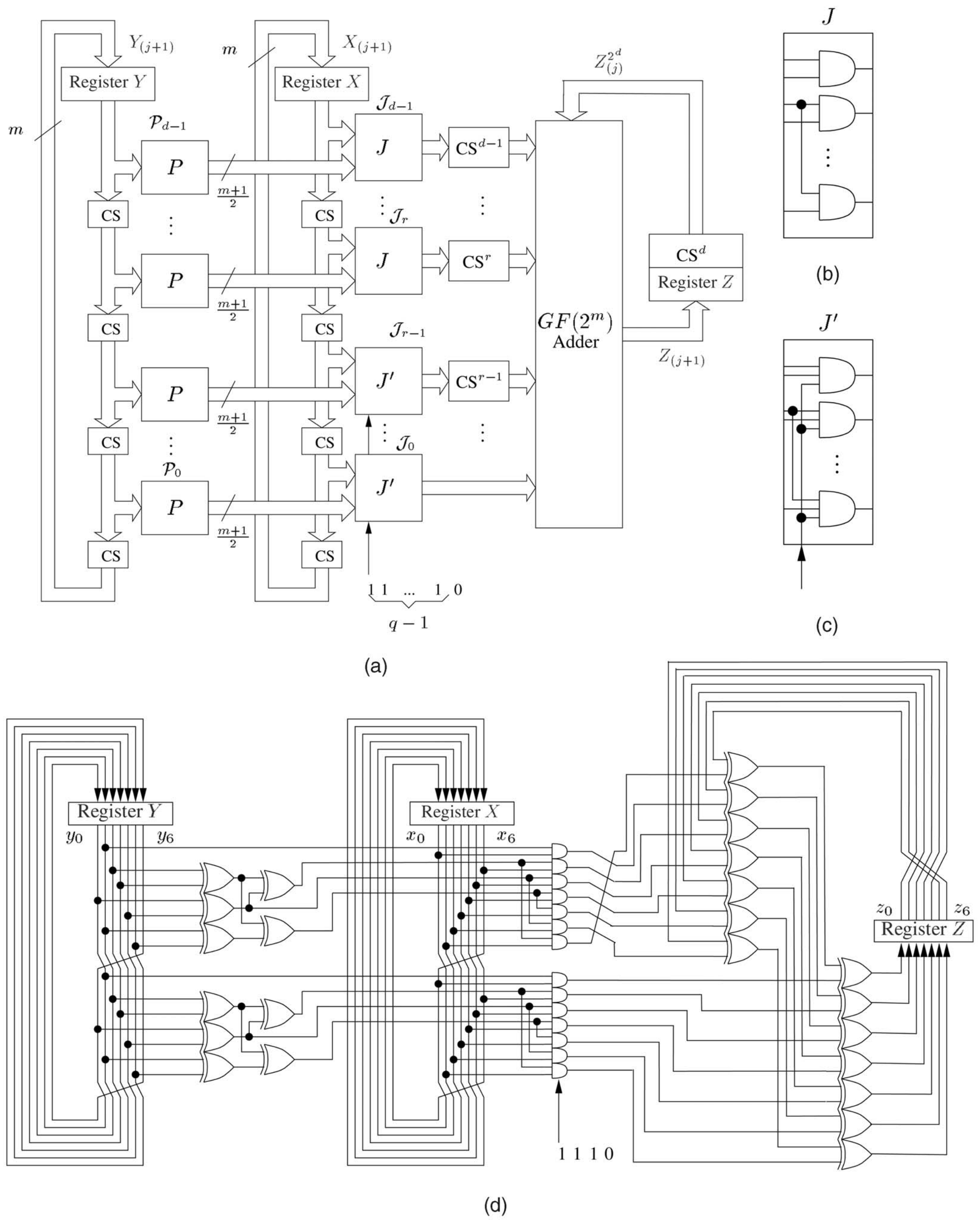


Fig. 3. (a) The proposed digit-level GNB multiplier with parallel output (DLGM_p). (b) and (c) Details of J and J' blocks. (d) The type 4 DLGM_p over $GF(2^7)$ ($d = 2$, $r = 1$).

respectively. Then, by writing (40) into the following recursive equations:

$$Z_{(j+1)} = Z_{(j)}^{2^d} \oplus L(X_{(j)}, Y_{(j)}), \tag{43}$$

$$X_{(j+1)} = X_{(j)}^{2^d}, \tag{44}$$

$$Y_{(j+1)} = Y_{(j)}^{2^d}, \tag{45}$$

one can verify that, after q clock cycles, the output register contains the coordinates of $C = AB$, i.e., $Z_{(q)} = C$.

In Fig. 3a, the CS^d block is a d-fold right cyclic shift which generates $Z_{(j)}^{2^d}$ in (43). Also, two d CS (cyclic shift) blocks in the paths between the input and output of the registers X and Y realize two equations of (44) and (45), respectively.

$L(X_{(j)}, Y_{(j)})$ in (43) is implemented by adding d inputs to the left side of the $GF(2^m)$ adder in the architecture. Each input corresponds to a term in (41) for $0 \le j \le q-2$. During the last clock cycle, i.e., $j = q-1$, all r inputs generated from the $\mathcal{J}_0, \cdots, \mathcal{J}_{r-1}$ blocks to the left side of the $GF(2^m)$ adder are $0 \in GF(2^m)$ and the remaining inputs correspond to the terms appearing in (42). This is controlled by a signal, which is $0 \in GF(2)$ only during the last clock cycle, connected to the $\mathcal{J}_0, \cdots, \mathcal{J}_{r-1}$ blocks. To implement each term in (41), one can write the function $J(X, Y)$ in (37) as

$$J(X, Y) = X'(X) \odot P(Y), \tag{46}$$

TABLE 4
Comparison of Digit-Level Gaussian Normal Basis Multipliers in Terms of Space Complexity

GNB Multipliers	#AND		#XOR		#Latches
	type T	type 2	type T	type 2	
Digit-level structures $(1 \leq d \leq m)$					
DLMO [3]	dC_N	$d(2m-1)$	$\leq d(C_N-1)$	$d(2m-2)$	$2m$
IMO [9]	dm	dm	$\leq d(C_N-1)$	$d(2m-2)$	$2m$
AEDS [22]	$\gamma+d$	$d(m-0.5d+0.5)$	$2\gamma+\frac{d}{2}(C_N-1)$	$d(3m-d-2)$	$2m$
XEDS [22]	$2\gamma+d$	$d(2m-d)$	$\gamma+\frac{d}{2}(C_N-1)$	$d(2m-0.5d-1.5)$	$2m$
DLGM$_S$	dm	dm	$\leq d((m-\frac{d+1}{2})T+\frac{d-1}{2})$	$d(2m-0.5d-1.5)$	$2m$
d-SMPO$_{\text{I}}$ [10]	$d\frac{m+1}{2}+m$	$d\frac{m+1}{2}+m$	$\leq \frac{d}{2}(C_N+2m-1)$	$d(2m-1)$	$3m$
d-SMPO$_{\text{II}}$ [10]	$dm+m$	$dm+m$	$\leq \frac{d}{2}(C_N+m)$	$\frac{d}{2}(3m-1)$	$3m$
DLGM$_P$	dm	dm	$\leq \frac{d}{2}(C_N+m)$	$\frac{d}{2}(3m-1)$	$3m$
Bit-serial structures $(d=1)$					
[28]	m	m	$\leq m+\frac{m-1}{2}(T-1)$	$\frac{1}{2}(3m-1)$	$3m$
[32]	-	m	-	$\frac{1}{2}(3m-1)$	$3m$
Bit-parallel structures $(d=m)$					
[13]	m^2	m^2	$\leq \frac{m}{2}(C_N+m-2)$	$\frac{3m}{2}(m-1)$	-
[12], [33]	-	m^2	-	$\frac{3m}{2}(m-1)$	-

Note that $2m-1 =\leq C_N \leq Tm-T+1$.

where $X'(X)=(x_0,\, x_{m-1},\, x_{m-2},\, \cdots,\, x_2, x_1)$ and

$$P(Y)=(y_1,\, s'_0(1,Y),\, s'_0(2,Y),\, \cdots,\, s'_0(2,Y),\, s'_0(1,Y)). \quad (47)$$

In Fig. 3a, the input of $\mathcal{P}_{d-1-i}$, $0 \leq i \leq d-1$, is $Y_{(j)}^{2^i}$ and it implements $P(Y_{(j)}^{2^i})$ using (47), where $s'_0(k,Y)$ can be found from (29). Then, the output of $\mathcal{J}_{d-1-i}$, $0 \leq i \leq d-1$, implements $J(X_{(j)}^{2^i}, Y_{(j)}^{2^i})$ using (46).

For the purpose of illustration, the architecture of DLGM$_P$ $(d=2,\ r=1)$ for the type 4 GNB over $GF(2^7)$ is shown in Fig. 3d. In this figure, the X and Y registers should be initialized with the coordinates of $A^{2^{1-r}}=A$ and $B^{2^{1-r}}=B$, respectively. Initializing the Z register with $(0,0,\cdots,0)$, it will contain the coordinates of $C=AB$ after four clock cycles. In Fig. 3d, $s'_0(1,Y)=y_{1-1}+y_{3-1}+y_{4-1}+y_{5-1}=(y_2+y_3)+(y_0+y_4)$, $s'_0(2,Y)=y_{2-2}+y_{6-2}=y_0+y_4$, and $s'_0(3,Y)=y_{1-3}+y_{4-3}+y_{5-3}+y_{6-3}=(y_5+y_1)+(y_2+y_3)$ are obtained from (29) using (19).

6.3 Complexity

From Fig. 3, one can easily see that the DLGM$_P$ architecture requires $(d-r)m$ two-input AND gates and rm three-input AND gates. Using (29), one can figure out that $s'_0(k,Y)$ requires $h_{2k}-1$ XOR gates. Since $\sum_{k=1}^{\frac{m-1}{2}} h_k = 0.5(C_N-1)$ [30] and $\sum_{k=1}^{\frac{m-1}{2}} h_{2k} = \sum_{k=1}^{\frac{m-1}{2}} h_k$, the P block in Fig. 3a requires $\frac{C_N-m}{2}$ XOR gates. It is noted that the number of XOR gates can be reduced if some common terms are reused. These terms are common between $s'_0(k,Y)$s inside a single P block, as shown in Fig. 3d, and/or different P blocks. Thus, the number of XOR gates in the P blocks is $\leq d\frac{C_N-m}{2}$. Since the $GF(2^m)$ adder requires md XOR gates, the total number of XOR gates is $\leq d\frac{C_N+m}{2}$.

The critical path delay of the multiplier includes the delays of the P block ($\lceil \log_2 T \rceil T_X$), the J block (T_A), and the $GF(2^m)$ adder ($\leq \lceil \log_2(d+1) \rceil T_X$). In order to have an optimized design, we choose r between two consecutive integers such that $r \leq d-r$. Thus, the delay of the multiplier is reduced to $\leq T_A + (\lceil \log_2 T \rceil + \lceil \log_2(d-r+1) \rceil)T_X$. From the above discussion, we can state the complexities of the DLGM$_P$ as follows:

Proposition 4. *For type T GNB over $GF(2^m)$, the proposed digit-level GNB multiplier with parallel output (DLGM$_P$) requires dm AND gates,*[2] *$\leq d(\frac{C_N+m}{2})$ XOR gates, and $3m$ latches. Also, the critical path delay of the multiplier is $\leq T_A + (\lceil \log_2 T \rceil + \lceil \log_2(d-r+1) \rceil)T_X$, where T_A and T_X are the time delay of a two-input AND gate and a two-input XOR gate, respectively.*

Remark 4. The bit-parallel $(d=m,\ r=0)$ architecture of Fig. 3 can be obtained by removing the three registers of X, Y, and Z as well as removing the connection of the output of the Z register to the top input of the $GF(2^m)$ adder. Thus, the number of XOR gates of the $GF(2^m)$ adder is reduced to $m(d-1)$. Therefore, the number of XOR gates and time delay of the multiplier are $\leq m(\frac{C_N+m-2}{2})$ and $\leq T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil)T_X$, respectively. These match the complexities of the multiplier proposed in [13]. □

6.4 Comparison

Tables 4 and 5 compare the proposed digit-level architectures with the existing ones in terms of number of AND gates, XOR gates, latches, and critical delay. In these tables, the DLMO (digit-level Massey-Omura) multiplier uses d identical bit-level Massey-Omura multipliers [3] and the IMO (improved Massey-Omura) multiplier is the one

2. It includes $(d-r)m$ two-input AND gates and rm three-input AND gates.

TABLE 5
Comparison of Digit-Level Gaussian Normal Basis Multipliers in Terms of Time Complexity

GNB Multipliers	Critical Path Delay		Output
	type T	type 2	
Digit-level structures ($1 \leq d \leq m$)			
DLMO [3]	$T_A + \lceil \log_2 C_N \rceil T_X$	$T_A + \lceil \log_2 (2m-1) \rceil T_X$	serial
IMO [9]	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil) T_X$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	serial
AEDS [22]	$T_A + \lceil \log_2 C_N \rceil T_X$	$T_A + \lceil \log_2 (2m-1) \rceil T_X$	serial
XEDS [22]	$T_A + \lceil \log_2 C_N \rceil T_X$	$T_A + \lceil \log_2 (2m-1) \rceil T_X$	serial
DLGM$_S$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil) T_X$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	serial
d-SMPO$_{\text{I}}$ [10]	$2T_A + (1 + \lceil \log_2 (d-1) \rceil + \lceil \log_2 (T+1) \rceil) T_X$	$2T_A + (3 + \lceil \log_2 (d-1) \rceil) T_X$	parallel
d-SMPO$_{\text{II}}$ [10]	$2T_A + (1 + \lceil \log_2 (d-1) \rceil + \lceil \log_2 (T+1) \rceil) T_X$	$2T_A + (3 + \lceil \log_2 (d-1) \rceil) T_X$	parallel
DLGM$_P$	$T_A + (\lceil \log_2 (d-r+1) \rceil + \lceil \log_2 T \rceil) T_X$	$T_A + (1 + \lceil \log_2 (d-r+1) \rceil) T_X$	parallel
Bit-serial structures ($d = 1$)			
[28]	$T_A + (1 + \lceil \log_2 T \rceil) T_X$	$T_A + 2T_X$	parallel
[32]	-	$T_A + 2T_X$	parallel
Bit-parallel structures ($d = m$)			
[13]	$T_A + \lceil \log_2 C_N \rceil T_X$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	parallel
[12], [33]	-	$T_A + (1 + \lceil \log_2 m \rceil) T_X$	parallel

Note that $2m - 1 \leq C_N \leq Tm - T + 1$.

reported in [9]. Also, the AEDS/XEDS (AND/XOR efficient digit-serial) multipliers provide the outputs in serial forms [22], whereas the digit-level architectures of d-SMPO$_{\text{I/II}}$ (sequential multiplier with parallel output) generate the multiplication in parallel [10]. In these tables, γ is the total number of 1s in the upper triangular matrix of $\mathbf{M}^{(0)} \vee \mathbf{M}^{(1)} \vee \cdots \vee \mathbf{M}^{(d-1)}$, where $\vee$ denotes bit-wise OR operation. It is noted that γ is a function of the normal basis and is minimum if the type II optimal normal basis is used.[3] Thus, the gate count differences between the proposed multipliers and the ones presented in [22] are minimum for type II ONBs. For this reason, the comparisons are made for general T and $T = 2$ in these tables.

As seen in Table 4, the proposed structure of DLGM$_S$ has the fewest number of total gates among the digit-level multipliers with serial output. Also, it has the fewest number of XOR gates as compared with its counterparts. For type II ONBs, it has fewer number of AND gates with the same number of XOR gates as the ones of the XEDS multiplier. As seen from Table 5, it also has the same critical path delay as the ones with serial output.

It is seen from Table 5 that the architecture of DLGM$_P$ is the fastest multiplier among all multipliers with serial and parallel output. Also, it has the same number of total gates as its counterparts with parallel output, i.e., d-SMPO$_{\text{I/II}}$.

It is interesting to compare the time complexity of the proposed digit-level multipliers with the ones using polynomial basis (PB) [31]. In [31], two word-level multipliers, namely, LSD-first and MSD-first, are proposed. Those PB multipliers generate the product in parallel after $\lceil \frac{m}{d} \rceil + 1$, $d \geq 2$, clock cycles. Therefore, one can see that the proposed DLGM$_P$ requires one fewer clock cycle than the ones proposed in [31]. Moreover, one can compare the critical path delay of the DLGM$_P$ with the ones of the LSD-first and MSD-first multipliers, i.e., $T_A + \lceil \log_2 (d+1) \rceil T_X$ and $T_A + \lceil \log_2 (2d+1) \rceil T_X$, respectively. In terms of gate counts, no comparison can be made since the gate counts of the LSD-first and MSD-first multipliers are not presented in [31].

One important feature of the proposed digit-level structures is that they can be easily scaled down to bit-serial type ($d = 1$) or up to bit-parallel type ($d = m$) and the resultant multipliers will still each have the best time delay and gate counts in the respective categories. This is also shown in Tables 4 and 5.

For bit-serial structures ($d = 1$, $r = 0$), the complexities of the DLGM$_S$ and the DLGM$_P$ match the complexities of the best multipliers available in the open literature, i.e., [9] and [28], [21], respectively. Also, for type II optimal normal bases, the complexities of the DLGM$_P$ match the complexities of the multiplier presented in [32].

For bit-parallel structures ($d = m$, $r = 0$), the complexities of the DLGM$_S$ and the DLGM$_P$ match the complexities of the one proposed in [13]. Moreover, for type II optimal normal bases, they also match the ones of the multipliers proposed in [12] and [33].

7 Conclusions

In this paper, we have proposed new $GF(2^m)$ multiplication algorithms and digit-level multiplier architectures using type T Gaussian normal bases, when m is odd. Also, the complexities of their implementations in both software and hardware have been considered. The proposed software algorithm outperforms the existing normal basis multiplication algorithms in both speed and memory combined. However, it is still slower than the best software algorithm for multiplication using polynomial bases. The proposed digit-level multiplier with serial output requires the fewest number of total gates and the one with parallel output is the

3. Type I optimal normal bases cannot be considered in this paper because m is always even for such cases.

fastest multiplier as compared with their digit-level counterparts. Moreover, the complexities of both architectures match the best ones available in the open literature when they are used as the bit-serial and bit-parallel multipliers.

Acknowledgments

The author would like to thank Professor Anwar Hasan for his invaluable comments and suggestions and Ray Zhao for his help with implementations of the algorithms. The author would also like to thank the reviewers for their comments. This work has been supported in part by an NSERC Discovery grant awarded to the author. A preliminary version of this article can be found in [21].

References

[1] IEEE Standard 1363-2000, "IEEE Standard Specifications for Public-Key Cryptography," Jan. 2000.
[2] Nat'l Inst. of Standards and Technology, *Digital Signature Standard,* FIPS Publication 186-2, Jan. 2000.
[3] J.L. Massey and J.K. Omura, "Computational Method and Apparatus for Finite Field Arithmetic," US Patent No. 4,587,627, 1986.
[4] M. Feng, "A VLSI Architecture for Fast Inversion in $GF(2^m)$," *IEEE Trans. Computers,* vol. 38, no. 10, pp. 1383-1386, Oct. 1989.
[5] T. Beth and D. Gollman, "Algorithm Engineering for Public Key Algorithms," *IEEE J. Selected Areas in Comm.,* vol. 7, no. 4, pp. 458-465, May 1989.
[6] W. Geiselmann and D. Gollmann, "Symmetry and Duality in Normal Basis Multiplication," *Proc. Sixth Symp. Applied Algebra, Algebraic Algorithms, and Error Correcting Codes (AAECC-6),* pp. 230-238, July 1988.
[7] G.B. Agnew, R.C. Mullin, I.M. Onyszchuk, and S.A. Vanstone, "An Implementation for a Fast Public-Key Cryptosystem," *J. Cryptology,* vol. 3, pp. 63-79, 1991.
[8] R.C. Mullin, "Multiple Bit Multiplier," US Patent No. 5,787,028, July 1998.
[9] L. Gao and G.E. Sobelman, "Improved VLSI Designs for Multiplication and Inversion in $GF(2^M)$ over Normal Bases," *Proc. 13th Ann. IEEE Int'l ASIC/SOC Conf.,* pp. 97-101, 2000.
[10] A. Reyhani-Masoleh and M.A. Hasan, "Low Complexity Word-Level Sequential Normal Basis Multipliers," *IEEE Trans. Computers,* vol. 54, no. 2, pp. 98-110, Feb. 2005.
[11] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, and I.S. Reed, "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$," *IEEE Trans. Computers,* vol. 34, no. 8, pp. 709-716, Aug. 1985.
[12] B. Sunar and C.K. Koç, "An Efficient Optimal Normal Basis Type II Multiplier," *IEEE Trans. Computers,* vol. 50, no. 1, pp. 83-88, Jan. 2001.
[13] A. Reyhani-Masoleh and M.A. Hasan, "A New Construction of Massey-Omura Parallel Multiplier over $GF(2^m)$," *IEEE Trans. Computers,* vol. 51, no. 5, pp. 511-520, May 2002.
[14] P. Ning and Y.L. Yin, "Efficient Software Implementation for Finite Field Multiplication in Normal Basis," *Proc. Int'l Conf. Information and Comm. Security (ICICS 2001),* pp. 177-181, Nov. 2001.
[15] Y.L. Yin and P. Ning, "Efficient Finite Field Multiplication in Normal Basis," US Patent No. 6,389,442, May 2002. Assignee: RSA Security Inc.
[16] A. Reyhani-Masoleh and M.A. Hasan, "Fast Normal Basis Multiplication Using General Purpose Processors," *IEEE Trans. Computers,* vol. 52, no. 11, pp. 1379-1390, Nov. 2003.
[17] H. Fan and Y. Dai, "Two Software Normal Basis Multiplication Algorithms for $GF(2^n)$," Report 2004/126, Cryptology ePrint Archive, 2004, http://eprint.iacr.org/.
[18] R. Dahab, D. Hankerson, F. Hu, M. Long, J. López, and A. Menezes, "Software Multiplication Using Normal Bases," Technical Report CACR 2004-12, Centre for Applied Cryptographic Research, Univ. of Waterloo, Canada, Dec. 2004, *IEEE Trans. Computers,* to appear.
[19] D.W. Ash, I.F. Blake, and S.A. Vanstone, "Low Complexity Normal Bases," *Discrete Applied Math.,* vol. 25, pp. 191-210, 1989.
[20] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications.* Cambridge Univ. Press, 1994.
[21] A. Reyhani-Masoleh, "Efficient Algorithms and Architectures for Field Multiplication Using Gaussian Normal Bases," Technical Report CACR 2004-04, Centre for Applied Cryptographic Research, Univ. of Waterloo, Canada, July 2004.
[22] A. Reyhani-Masoleh and M.A. Hasan, "Efficient Digit-Serial Normal Basis Multipliers over $GF(2^m)$," *ACM Trans. Embedded Computing Systems (TECS),* special issue on embedded systems and security, vol. 3, no. 3, pp. 575-592, Aug. 2004.
[23] R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, and R.M. Wilson, "Optimal Normal Bases in $GF(p^n)$," *Discrete Applied Math.,* vol. 22, pp. 149-161, 1988/1989.
[24] D. Johnson, A. Menezes, and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)," *Int'l J. Information Security,* vol. 1, pp. 36-63, 2001.
[25] C.C. Wang, "An Algorithm to Design Finite Field Multipliers Using a Self-Dual Normal Basis," *IEEE Trans. Computers,* vol. 38, no. 10, pp. 1457-1460, Oct. 1989.
[26] S. Gao, J. Gathen, D. Panario, and V. Shoup, "Algorithms for Exponentiation in Finite Fields," *J. Symbolic Computation,* vol. 29, pp. 879-889, 2000.
[27] M. Rosing, *Implementing Elliptic Curve Cryptography.* Manning Publications Company, 1999.
[28] S. Kwon, K. Gaj, C.H. Kim, and C.P. Hong, "Efficient Linear Array for Multiplication in $GF(2^m)$ Using a Normal Basis for Elliptic Curve Cryptography," *Proc. Workshop Cryptographic Hardware and Embedded Systems (CHES 2004),* pp. 76-91, Aug. 2004.
[29] D. Hankerson, J. Hernandez, and A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields," *Proc. Workshop Cryptographic Hardware and Embedded Systems (CHES 2000),* pp. 1-24, 2000.
[30] C.-C. Lu, "A Search of Minimal Key Functions for Normal Basis Multipliers," *IEEE Trans. Computers,* vol. 46, no. 5, pp. 588-592, May 1997.
[31] L. Song and K.K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers," *J. VLSI Signal Processing,* vol. 19, pp. 149-166, 1998.
[32] D.J. Yang, C.H. Kim, Y. Park, Y. Kim, and J. Lim, "Modified Sequential Normal Basis Multipliers for Type II Optimal Normal Bases," *Proc. Int'l Conf. Computation Science and Its Applications (ICCSA 2005),* pp. 647-656, May 2005.
[33] M. Elia and M. Leone, "On the Inherent Space Complexity of Fast Parallel Multipliers for $GF(2^m)$," *IEEE Trans. Computers,* vol. 51, no. 3, pp. 346-351, Mar. 2002.

Arash Reyhani-Masoleh received the BSc degree from Iran University of Science and Technology in 1989, the MSc degree from the University of Tehran in 1991, both with the first rank in electrical and electronic engineering, and the PhD degree in electrical and computer engineering from the University of Waterloo in 2001. From 1991 to 1997, he was with the Department of Electrical Engineering, Iran University of Science and Technology. From June 2001 to September 2004, he was with the Centre for Applied Cryptographic Research, University of Waterloo. In October 2004, he joined the Department of Electrical and Computer Engineering, University of Western Ontario, London, Ontario, Canada, as an assistant professor. His current research interests include algorithms and VLSI architectures for computations in finite fields, fault-tolerant computing, and error-control coding. Dr. Reyhani-Masoleh was awarded an NSERC (Natural Sciences and Engineering Research Council of Canada) postdoctoral fellowship in 2002. He is a member of the IEEE and the IEEE Computer Society.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.