Scalable Architecture for Dual Basis Multiplication over GF(2^m)

Liang-Hwa Chen
Department of Computer Information and Network Engineering, Lunghwa University of Science and Technology
Email: whallis2000@mail.lhu.edu.tw

Po-Lun Chang
Department of Electrical Engineering, Lunghwa University of Science and Technology
Email: whc1223@ms7.hinet.net

Chiou-Yng Lee
Department of Computer Information and Network Engineering, Lunghwa University of Science and Technology
Email: PP010@mail.lhu.edu.tw

***Abstract*—A novel low-complexity scalable and systolic dual basis multiplier over GF(2^m) is proposed in this paper. It is derived by utilizing the block Hankel matrix-vector representation and is suitable for finite fields generated by irreducible trinomials. The proposed scalable architecture can achieve good trade-off between throughout performance and hardware complexity for implementing cryptographic schemes in a constrained environment such as embedded systems by choosing appropriate digit size *d*. Analytical results reveal that the proposed scalable architecture has lower space complexity as compared to non-scalable architectures. Furthermore, the proposed architecture has the features of regularity, modularity and concurrency, and is well suitable for VLSI implementations.**

***Index Terms*—Finite field, Galois field, Cryptography, Dual basis, Hankel matrix-vector, Scalable multiplier**

I. INTRODUCTION

Arithmetic operations in finite (Galois) field GF(2^m) have received much attention in recent years because of their importance and practical applications in the areas of error-correcting codes and cryptography [1-3]. Among these operations, multiplication is the most important and time-consuming computation. Other complex arithmetic operations such as exponentiation, division and multiplicative inversion could be performed by repeating multiplications. Hence, there is demand for efficient design and implementation of the finite field multiplier with low complexity.

For finite field GF(2^m), there are three popular bases, termed polynomial basis, normal basis and dual basis to represent its elements. Each representation has its own distinct advantage. The polynomial basis multiplier does not require basis conversion and has regularity and simplicity feature. The normal basis multiplier is quite effective in performing the squaring of an element in finite field. The dual basis multiplier needs the least number of gates that leads to the smallest chip area demand for VLSI implementation [4]. In past years, most finite field multipliers on these bases proposed in the literature were generally classified as bit-serial [5,6] and bit-parallel architectures [7-10]. For cryptography applications which heavily rely on large word length of operands, the bit-serial architecture requires less chip area, but is too slow, while the bit-parallel architecture are typically faster, but is more complex and requires more chip area and power consumption. In order to enhance the trade-off between throughout performance and hardware complexity, hybrid multipliers for composite fields GF((2^m)^k) [11] and digit-serial architectures [12-14] were presented. These architectures are based on a cut-set systolization technique to speed up computation process. However, such multipliers have a similar space complexity as compared to the original bit-level multiplier designs.

Another architecture, called scalable architecture [15,16], is a combination of serial and parallel schemes. Each *m*-bit data word is separated into $k = \lceil m/d \rceil$ *d*-bit sub-words (also termed digits) where the selected digit size *d* is the scalable factor. The computation of two digits is performed with a parallel scheme while the computation of two data words is performed in digits with a serial scheme. Hence, considering the trade-off between throughput performance and hardware complexity, the scalable architecture can generate an optimal realization in hardware implementations. Besides, it has the advantage of

flexibility in re-usage. Suppose there has been a multiplier designed for 768-bit data words. It cannot be directly applied to a system whose data word length is 1024 bits. The non-scalable (bit-parallel) multiplier has to be re-designed to match the system. Conversely, with the scalable architecture, it does not need to change the core multiplier. By only adjusting the register array numbers to match the required longer word length and reusing the core multiplier, the scalable multiplier can then be applied to the system. In this paper, a novel scalable dual basis multiplication algorithm over GF(2^m) is proposed. We utilize the block Hankel matrix-vector representation to derive the proposed algorithm from which a low-complexity scalable and systolic multiplier is then derived. The proposed scalable multiplier is suitable for the finite fields GF(2^m) generated by irreducible trinomials. Analytical results reveal that the proposed scalable multiplier has lower space complexity as compared to traditional digit-serial and bit-parallel multipliers.

The rest of this paper is organized as follows: Section II briefly reviews the dual basis multiplication over GF(2^m) and a bit-parallel dual basis multiplication algorithm. Section III then presents the proposed novel scalable and systolic algorithm and architecture for dual basis multiplication over GF(2^m). Its time and space complexities are discussed in Section IV. Conclusions are finally drawn in Section V.

II. PRELIMINARIES

It is commonly known that the finite (Galois) field GF(2^m) can be viewed as a vector space of dimension m over GF(2), where the field is generated by the irreducible polynomial $F(x) = f_0 + f_1 x + \ldots + f_{m-1}x^{m-1} + x^m$ of degree m over GF(2). Suppose that α is a root of the irreducible polynomial $F(x)$. Then, any element A in the finite field GF(2^m) can be represented as $A = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1}$, where the coordinates $a_i \in$ GF(2) for $0 \le i \le m-1$ and the set $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ is called the polynomial basis (PB) of GF(2^m).

Definition 1. The trace function $Tr(x)$ over GF(2^m) is defined as [5]

$$Tr(x) = \sum_{i=0}^{m-1} x^{2^i}.$$ □

Definition 2. A basis $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ in GF(2^m) is said to be the dual basis (DB) of $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ if the following condition is satisfied:

$$Tr(\gamma\alpha^i\beta_j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \ne j, \end{cases} \quad (1)$$

where γ is chosen so as to simplify the conversion between polynomial and dual bases. □

For any element $A = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1}$ in GF(2^m), its dual basis representation can be expressed as $A = Tr(\gamma A)\beta_0 + Tr(\gamma\alpha A)\beta_1 + \cdots + Tr(\gamma\alpha^{m-1}A)\beta_{m-1}$. For any two elements A and B in GF(2^m) represented in polynomial and dual basis respectively, i.e., $A = \sum_{i=0}^{m-1} a_i\alpha^i$, $B = \sum_{i=0}^{m-1} b_i\beta_i$, their product $C = AB$ represented in dual basis, i.e., $C = \sum_{i=0}^{m-1} c_i\beta_i$, can be computed with the following discrete-time Wiener-Hopf equation (DTWHE) [17]:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} b_0 & b_1 & \cdots & b_{m-1} \\ b_1 & b_2 & \cdots & b_m \\ \vdots & \vdots & \ddots & \vdots \\ b_{m-1} & b_m & \cdots & b_{2m-2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{bmatrix} \quad (2)$$

where

$$b_{m+i} = \sum_{j=0}^{m-1} f_j b_{i+j} = f_0 b_i + f_1 b_{i+1} + \cdots + f_{m-1} b_{i+m-1}, \quad \text{for } i = 0, 1, \cdots, m-1. \quad (3)$$

It is derived as follows: First, from Definition 2, the coordinates b_i of B can be obtained as $b_i = Tr(\gamma\alpha^i B)$ for $0 \le i \le m-1$. Next, since

$F(\alpha)=0$, thus,

$$\alpha^m=\sum_{i=0}^{m-1}f_i\alpha^i=f_0+f_1\alpha+f_2\alpha^2+\cdots+f_{m-1}\alpha^{m-1}, \quad (4)$$

$$\alpha^{m+i}=\sum_{j=0}^{m-1}f_j\alpha^{i+j} =f_0\alpha^i+f_1\alpha^{i+1}+f_2\alpha^{i+2}+\cdots+f_{m-1}\alpha^{i+m-1} \quad (5)$$

Let us define that $b_{m+i}=Tr(\gamma\alpha^{m+i}B)$, then, according to (5),

$$b_{m+i}\triangleq Tr(\gamma\alpha^{m+i}B)=Tr\left(\gamma(\sum_{j=0}^{m-1}f_j\alpha^{i+j})B\right) =\sum_{j=0}^{m-1}f_jTr(\gamma\alpha^{i+j}B) =f_0b_i+f_1b_{i+1}+f_2b_{i+2}+\cdots+f_{m-1}b_{i+m-1} \quad (6)$$

for $i=0,\ 1,\cdots,m-1$.

From Definition 2, the coordinates c_i of C can also be obtained as $c_i=Tr(\gamma\alpha^iC)$. With the fact that $C=AB$, $A=\sum_{i=0}^{m-1}a_i\alpha^i$, and according to (6), we get

$$c_i=Tr(\gamma\alpha^iC)=Tr(\gamma\alpha^iAB) =Tr\left(\gamma\alpha^i(\sum_{j=0}^{m-1}a_j\alpha^j)B\right)=\sum_{j=0}^{m-1}a_jTr(\gamma\alpha^{i+j}B) =a_0b_i+a_1b_{i+1}+a_2b_{i+2}+\cdots a_{m-1}b_{i+m-1}, \quad (7)$$

for $i=0,\ 1,\cdots,m-1$.

Express (7) as matrix form, the DTWHE in (2) is then obtained. Besides, if we define the following vectors: $\boldsymbol{A}=[a_0,a_1,\cdots,a_{m-1}]$, $\boldsymbol{F}=[f_0,f_1,\cdots,f_{m-1}]$ and

$$\boldsymbol{B}^{(i)}=[b_i,b_{i+1},\cdots,b_{i+m-1}],\ \text{for } i=0,\ 1,\cdots,m-1, \quad (8)$$

then, (6) and (7) can also be expressed as

$$b_{m+i}=\boldsymbol{B}^{(i)}\odot\boldsymbol{F} \quad (9)$$

$$c_i=\boldsymbol{B}^{(i)}\odot\boldsymbol{A} \quad (10)$$

where "$\odot$" denotes the inner product operation of two vectors. Note that $\boldsymbol{B}^{(0)}=\boldsymbol{B}=[b_0,b_1,\cdots,b_{m-1}]$. Applying (9) and (10), the DB multiplication can be carried out by the following algorithm.

Algorithm 1: [7]

Input: $\boldsymbol{A}=[a_0,a_1,\cdots,a_{m-1}]$, $\boldsymbol{B}=[b_0,b_1,\cdots,b_{m-1}]$ and $\boldsymbol{F}=[f_0,f_1,\cdots,f_{m-1}]$

Output: $\boldsymbol{C}=[c_0,c_1,\cdots,c_{m-1}]=\boldsymbol{AB}$

1. Initial step
   1.1 $\boldsymbol{C}=[0,0,\cdots,0]$
   1.2 $\boldsymbol{B}^{(0)}=\boldsymbol{B}$
2. Multiplication step
   2.1 For $i=0$ to $m-1$ do
   2.2 $b_{m+i}=\boldsymbol{B}^{(i)}\odot\boldsymbol{F}$
   2.3 $c_i=\boldsymbol{B}^{(i)}\odot\boldsymbol{A}$
   2.4 $\boldsymbol{B}^{(i+1)}=\boldsymbol{B}^{(i)}\ll 1+[0,\cdots,0,b_{m+i}]$
   2.5 Endfor
3. Return $\boldsymbol{C}$.

According to the above algorithm, Lee, et al. [7] proposed a bit-parallel systolic DB multiplier consisting of m^2 cell which consists of one AND gate, one XOR gate and two 1-bit latches. Due to the regularity of its architecture, this DB multiplier is suitable for VLSI implementation. However, for large field size of binary finite fields, such as GF(2^{233}) in ECDSA (elliptic curve digital signature algorithm) recommended by NIST (National Institute for Standards and Technology) [18], the corresponding large space complexity $O(m^2)$ makes such kind of multiplier inappropriate for implementing in constrained hardware environments such as smart cards and mobile handsets. To overcome this problem, we propose in the next section a scalable scheme for the DB multiplication that divides *m*-bit word into several *d*-bit digits and then iteratively applies a smaller scale multiplier to get the complete *m*-bit multiplication.

III. Proposed Scalable Systolic Dual Basis Multiplier over GF(2^m)

To derive the scalable architecture of DB multiplier, we need first to introduce the Hankel matrix-vector representation.

A. Hankel matrix-vector representation

Definition 3. An $m\times m$ matrix $\mathbf{H}$ is called a Hankel matrix if it satisfies the relation $\mathbf{H}(p,q)=\mathbf{H}(p+1,q-1)$, for $0\le p\le m-2$, $1\le q\le m-1$, where $\mathbf{H}(p,q)$ represents the

element in the intersection of row i and column j. □

A Hankel matrix can be entirely determined by the $2m-1$ entries that locate on its first row and last column. That is, it can be defined by the corresponding Hankel vector $\overline{\boldsymbol{H}} = [h_0, h_1, \cdots, h_{2m-2}]$. With the Hankel matrix-vector representation, the product of a Hankel matrix **H** and a vector $\boldsymbol{V} = [v_0, v_1, \cdots v_{m-1}]$, i.e., $\mathbf{H}\boldsymbol{V}$, is denoted as $\overline{\boldsymbol{H}} \otimes \boldsymbol{V}$. With such notation, the DB multiplication in (2) can be expressed as

$$\boldsymbol{C} = \overline{\boldsymbol{B}} \otimes \boldsymbol{A}, \tag{11}$$

where $\overline{\boldsymbol{B}} = [b_0, b_1, \cdots, b_{m-1}, b_m, \cdots, b_{2m-2}]$ is the corresponding Hankel vector of the matrix in (2).

B. Algorithm

For digit size chosen as d-bits, and $k = \lceil m/d \rceil$, Eq. (2) can also be expressed as the following block Hankel matrix-vector form:

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}_0 \\ \boldsymbol{C}_1 \\ \vdots \\ \boldsymbol{C}_{k-1} \end{bmatrix} = \mathbf{B}\boldsymbol{A} = \begin{bmatrix} \mathbf{B}_0 & \mathbf{B}_1 & \cdots & \mathbf{B}_{k-1} \\ \mathbf{B}_1 & \mathbf{B}_2 & \cdots & \mathbf{B}_k \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}_{k-1} & \mathbf{B}_k & \cdots & \mathbf{B}_{2k-2} \end{bmatrix} \begin{bmatrix} \boldsymbol{A}_0 \\ \boldsymbol{A}_1 \\ \vdots \\ \boldsymbol{A}_{k-1} \end{bmatrix} \tag{12}$$

where

$$\boldsymbol{C}_i = [c_{id}, c_{id+1}, \cdots, c_{id+(d-1)}], \text{ for } 0 \le i \le k-1, \tag{13}$$

$$\boldsymbol{A}_i = [a_{id}, a_{id+1}, \cdots, a_{id+(d-1)}], \text{ for } 0 \le i \le k-1, \tag{14}$$

are all $d \times 1$ vectors and

$$\mathbf{B}_i = \begin{bmatrix} b_{id} & b_{id+1} & \cdots & b_{id+(d-1)} \\ b_{id+1} & b_{id+2} & \cdots & b_{id+d} \\ \vdots & \vdots & \ddots & \vdots \\ b_{id+(d-1)} & b_{id+d} & \cdots & b_{id+(2d-2)} \end{bmatrix}, \quad \text{for } 0 \le i \le 2k-2, \tag{15}$$

are all $d \times d$ Hankel matrices and their corresponding Hankel vectors are

$$\overline{\boldsymbol{B}}_i = [b_{id}, b_{id+1}, \cdots, b_{id+(d-1)}, b_{id+d}, \cdots, b_{id+(2d-2)}], \quad \text{for } 0 \le i \le 2k-2. \tag{16}$$

With the Hankel matrix-vector representation, we can then get the following equations from (12) :

$$\begin{aligned} \boldsymbol{C}_i &= \overline{\boldsymbol{B}}_i \otimes \boldsymbol{A}_0 + \overline{\boldsymbol{B}}_{i+1} \otimes \boldsymbol{A}_1 + \cdots + \overline{\boldsymbol{B}}_{i+k-1} \otimes \boldsymbol{A}_{k-1} \\ &= \sum_{j=0}^{k-1} \overline{\boldsymbol{B}}_{i+j} \otimes \boldsymbol{A}_j, \qquad \text{for } 0 \le i \le k-1. \end{aligned} \tag{17}$$

Here, the Hankel vectors $\overline{\boldsymbol{B}}_0, \overline{\boldsymbol{B}}_1, \cdots, \overline{\boldsymbol{B}}_{k-2}$ consist of $b_0, b_1, \cdots, b_{m-2}$ which can be directly picked up from the original input vector $\boldsymbol{B}$. The remaining Hankel vectors $\overline{\boldsymbol{B}}_{k-1}, \overline{\boldsymbol{B}}_k, \cdots, \overline{\boldsymbol{B}}_{2k-2}$ consist of $b_{m-d}, b_{m-d+1}, \cdots, b_{2m-2}$ where $b_m, b_{m+1}, \cdots, b_{2m-2}$ have to be computed out from $b_0, b_1, \cdots, b_{m-1}$ and depend on the generating function $F(x)$. From (17), it shows that each digit of the product word $\boldsymbol{C}$, i.e., $\boldsymbol{C}_i$, can be obtained with the summation of the k Hankel matrix-vector multiplications, i.e., $\overline{\boldsymbol{B}}_{i+j} \otimes \boldsymbol{A}_j$, for $0 \le j \le k-1$.

To compute $\boldsymbol{C}_0$, according to (17), the Hankel vectors $\overline{\boldsymbol{B}}_0, \overline{\boldsymbol{B}}_1, \cdots, \overline{\boldsymbol{B}}_{k-1}$ are required. They can be generated from the vector $\overline{\boldsymbol{B}}^{(0)} = [b_0, b_1, \cdots, b_{m-1}, b_m, \cdots, b_{m+d-1}]$ whose former part of coordinates, $b_0, b_1, \cdots, b_{m-1}$, are exactly those of the original input vector $\boldsymbol{B}$. That is, $b_i^{(0)} = b_i$, for $0 \le i \le m-1$. But its latter part of coordinates, $b_m, b_{m+1}, \cdots, b_{m+d-1}$, i.e., $b_i^{(0)}$ for $m \le i \le m+d-1$, have to be derived from its former part of coordinates. When the generating function is an irreducible trinomial, i.e., $F(x) = x^m + x^n + 1$, the values of $b_m, b_{m+1}, \cdots, b_{m+d-2}, b_{m+d-1}$ can be pre-computed simultaneously as follows: Let α be the root of $F(x)$, then $\alpha^m = \alpha^n + 1$. Because b_{m+i} is defined as $Tr(\gamma\alpha^{m+i}\boldsymbol{B})$, thus,

$$\begin{aligned} b_{m+i} &= Tr(\gamma\alpha^{m+i}\boldsymbol{B}) = Tr\left(\gamma(\alpha^n + 1)\alpha^i\boldsymbol{B}\right) \\ &= Tr(\gamma\alpha^{n+i}\boldsymbol{B}) + Tr(\gamma\alpha^i\boldsymbol{B}) = b_{n+i} + b_i, \\ &\qquad \text{for } 0 \le i \le d-1. \end{aligned} \tag{18}$$

When n and d are chosen as smaller than $m/2$, i.e., $0 < n, d \le \lceil m/2 \rceil$, then $n + d - 1 \le m - 1$. By using (18), we obtain that $[b_m, b_{m+1}, \cdots, b_{m+d-1}] = [(b_n + b_0), (b_{n+1} + b_1), \cdots, (b_{n+d-1} + b_{d-1})]$. That is, they can be pre-computed simultaneously from $b_0, b_1, \cdots, b_{m-1}$ with d XOR gates.

To compute $\boldsymbol{C}_i$, $1 \le i \le k-1$, we need Hankel vectors $\overline{\boldsymbol{B}}_i, \overline{\boldsymbol{B}}_{i+1}, \cdots, \overline{\boldsymbol{B}}_{i+k-1}$ which can be generated from the vector $[b_{id}, b_{id+1}, \cdots, b_{id+m-1}, b_{id+m}, \cdots, b_{id+m+d-1}]$ which is defined as $\overline{\boldsymbol{B}}^{(i)}$. That is,

$$\begin{aligned}\overline{\boldsymbol{B}}^{(i)} &= [b_0^{(i)}, b_1^{(i)}, \cdots, b_{m-1}^{(i)},\ b_m^{(i)}, b_{m+1}^{(i)}, \cdots, b_{m+d-1}^{(i)}] \\ &\triangleq [b_{id}, b_{id+1}, \cdots, b_{id+m-1},\ b_{id+m}, \cdots, b_{id+m+d-1}].\end{aligned} \tag{19}$$

$\overline{\boldsymbol{B}}^{(i)}$ can be obtained by the operation of $\alpha^d \overline{\boldsymbol{B}}^{(i-1)}$ because the j-th coordinate of $\alpha^d \overline{\boldsymbol{B}}^{(i-1)}$ is

$$\begin{aligned}&Tr(\gamma\alpha^j \alpha^d \boldsymbol{B}^{(i-1)}) = Tr(\gamma\alpha^{j+d}\boldsymbol{B}^{(i-1)}) \\ &= b_{(i-1)d+(j+d)} = b_{id+j},\ \text{for } 0 \le j \le m+d-1\end{aligned} \tag{20}$$

which is exactly the j-th coordinate, $b_j^{(i)}$, of $\overline{\boldsymbol{B}}^{(i)}$. The operation of $\alpha^d \overline{\boldsymbol{B}}^{(i-1)}$ can be divided into two parts: For the former part of $\overline{\boldsymbol{B}}^{(i)}$, i.e., $[b_0^{(i)}, b_1^{(i)}, \cdots, b_{m-1}^{(i)}] = [b_{id}, b_{id+1}, \cdots, b_{id+m-1}]$, it is directly obtained by a d-digit left-shifting operation on $\overline{\boldsymbol{B}}^{(i-1)}$ because

$$\begin{aligned}&[b_0^{(i)}, b_1^{(i)}, \cdots, b_{m-1}^{(i)}] = [b_{id}, b_{id+1}, \cdots, b_{id+m-1}] \\ &= [b_{(i-1)d+d}, b_{(i-1)d+d+1}, \cdots, b_{(i-1)d+d+m-1}] \\ &= [b_d^{(i-1)}, b_{d+1}^{(i-1)}, \cdots, b_{m+d-1}^{(i-1)}].\end{aligned} \tag{21}$$

For the latter part, i.e., $[b_m^{(i)}, b_{m+1}^{(i)}, \cdots, b_{m+d-1}^{(i)}] = [b_{id+m}, b_{id+m+1}, \cdots, b_{id+m+d-1}]$, it is computed from the coordinates of $\overline{\boldsymbol{B}}^{(i-1)}$ as follows:

$$\begin{aligned}b_{m+j}^{(i)} &= b_{id+m+j} = Tr(\gamma\alpha^{id+m+j}B) \\ &= Tr\left(\gamma(\alpha^n+1)\alpha^{id+j}B\right) \\ &= Tr(\gamma\alpha^{n+id+j}B) + Tr(\gamma\alpha^{id+j}B) \\ &= b_{(i-1)d+d+n+j} + b_{(i-1)d+d+j} \\ &= b_{n+d+j}^{(i-1)} + b_{d+j}^{(i-1)}, \quad \text{for } 0 \le j \le d-1.\end{aligned} \tag{22}$$

When n and d are chosen as smaller than $m/2$, i.e., $0 < n, d \le \lceil m/2 \rceil$, then $n+2d-1 \le m+d-1$. According to (22), we obtain that

$$\begin{aligned}&[b_m^{(i)}, b_{m+1}^{(i)}, \cdots, b_{m+d-1}^{(i)}] \\ &= [(b_{n+d}^{(i-1)} + b_d^{(i-1)}), (b_{n+d+1}^{(i-1)} + b_{d+1}^{(i-1)}), \cdots, (b_{n+2d-1}^{(i-1)} + b_{2d-1}^{(i-1)})].\end{aligned} \tag{23}$$

That is, the lattermost d coordinates of $\overline{\boldsymbol{B}}^{(i)}$ can be computed simultaneously from the coordinates of $\overline{\boldsymbol{B}}^{(i-1)}$ with d XOR gates.

In summary, we obtain the following equations of $\overline{\boldsymbol{B}}^{(i)}$ from the above derivation:

$$\begin{aligned}\overline{\boldsymbol{B}}^{(0)} &= [b_0^{(0)}, b_1^{(0)}, \cdots, b_{m-1}^{(0)},\ b_m^{(0)}, b_{m+1}^{(0)}, \cdots, b_{m+d-1}^{(0)}] \\ &= [b_0, b_1, \cdots, b_{m-1},\ b_m, b_{m+1}, \cdots, b_{m+d-1}] \\ &= [b_0, b_1, \cdots, b_{m-1},\ (b_n + b_0), (b_{n+1} + b_1), \\ &\quad \cdots, (b_{n+d-1} + b_{d-1})].\end{aligned} \tag{24}$$

The recursive form of $\overline{\boldsymbol{B}}^{(i)}$, for $1 \le i \le k-1$, is

$$\begin{aligned}\overline{\boldsymbol{B}}^{(i)} &= [b_0^{(i)}, b_1^{(i)}, \cdots, b_{m-1}^{(i)}, b_m^{(i)}, b_{m+1}^{(i)}, \cdots, b_{m+d-1}^{(i)}] \\ &= [b_{id}, b_{id+1}, \cdots, b_{id+m-1}, b_{id+m}, b_{id+m+1}, \cdots, b_{id+m+d-1}] \\ &= \alpha^d \overline{\boldsymbol{B}}^{(i-1)} \\ &= [b_d^{(i-1)}, b_{d+1}^{(i-1)}, \cdots, b_{d+m-1}^{(i-1)}, b_{d+m}^{(i-1)}, b_{d+m+1}^{(i-1)}, \cdots, b_{m+2d-1}^{(i-1)}] \\ &= [b_d^{(i-1)}, b_{d+1}^{(i-1)}, \cdots, b_{d+m-1}^{(i-1)}, (b_{n+d}^{(i-1)} + b_d^{(i-1)}), \\ &\qquad (b_{n+d+1}^{(i-1)} + b_{d+1}^{(i-1)}), \cdots, (b_{n+2d-1}^{(i-1)} + b_{2d-1}^{(i-1)})].\end{aligned} \tag{25}$$

Based on the above derivation, we propose a scalable dual-basis multiplication algorithm with digit size d as follows:

Algorithm 2:

Input: $\boldsymbol{A} = [a_0, a_1, \cdots, a_{m-1}]$, $\boldsymbol{B} = [b_0, b_1, \cdots, b_{m-1}]$

Output: $\boldsymbol{C} = [c_0, c_1, \cdots, c_{m-1}] = \boldsymbol{AB}$

1. Initial step:
   1.1 Clear each output sub-vector $\boldsymbol{C}_i$, $0 \le i \le k-1$, $k = \lceil m/d \rceil$.
   1.2 Build each sub-vectors $\boldsymbol{A}_i$, $0 \le i \le k-1$, from $\boldsymbol{A}$ according to (14).
   1.3 Generate $\overline{\boldsymbol{B}}^{(0)}$ from $\boldsymbol{B}$ according to (24).
2. Multiplication step:
   2.1 For $i = 0$ to $k-1$ do
   2.2 Generate Hankel vectors $\overline{\boldsymbol{B}}_i, \overline{\boldsymbol{B}}_{i+1}, \cdots, \overline{\boldsymbol{B}}_{i+k-1}$ from $\overline{\boldsymbol{B}}^{(i)}$ according to (16) and (19).
   2.3 For $j = 0$ to $k-1$ do
   2.4 $\boldsymbol{C}_i = \boldsymbol{C}_i + \overline{\boldsymbol{B}}_{i+j} \otimes \boldsymbol{A}_j$
   2.5 Endfor
   2.6 Generate $\overline{\boldsymbol{B}}^{(i+1)} = \alpha^d \overline{\boldsymbol{B}}^{(i)}$ according to (25).
   2.7 Endfor
3. Return $\boldsymbol{C} = [\boldsymbol{C}_0, \boldsymbol{C}_1, \cdots, \boldsymbol{C}_{k-1}]$.

The PB element $\boldsymbol{A}$ is divided into k sub-vectors

$\boldsymbol{A}_i$, and the DB element $\boldsymbol{B}$ is transformed into the vector $\overline{\boldsymbol{B}}^{(0)}$ to generate the k Hankel vectors $\overline{\boldsymbol{B}}_0, \overline{\boldsymbol{B}}_1, \cdots, \overline{\boldsymbol{B}}_{k-1}$. After totally k rounds of computation are performed, the complete product output vector $\boldsymbol{C}$ is obtained. In each round, the required k Hankel vectors $\overline{\boldsymbol{B}}_i, \overline{\boldsymbol{B}}_{i+1}, \cdots, \overline{\boldsymbol{B}}_{i+k-1}$ are generated from the vector $\overline{\boldsymbol{B}}^{(i)}$ which is transformed recursively from $\overline{\boldsymbol{B}}^{(i-1)}$ (step 2.6). Then, with k times of Hankel multiplication are iteratively performed and summed up together (step 2.4), the sub-vector $\boldsymbol{C}_i$ is obtained.

C. Architecture

To derive the proposed scalable architecture, we need more equations. Let us define the following sub-vectors of d-bit length:

$$\overline{\boldsymbol{B}}_j^{(i)} = [b_{jd}^{(i)}, b_{jd+1}^{(i)}, \cdots, b_{(j+1)d-1}^{(i)}], \quad \text{for } 0 \le i \le k-1,\ 0 \le j \le k. \tag{26}$$

Then, the vector $\overline{\boldsymbol{B}}^{(i)}$ in each round can be expressed as the composition of $\overline{\boldsymbol{B}}_j^{(i)}$:

$$\overline{\boldsymbol{B}}^{(i)} = [\overline{\boldsymbol{B}}_0^{(i)}, \overline{\boldsymbol{B}}_1^{(i)}, \overline{\boldsymbol{B}}_2^{(i)} \cdots, \overline{\boldsymbol{B}}_{k-1}^{(i)}, \overline{\boldsymbol{B}}_k^{(i)}], \tag{27}$$

and the subsequently generated Hankel vectors $\overline{\boldsymbol{B}}_{i+j}$ expressed as

$$\overline{\boldsymbol{B}}_{i+j} = [\overline{\boldsymbol{B}}_j^{(i)},\ \overline{\boldsymbol{B}}_{j+1}^{(i)} \setminus b_{(j+2)d-1}^{(i)}], \quad \text{for } 0 \le j \le k-1, \tag{28}$$

where $\overline{\boldsymbol{B}}_{j+1}^{(i)} \setminus b_{(j+2)d-1}^{(i)}$ denotes removing the lattermost bit $b_{(j+2)d-1}^{(i)}$ from $\overline{\boldsymbol{B}}_{j+1}^{(i)}$. Besides, according to the recursive equation in (25), we get

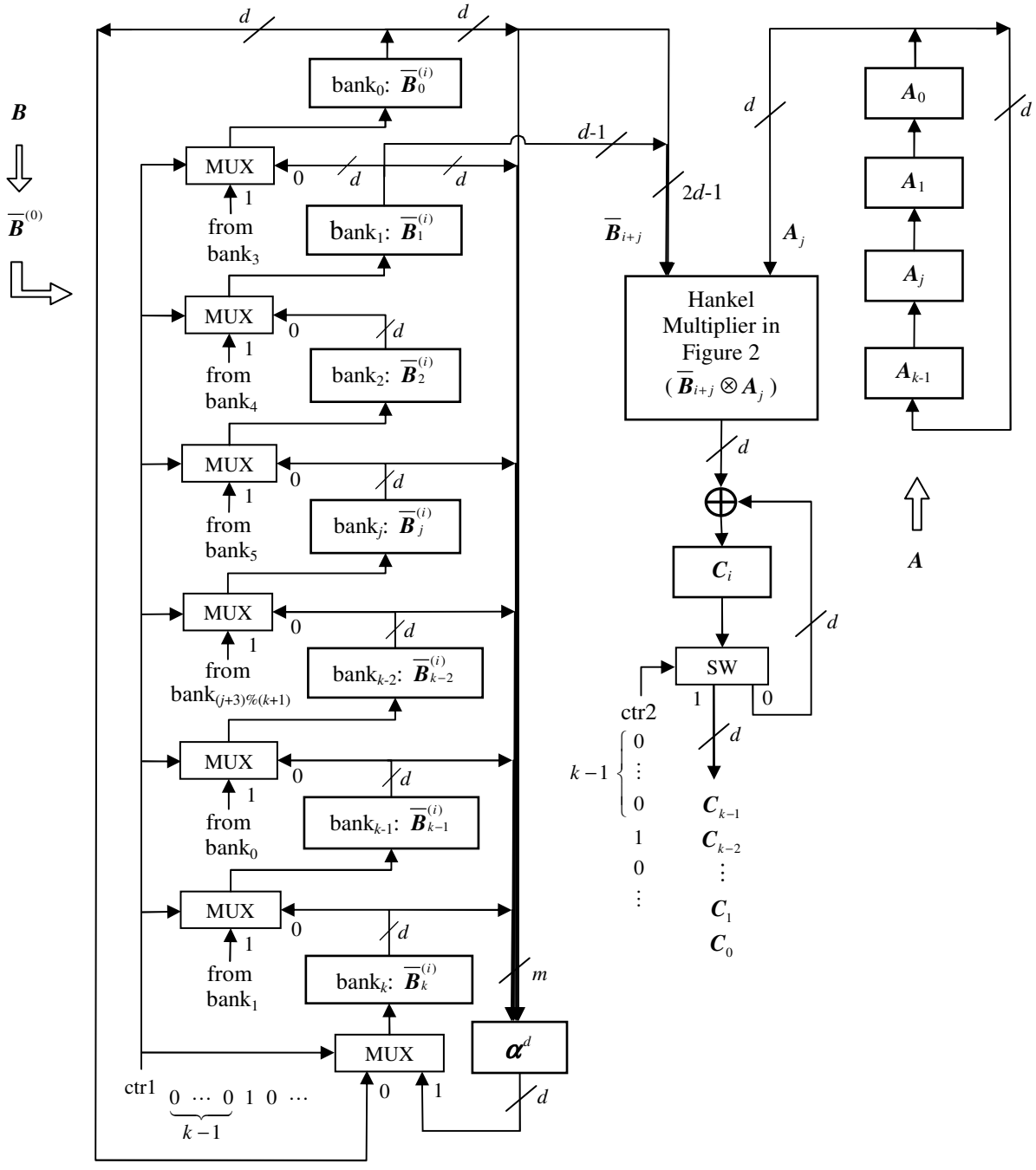
$$\overline{\boldsymbol{B}}_j^{(i)} = \overline{\boldsymbol{B}}_{j+1}^{(i-1)}, \quad \text{for } 0 \le j \le k, \tag{29}$$

and $\overline{\boldsymbol{B}}_k^{(i)} = [b_m^{(i)}, b_{m+1}^{(i)}, \cdots, b_{m+d-1}^{(i)}]$ can be calculated from $\overline{\boldsymbol{B}}_1^{(i-1)}, \overline{\boldsymbol{B}}_2^{(i-1)}, \cdots, \overline{\boldsymbol{B}}_k^{(i-1)}$ with d XOR gates.

Based on Algorithm 2 and the above equations, the proposed scalable architecture for dual-basis multiplication over GF(2^m) is illustrated in Figure 1. This architecture is majorly composed of one $d \times d$ Hankel multiplier, three registers for $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ respectively, one summation circuit ($\oplus$) for $\boldsymbol{C}$ and one recursion circuit (α^d block) for $\boldsymbol{B}$. The $d \times d$ Hankel multiplier (shown in Figure 2) is applied to perform the Hankel matrix-vector multiplication, $\overline{\boldsymbol{B}}_{i+j} \otimes \boldsymbol{A}_j$, and is composed of d^2 U-cell. Each U-ce ll (shown in Figure 3) consists of one AND gate, one XOR gate and two 1-bit latches. This systolic Hankel multiplier is similar to that one presented in [7]. The register A consists of k d-bit latches and performs as a circular-shift register. The register B is composed of $k+1$ banks which are all d-bit latches. When the control signal of the MUXs $\text{ctr1} = 0$, the register B works as a circular-shift register, and when $\text{ctr1} = 1$, it performs the recursive transformation operation $\overline{\boldsymbol{B}}^{(i)} = \alpha^d \overline{\boldsymbol{B}}^{(i-1)}$ in (25). The α^d block here is composed of d XOR gates and performs the generation of the lattermost d coordinates of $\overline{\boldsymbol{B}}^{(i)}$ from $\overline{\boldsymbol{B}}^{(i-1)}$. The register C is a d-bit latch and is responsible for accumulating and outputting the sub-vector $\boldsymbol{C}_i$ in each computation round. When the control signal of the SW $\text{ctr2} = 0$, the register C accumulates the outputs of the Hankel multiplier, and when $\text{ctr2} = 1$, the sub-vector $\boldsymbol{C}_i$ is sent out.

Initially, the register C is cleared (step 1.1). The input vector $\boldsymbol{A}$ is divided into k sub-vectors $\boldsymbol{A}_j$ and stored into register A (step 1.2). Input vector $\boldsymbol{B}$ is transformed into $\overline{\boldsymbol{B}}^{(0)}$ (step 1.3) which is divided into $k+1$ sub-vectors $\overline{\boldsymbol{B}}_j^{(0)}$ (Eq. (26)) and stored into register B. In round 0, the control signals ctr1 and ctr2 are all assigned to the value 0. Register B performs as a circular-shift register and thus $\overline{\boldsymbol{B}}_j = [\overline{\boldsymbol{B}}_j^{(0)},\ \overline{\boldsymbol{B}}_{j+1}^{(0)} \setminus b_{(j+2)d-1}^{(0)}]$, and $\boldsymbol{A}_j$, $0 \le j \le k-1$, are sequentially sent into the Hankel multiplier to perform the product operations $\overline{\boldsymbol{B}}_j \otimes \boldsymbol{A}_j$. In the meantime, register C accumulates the outputs of the Hankel multiplier and thus performs the summation operation $\boldsymbol{C}_0 = \boldsymbol{C}_0 + \overline{\boldsymbol{B}}_j \otimes \boldsymbol{A}_j$ (step 2.4). The signal ctr2 is changed to the value 1 to make the result $\boldsymbol{C}_0$ be outputted when the multiplier outputs the product of $\overline{\boldsymbol{B}}_{k-1} \otimes \boldsymbol{A}_{k-1}$. The signal ctr1 is changed to the value 1 when the data $\overline{\boldsymbol{B}}_{k-1} = [\overline{\boldsymbol{B}}_{k-1}^{(0)},\ \overline{\boldsymbol{B}}_k^{(0)} \setminus b_{m+d-1}^{(0)}]$

and A_{k-1} are about to be sent into the multiplier. At that time, the content of register B (from bank_0 to bank_k) is $[\overline{B}_{k-1}^{(0)}, \overline{B}_k^{(0)}, \overline{B}_0^{(0)}, \overline{B}_1^{(0)}, \cdots, \overline{B}_{k-3}^{(0)}, \overline{B}_{k-2}^{(0)}]$. With the effect of the MUXs when $\text{ctr1} = 1$, the contents will, at the next clock cycle, be changed to $[\overline{B}_1^{(0)}, \overline{B}_2^{(0)}, \overline{B}_3^{(0)}, \cdots, \overline{B}_k^{(0)}, \overline{B}_{k+1}^{(0)}]$ which is exactly equal to $[\overline{B}_0^{(1)}, \overline{B}_1^{(1)}, \overline{B}_2^{(1)}, \cdots, \overline{B}_{k-1}^{(1)}, \overline{B}_k^{(1)}]$ (according to (29)). Note that the content of bank_k is changed to $\overline{B}_{k+1}^{(0)} = \overline{B}_k^{(1)}$ which is the output of the α^d block.



Note: Each bank_j is a d-bit latch. "%" denotes the mod operation.

Figure 1. Proposed scalable DB multiplier over $\text{GF}(2^m)$

At the same time, the next computation round (round 1) begins. The signals ctr1 and ctr2 are changed back to 0 and the computation $\boldsymbol{C}_1 = \boldsymbol{C}_1 + \overline{\boldsymbol{B}}_{1+j} \otimes \boldsymbol{A}_j$ is then performed with same scheme as that in round 0. The remaining sub-vectors $\boldsymbol{C}_i$, $2 \le i \le k-1$ are then sequentially computed in the same manner and outputted from the register C in the remaining rounds *i*.

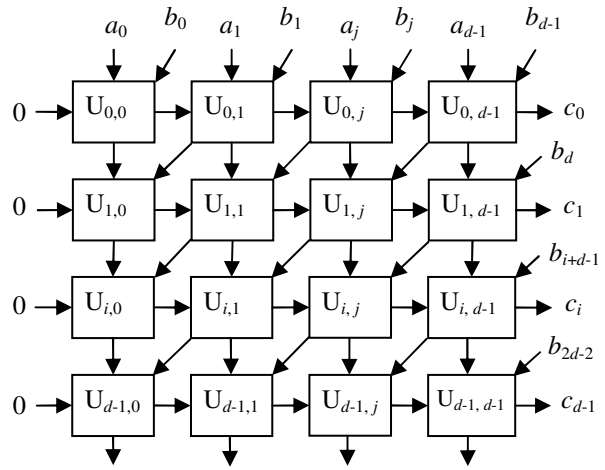
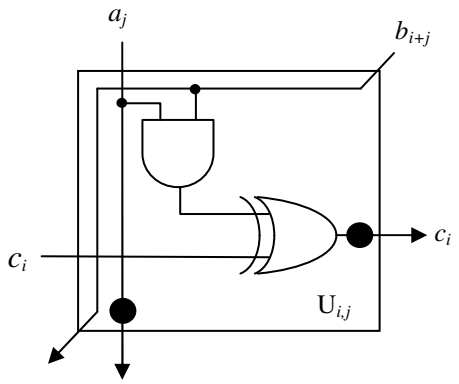


Figure 2. Systolic DB Hankel multiplier used to perform $\overline{\boldsymbol{B}}_{i+j} \otimes \boldsymbol{A}_j$



Note: Symbol ● denotes 1-bit latch

Figure 3. Detailed circuit of a U-cell

IV. Time and Space Complexity

The proposed scalable DB multiplier contains one $d \times d$ Hankel multiplier (Figure 2) which consists of d^2 U-cells. Each U-cell (Figure 3) comprises one AND gate, one XOR gate and two 1-bit latches. Thus, d^2 AND gates are required. Besides, the summation circuit ($\oplus$) for $\boldsymbol{C}$ and the α^d block all consist of d XOR gates. Thus, $d^2 + 2d$ XOR gates are required. As for latches, the register A, B and C are composed of k d-bit latches, $k+1$ d-bit latches and one d-bit latch, respectively. Thus, totally $2d^2 + 2kd + 2d$ 1-bit latches are required. Moreover, the multiplier requires d switches for register C and $kd + d$ MUXs for register B.

As for the computation latency, the proposed scalable multiplier requires k^2 Hankel matrix-vector computations to perform a complete m-bit multiplication. Each Hankel matrix-vector computation performed with the $d \times d$ Hankel multiplier requires a latency of $2d-1$ clock cycles. Moreover, in each computation round, the sub-vector $\boldsymbol{C}_i$ is outputted after k clock cycles due to the feedback structure of the summation circuit for register C. The lattermost sub-vector $\boldsymbol{C}_{k-1}$ is then outputted after k computation rounds. Hence, the total latency for obtaining the desired complete product vector $\boldsymbol{C}$ is $k^2 + 2d - 2$ clock cycles. Besides, the critical path delay is the time duration required by each U-cell in the $d \times d$ Hankel multiplier that is $T_A + T_X$ where T_A and T_X are the time delay of a 2-input AND gate and a 2-input XOR gate, respectively. Table 1 summarizes the above space and time complexities of the proposed scalable multiplier and shows the comparisons between our multiplier and other non-scalable multipliers (bit-parallel [7] and digit-serial [12,13]). The table reveals that the proposed multiplier has lower space complexity $O(d^2)$ as compared to the non-scalable architectures ($O(m^2)$ for bit-parallel and $O(kd^2)$ for digit-serial). It clearly demonstrates the superiority of the proposed scalable multiplier.

Table 1. Comparisons between various multipliers over GF(2^m)

Multiplier	Kim et al. [12]	Ibrahim et al. [13]	Lee et al. [7]	Proposed (Fig. 1)
Basis	Polynomial	Dual	Dual	Dual
Architecture	Digit-serial	Digit-serial	Bit-parallel	Scalable

Space complexity				
#2-input AND	$k(2d^2+d)$	$2kd^2$	m^2	d^2
#2-input XOR	$2kd^2$	$2kd^2$	m^2	d^2+2d
#1-bit latch	$10kd+k$	$6kd$	$2m^2+m$	$2kd+2d^2$ $+2d$
#1×2 SW	0	0	m	d
#2×1 MUX	$2kd$	d	m	$kd+d$
Critical path delay time	dT_A+dT_X $+(d-1)T_{MUX}$	T_A+2dT_X $+T_{MUX}$	T_A+T_X	T_A+T_X
Latency (unit = cycle)	$3k$	$2k$	$2m$	k^2+2d -2

$k=\lceil m/d \rceil$, d : selected digit size

V. CONCLUSIONS

This paper investigates a scalable scheme for dual basis multiplication over GF(2^m) . By utilizing the block Hankel matrix-vector representation, a novel low-complexity scalable and systolic dual basis multiplier for GF(2^m) generated by irreducible trinomials is derived and proposed. The scalable architecture has the advantage of achieving good trade-off between throughout performance and hardware complexity for implementing cryptographic schemes in a constrained environment such as smart cards and embedded systems by choosing appropriate digit size *d*. Analytical results have confirmed that the proposed scalable architecture has lower space complexity as compared to non-scalable architectures. Furthermore, due to the features of regularity, modularity and concurrency, the proposed scalable architecture is well suited to VLSI implementations.

REFERENCE

[1] D.E.R. Denning, *Cryptography and Data Security*, Addison-Wesley Longman Publishing Co., Inc., 1983.

[2] M.Y. Rhee, *Cryptography and Secure Communications*, McGraw-Hill, Singapore, 1994.

[3] A.J. Menezes, P.C.V. Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[4] I.S. Hsu, T.K. Truong, L.J. Deutsch, and I.S. Reed, "A comparison of VLSI architecture of finite field multipliers using dual, normal, or standard bases," *IEEE Transactions on Computers*, vol.37, no.6, pp.735-739, 1988.

[5] S.T.J. Fenn, M. Benaissa, and D. Taylor, "GF(2^m) multiplication and division over the dual basis," *IEEE Transactions on Computers*, vol.45, no.3, pp.319-327, 1996.

[6] C.W. Wu and M.K. Chang, "Bit-level systolic arrays for finite-field multiplications," *The Journal of VLSI Signal Processing*, vol.10, no.1, pp.85-92, Jun.1995.

[7] C.Y. Lee, J.S. Horng, and I.C. Jou, "Low-Complexity Bit-Parallel Multiplier over GF(2^m) Using Dual Basis Representation," *Journal of Computer Science and Technology*, vol.21, no.6, pp.887-892, Nov.2006.

[8] C.Y. Lee and C.W. Chiou, "Efficient Design of Low-Complexity Bit-Parallel Systolic Hankel Multipliers to Implement Multiplication in Normal and Dual Bases of GF(2^m)," *IEICE Trans Fundamentals*, vol.E88-A, no.11, pp.3169-3179, Nov.2005.

[9] A. Reyhani-Masoleh and M.A. Hasan, "Low complexity bit-parallel architectures for polynomial basis multiplication over GF(2^m)," *IEEE Transactions on Computers*, vol.53, no.8, pp.945-959, 2004.

[10] C.Y. Lee, "Low complexity bit-parallel systolic multiplier over GF(2^m) using irreducible trinomials," *IEE Proceedings Computers and Digital Techniques*, vol.150, no.1, pp.39-42, 2003.

[11] C. Paar, P. Fleischmann, and P. Soria-Rodriguez, "Fast arithmetic for public-key algorithms in Galois fields with composite exponents," *IEEE Transactions on Computers*, vol.48, no.10, pp.1025-1034, 1999.

[12] C.H. Kim, C.P. Hong, and S. Kwon, "A digit-serial multiplier for finite field GF(2^m)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.13, no.4, pp.476-483, 2005.

[13] M.K. Ibrahim and A. Aggoun, “Dual basis digit serial GF(2^m) multiplier,” *International Journal of Electronics*, vol.89, no.7, pp.517-523, 2002.
[14] J.H. Guo and C.L. Wang, “Digit-serial systolic multiplier for finite fields GF(2^m),” *IEE Proceedings Computers and Digital Techniques*, vol.145, no.2, pp.143-148, 1998.
[15] A.F. Tenca and C.K. Koc, “A scalable architecture for modular multiplication based on Montgomery's algorithm,” *IEEE Transactions on Computers*, vol.52, no.9, pp.1215-1221, 2003.
[16] C.Y. Lee, C.W. Chiou, J.M. Lin, and C.C. Chang, “Scalable and systolic Montgomery multiplier over GF(2^m) generated by trinomials,” *IET Circuits, Devices & Systems*, vol.1, no.6, pp.477-484, 2007.
[17] M. Morii, M. Kasahara, and D.L. Whiting, “Efficient bit-serial multiplication and the discrete-time Wiener-Hopf equation over finite fields,” *IEEE Transactions on Information Theory*, vol.35, no.6, pp.1177-1183, 1989.
[18] National Institute for Standards and Technology, *Digital signature standard*, FIPS Publication 186-2, 2000.