Cellular Automata Based Multiplier for Public-Key Cryptosystem

Hyun-Sung Kim[1] and Kee-Young Yoo[2]

[1] Kyungil University, Computer Engineering,
712-701, Kyungsansi, Kyungpook Province, Korea
kim@kiu.ac.kr
[2] Kyungpook National University, Computer Engineering,
702-701 Daegu, Korea
yook@knu.ac.kr

Abstract. This paper proposes two new multipliers based on cellular automata over finite field. Finite fields arithmetic operations have been widely used in the areas of data communication and network security applications. First, a multiplier with generalized irreducible polynomial is implemented with MSB-first fashion. Then, new algorithm and architecture are proposed to reduce the size of the first multiplier. The algorithm and architecture uses the property of irreducible all one polynomial as a modulus. Since the proposed architectures have regularity, modularity and concurrency, they are suitable for VLSI implementation and could be used in IC cards because they have particularly simple architecture. They can be used as a basic architecture for the public-key cryptosystems.

1 Introduction

Finite field $GF(2^m)$ arithmetic is fundamental to the implementation of a number of modern cryptographic systems and schemes of certain cryptographic systems[1][2]. Most arithmetic operations, such as exponentiation, inversion, and division operations, can be carried out using just a modular multiplier or using modular multiplier and squarer. Therefore, to reduce the complexity of these arithmetic architectures, an efficient architecture for multiplication over $GF(2^m)$ is necessary.

In 1984, Yeh, et al. [7] developed a parallel systolic architecture for performing the operation $AB + C$ in a general $GF(2^m)$. A semi-systolic array architecture in [8] was proposed using the standard basis, whereas architecture to compute multiplication and inversion were represented using the normal basis [9]. A systolic power-sum circuit was presented in [10], and many bit-parallel systolic multipliers have been proposed. However, these multipliers still have some shortages for cryptography applications due to their system complexity. For the better complexity, Itoh and Tsujii [11] designed two low-complexity multipliers for the class of $GF(2^m)$, based on an irreducible all one polynomial (AOP) of degree m and irreducible equally spaced polynomial of degree m. Later, Fenn, et al. in [12] and Kim in [13] developed linear feedback shift register (LFSR) based multipliers with low complexity of hardware architecture using the property of AOP.

D. Hutter et al. (Eds.): Security in Pervasive Computing 2003, LNCS 2802, pp. 227–236, 2004.

Cellular automata (CA), first introduced by John Von Neumann in the 1950s, have been accepted as a good computational model for the simulation of complex physical systems [3]. Wolfram defined the basic classification of cellular automata using their qualitative and functional behavior [4], whereas Pries, et al. focused on the group properties induced by the patterns of behavior [5]. Choudhury proposed an LSB-first multiplier using CA with low latency [14]. However, all such previously designed systems still have certain shortcomings.

Accordingly, the purpose of this paper is to propose two new modular multipliers over $GF(2^m)$. Two multipliers deploy the mixture of the advantages from the previous architectures in the perspective of area and time complexity. First, a multiplier, denoted by GMM, is implemented with generalized irreducible polynomial. It uses MSB-first modular multiplication algorithm. Then, new algorithm and architecture are proposed to reduce the size of GMM. The algorithm and architecture uses the property of irreducible all one polynomial as a modulus. The proposed architectures can be used as a kernel circuit for exponentiation, inversion, and division architectures. These operations are very important part to implement public key cryptosystems. If we use the proposed multipliers to implement cryptosystem, we can get a great cryptosystem with low hardware and time complexity. They are easy to implement VLSI hardware and could be used in IC cards as the multipliers have a particularly simple architecture.

2 Background

This section provides the necessary operations in the public-key cryptosystem and a brief description of finite fields and cellular automata. These properties will be used to derive new multipliers.

2.1 Public-Key Cryptosystem

Elgamal proposed public-key cryptosystem[2]; it gets its security from the difficulty of calculating discrete logarithms in a finite field. To generate a key pair, first choose a prime, p, and two random numbers, g and x, such that both g and x are less than p, then calculate

$$y = g^x \bmod p \tag{1}$$

The public key is y, g, and p. Both g and p can be shared among a group of users. The private key is x. The Elgamal scheme using this property of equation 1 can be used for both digital signatures and encryption.

When designing ECC (Elliptic Curve Cryptosystem), the sum of two points on the elliptic curve requires a number of divisions, which can be computed using multiplication and a multiplicative inverse, $A/B = AB^{-1}$. An inverse can be regarded as a special case of exponentiation because $B^{-1} = B^{(2^m-2)}$.

Therefore, modular exponentiation over finite fields is very critical operation to implement public-key cryptosystem as described in the above. So, it is necessary to see the operation in detail. Let C and M be elements of $GF(2^m)$, the exponentiation of M is then defined as

$$C = M^E,\ 0 \le E \le n \tag{2}$$

For a special case, $M = \alpha$. The exponent E, which is an integer can be expressed by $E = e_{m-1}2^{m-1} + e_{m-2}2^{m-2} + ... + e_1 2^1 + e_0$. The exponent also can be represented with vector representation $[e_{m-1}e_{m-2}...e_1e_0\ ]$. A popular algorithm for computing exponentiation is the binary method proposed by Knuth [15]. The exponentiation of M can be expressed as

$$M^E = M^{e_0}(M^{2^1})^{e_1}(M^{2^2})^{e_2}...(M^{2^{m-1}})^{e_{m-1}} \tag{3}$$

Based on equation 3, an algorithm for computing exponentiations is presented as following algorithm.

[Algorithm 1] Knuth's Binary Method
Input : $M, E, f(x)$
Output : $C=M^E \bmod f(x)$

1: $T=M$
2: if (e_0==1) $C=T$ else $C=\alpha^0$
3: for i=1 to m-1
4: $T=TT \bmod f(x)$
5: if (e_i==1) $C=CT \bmod f(x)$

The algorithm shows that the exponentiation can be performed with squaring in line 4 and multiplication in line 5. Modular squaring can be considered as a special case of modular multiplication, which has the same input for operand and operator. Next section presents new architectures for this modular multiplication with significantly low complexity of operations.

2.2 Finite Fields

A finite field $GF(2^m)$ contains 2^m elements that are generated by an irreducible polynomial of degree m over $GF(2)$. A polynomial $f(x)$ of degree m is said to be irreducible if the smallest positive integer n for which $f(x)$ divides $x^n + 1$ is $n = 2^m - 1$[6]. Let $f(x) = x^m + f_{m-1}x^{m-1} + ... + f_1x^1 + f_0$ be an irreducible polynomial over $GF(2^m)$ and be a root of $f(x)$. Any field element $GF(2^m)$ can be represented by a standard basis such as

$$a = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + ... + a_0 \tag{4}$$

where $a_i \in GF(2)$ for $0 \le i \le m-1$. $\{1, \alpha, \alpha^2, ..., \alpha^{m-2}, \alpha^{m-1}\}$is an ordinary standard basis of $GF(2^m)$.

It has been shown that an all one polynomial (AOP) is irreducible if and only if m+1 is a prime and 2 is a generator of the field $GF(m+1)$ [11]. The values of m for which an AOP of degree m is irreducible are 2, 4, 10, 12, 18, 28, 36, 52, 58, 60, 66, 82, and 100 for $m \le 100$. Let $ff(x) = x^m + x^{m-1} + ... + x + 1$ be an irreducible AOP over $GF(2^m)$ and be the root of $ff(x)$ such that $ff(\alpha) = \alpha^m + \alpha^{m-1} + ... + \alpha + 1 = 0$. Then we have

$$\alpha^m = \alpha^{m-1} + \alpha^{m-2} + ... + \alpha + 1, \alpha^{m+1} = 1$$

The reduction is often performed using the polynomial $\alpha^{m+1} + 1$. This property of irreducible polynomial is very adaptable for PBCA architecture, which will be described in sub-section 2.3.

If it is assumed that $\{1, \alpha, \alpha^2, \alpha^3, ..., \alpha^m\}$ is an extended standard basis, the field element *A* can also be represented as

$$A = A_m\alpha^m + A_{m-1}\alpha^{m-1} + A_{m-2}\alpha^{m-2} + ... + A_0 \quad (5)$$

where $A_m = 0$ and $A_i \in GF(2)$ for $0 \leq i \leq m$. Here, $a = A \pmod{ff(x)}$, where $ff(x)$ is an AOP of degree *m*, then the coefficients of *a* are given by $a_i = A_i + A_m \pmod 2, 0 \leq i \leq m-1$.

2.3 Cellular Automata

Cellular automata are finite state machines, defined as uniform arrays of simple cells in *n*-dimensional space. They can be characterized by looking at four properties: the cellular geometry, neighborhood specification, number of states per cell, and algorithm used for computing the successor state. Cells are restricted to local neighborhood interaction and have no global communication. A cell uses an algorithm, called its computation rule, to compute its successor state based on the information received from its nearest neighbors. An example is shown below for 2-state 3-neighborhood 1-dimensional CA [3-5].

Neighborhood state	:	111	110	101	100	011	010	001	000	
State coefficient	:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Next state	:	0	1	0	1	1	0	1	0	(rule 90)
Next state	:	1	1	1	1	0	0	0	0	(rule 240)

In the above example, the top row gives all eight possible states of the 3-neighboring cells at time *t*. The second row is the state coefficient, while the third and last rows give the corresponding states of the ith cell at time *t*+1 for two illustrative CA rules. If the next state function of a cell is expressed in the form of a truth table, then the decimal equivalent of the out column in the truth table is conventionally called the rule number for the cell. The next state of the CA is determined by the current state and the rules that govern its behavior. Let the previous, current, and next state be s_{i-1}, s_i, and s_{i+1}. The two rules 90 and 240 result in the following:

$$\text{rule 90: } S_i^+ = s_{i-1} \oplus s_{i+1}, \text{ rule 240: } S_i^+ = s_{i-1}$$

where S_i^+ denotes the next state for cell s_i, and '$\oplus$' denotes an XOR operation.

In addition, CA is composed of Linear CA, Non-Linear CA, and Additive CA according to the operation rules applied between cells. Linear CA are restricted to linear rules of operation, which means that the next state of the machine can only be computed from the previous state using a linear operation, i.e., an XOR operation. Non-Linear CA is composed of XOR operation plus other operations, while additive CA is composed of

only XOR and/or XNOR operations. CA is also classified as Uniform or Hybrid based on the rules applied. Uniform CA only has one rule, whereas Hybrid CA has two or more rules. Finally, CA can be referred to as named 1-dimensional, 2-dimensional, and 3-dimensional based on the structure of the array.

There are various possible boundary conditions, for example, a Null-Boundary CA (NBCA), where the extreme cells are connected to the ground level, a Periodic-Boundary CA (PBCA), where extreme cells are adjacent, etc.

The present state of a CA having *m* cells can be shown in terms of an *m* vector $v = (v_0\ v_1\ v_2\ ...\ v_{m-1})$, where v_i is the value of cell *i*, and v_i is an element of *GF*(2). The next state of a linear CA can be determined by multiplying the characteristic matrix with the vector in the present state, where the characteristic matrix denoted by *T* show all rules of the CA. If v^t is a column vector representing the state of the automata at the *t*-th instant of time, then the next state of the linear CA is given by $v^{t+1} = T \times v^t$.

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 1. Characteristic matrix for NBCA with rules (90, 240, 90, 240).

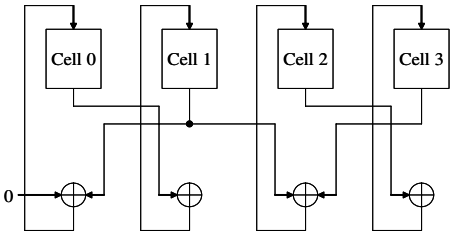

Fig. 2. One-dimensional 4-cell NBCA using rules (90, 240, 90, 240).

Fig. 2 shows a 4-cell 2-state 3-neighborhood 1-dimensional CA architecture using rules $\langle 90, 240, 90, 240 \rangle$.

3 Modular Multipliers

This section presents two new multipliers over $GF(2^m)$. These multipliers are based on cellular automata. First, we propose a new modular multiplier, denoted by GMM, using a generalized irreducible polynomial with the ordinary modular multiplication. Then, devise a new multiplication algorithm and a new multiplier, denoted by AMM, which uses the property of irreducible AOP as a modulus.

3.1 Generalized Modular Multiplier (GMM)

Let *a* and *b* be the elements over $GF(2^m)$ and $f(x)$ be the modulus. Then each element over ordinary standard basis is expressed as follows:

$$a = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + ... + a_0$$
$$b = b_{m-1}\alpha^{m-1} + b_{m-2}\alpha^{m-2} + ... + b_0$$
$$f(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + ... + f_0$$

For steps 4 and 5 for modular exponentiation in Algorithm 1, the modular multiplication $p = ab \mod f(x)$ can be represented with a recurrence equation with MSB (Most Significant Bit) first fashion as follows:

$$\begin{aligned} p &= ab \bmod f(x) \\ &= \{...[b_{m-1}a\,\alpha \bmod f(x) + b_{m-2}a]\alpha \bmod f(x) + ... + b_1 a\}\alpha \bmod f(x) + b_0 a \end{aligned} \quad (6)$$

Following algorithm 2 shows the ordinary modular multiplication.

[Algorithm 2] Ordinary Modular Multiplication
Input : $a, b, f(x)$
Output : $p = ab \bmod f(x)$
Initial value : $p^{(m)} = (p^{(m)}_{m-1}, p^{(m)}_{m-2}, \ldots, p^{(m)}_{0}) = (0, 0, \ldots, 0)$

1: for $i = m-1$ to 0 do
2: for $j = m-1$ to 0 do
3: $p^{(i)}_j = p^{(i+1)}_{j-1} + p^{(i+1)}_{m-1} f_j + b_i a_j$

The following is the basic operation for performing $p_j^{(i)} = p_{j-1}^{(i+1)} + p_{m-1}^{i+1} f_j + b_i a_j$ at step 3.

Operation 3-1: *m*-tuple of *p* is circular shifting 1-bit to the left as follows:

$$(p'_{m-2}, p'_{m-3}, ..., p'_{m-1}) \leftarrow (p_{m-1}, p_{m-2}, ..., p_1, p_0)$$

Operation 3-2: Multiply b_i to *m*-tuple of a, add it to *m*-tuple of *p*, and apply modular operation with $p_{m-1}^{(i+1)} f_j$ as follows:

$$\begin{aligned} (p_{m-1}, p_{m-2}, ..., p_1, p_0) \leftarrow (p'_{m-2}, p'_{m-3}, ..., p'_0, p'_{m-1}) + b_i(a_{m-1}, a_{m-2}, ..., a_1, a_0) \\ + p'_{m-1}(f_{m-1}, f_{m-2}, ..., f_1, f_0) \end{aligned}$$

In order to perform operation 3-1, the 1-dimensional PBCA having *m* cells is used which is the upper part with gray colored in Fig. 3. *p* is inputted into *m* cells of CA and CA has a characteristic matrix with all rules 170 for the operation 3-1. Fig. 3 shows the proposed generalized modular multiplier using PBCA with rules 170. It is possible to perform multiplication in *m* clock cycles over $GF(2^m)$.

Each cell has 2-AND (two 2-input AND gates) and 2-XOR (two 2-input XOR gates) except the last cell. It is because the last cell does not need one XOR operation since the last input always has the value of '0'.

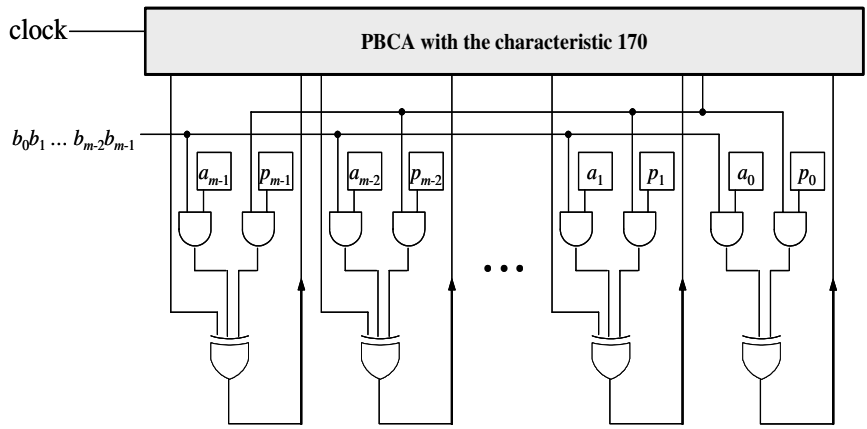

Fig. 3. Generalized modular multiplier (GMM).

3.2 AOP Modular Multiplier (AMM)

Let A and B be the elements over $GF(2^m)$ and $t(x)$ be the modulus which uses the property of an irreducible AOP. Then each element over extended standard basis is expressed as follows:

$$A = A_m\alpha^m + A_{m-1}\alpha^{m-1} + A_{m-2}\alpha^{m-2} + \ldots + A_0$$
$$B = B_m\alpha^m + B_{m-1}\alpha^{m-1} + B_{m-2}\alpha^{m-2} + \ldots + B_0$$
$$t(x) = \alpha^{m+1} + 1$$

From the recurrence equation 6, new modular multiplication $P = AB \bmod t(x)$ can be derived which applied the property of AOP as a modulus as follows:

$$\begin{aligned} P &= AB \bmod t(x) \\ &= [CLS(\ldots[CLS([CLS(B_mA)] + B_{m-1}A)] + \ldots)] + B_1A)] + B_0A \end{aligned} \quad (7)$$

Circular shifting 1-bit to the left is represented as $CLS()$ in equation 7. After the applying the property of AOP as a modulus, modular reduction is efficiently performed with just $CLS()$ operation.

Following shows the proposed modular multiplication with AOP as an irreducible polynomial.

[Algorithm 3] Proposed Modular Multiplication
Input : A, B
Output : $P = AB \bmod \alpha^{m+1}+1$
Initial value : $P^{(m+1)} = (P^{(m+1)}_m, P^{(m+1)}_{m-1}, \ldots, P^{(m+1)}_0) = (0, 0, \ldots, 0)$

1: for $i = m$ to 0 do
2: $P^i = CLS(P^{i+1})$
3: for $j = m$ to 0 do
4: $P^{(i)}_j = P^{(i)}_j + B_iA_j$

The following is the basic operation for performing modular multiplication from the above algorithm.

Operation 2: (m+1)-tuple of P is circular shifting 1-bit to the left as follows:

$$(P'_{m-1}, P'_{m-2}, ..., P'_0, P'_m) \leftarrow (P_m, P_{m-1}, P_{m-2}, ..., P_0)$$

Operation 4: Multiply B_i to (m+1)-tuple of A and add it to (m+1)-tuple of P as follows:

$$(P_m, P_{m-1}, P_{m-2}, ..., P_0) \leftarrow (P'_{m-1}, P'_{m-2}, ..., P'_0, P\prime_m) + B_i(A_m, A_{m-1}, ..., A_0)$$

In order to perform operation 2, the 1-dimensional PBCA having m+1 cells is also used which is the upper part with gray colored in Fig. 4. P is input into m+1 cells of CA and CA has the same characteristic matrix with GMM for operation 2. Operation 4 is very simplified compared with Operation 3-2 for GMM. Fig. 4 shows the proposed AOP modular multiplier using PBCA with rules 170. It is possible to perform multiplication in m+1 clock cycles over $GF(2^m)$.

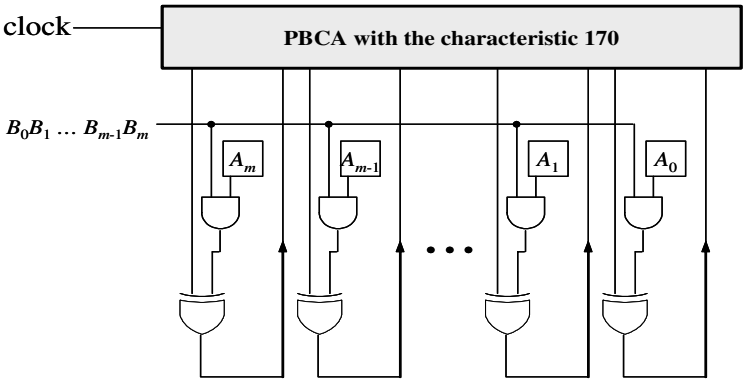

Fig. 4. AOP modular multiplier (AMM).

Each cell has one 1-AND and one 1-XOR (one 2-input XOR gate). AMM has more lower hardware complexity than GMM.

4 Analysis

Proposed two multipliers, GMM and AMM, were simulated using ALTERA MAX PLUSII simulation tool with FLEX10K device.

Table 1 shows a comparison between proposed and previous modular multipliers. For the comparison, it is assumed that n AND and n XOR represent n number of 2-input AND gate and XOR gate, respectively, and REG and Latch represents 1-bit register and latch, respectively. Comparison shows that the proposed architectures hybrid the advantages from previous architectures. Proposed two architectures, GMM and AMM, have

Table 1. Comparison for modular multipliers.

Item / Circuit	Function	Number of cells	Latency	Hardware Complexity	Critical Path
Yeh in [7]	$AB+C$	m	$3m$	$3m$ AND $2m$ XOR $11m$ Latch	AND+XOR
Chaudhury in [14]	$AB+C$	m	m	$2m$ AND $2m$ XOR $4m$ REG	AND+XOR
Fenn in [12]	AB	m+1	$2m$-1	$2m$-1 AND $2m$-2 XOR $2m$+2 REG	AND +($\log_2 m$)XOR
Kim in [13]	AB	m+1	$2m$+1	m+1 AND m XOR $2m$+2 REG	AND +($\log_2 m$)XOR
GMM	$AB+C$	m	m	$2m$ AND $2m$-1 XOR $3m$ REG	AND+2XOR
AMM	$AB+C$	m+1	m+1	m+1 AND m+1 XOR $3m$+3 REG	AND+XOR

good property in the perspective of hardware and time complexity compared with the architecture of Yeh. et al in [7]. Their architecture is based on systolic array. Chaudhury in [4] proposed multiplier based on cellular automata with LSB-first fashion. The multiplier has very similar property with GMM. But AMM reduced hardware complexity than Chaudhurys. Fenn et al. in [12] and Kim in [13] designed modular multipliers based on LFSR (Linear Feedback Shift Register) architecture. They both used the property of AOP as irreducible polynomial. The hardware complexity of their architecture is similar with AMM but AMM is very efficient for the time complexity.

5 Conclusion

This paper proposed two new modular multipliers over $GF(2^m)$. They are based on cellular automata, especially PBCA. First, a multiplier, denoted by GMM, with generalized irreducible polynomial was implemented based on the MSB-first algorithm. Then, new algorithm and architecture were proposed to reduce the size of GMM. New algorithm and architecture uses the property of irreducible all one polynomial as a modulus. Proposed multipliers hybrid the advantages from previous architectures.

Since the proposed multipliers have regularity, modularity and concurrency, they are suitable for VLSI implementation. The proposed multipliers can be used as a kernel circuit for public-key cryptosystem, which requires exponentiation, inversion, and division as their basic operation.

Acknowledgement

This research was supported by University IT Research Center Project.

References

1. I. S. Reed ,. T. K. Truong.: The use of finite fields to compute convolutions. IEEE Trans. on Information Theory. IT-21, pp. 208-213, Mar. 1975.
2. B. Schneier.: Applied Cryptography - second edition. John Wiley and Sons, Inc. 1996.
3. V. Neumann.: The theory of self-reproducing automata. Univ. of Illinois Press, Urbana London. 1966.
4. S. Wolfram.: Statistical mechanics of cellular automata. Rev. of Modern Physics. vol. 55, pp. 601-644, 1983.
5. W. Pries., A. Thanailakis., and H. C. Card.: Group properties of cellular automata and VLSI applications. IEEE Trans. on Computers. C-35, vol. 12, pp. 1013-1024, Dec. 1986.
6. E. R. Berlekamp.: Algebraic Coding Theory. New York: McGraw-Hill. 1986.
7. C. S. Yeh., S. Reed.,T. K. Truong.: Systolic multipliers for finite fields $GF(2^m)$. IEEE Trans. on Computers. vol. C-33, pp.357-360, Apr. 1984.
8. S. K. Jain., L. Song.: Efficient Semisystolic Architectures for finite field Arithmetic. IEEE Trans. on VLSI Systems. vol. 6, no. 1, pp. 101-113, Mar. 1998.
9. J. L. Massey., J. K. Omura.: Computational method and apparatus for finite field arithmetic. U. S. Patent application. submitted 1981.
10. S. W. Wei.: A systolic power-sum circuit for $GF(2^m)$. IEEE Trans. on Computers. vol. 43, pp. 226-229, Feb. 1994.
11. T. Itoh., S. Tsujii.: Structure of parallel multipliers for a class of finite fields $GF(2^m)$. Info. Comp. vol. 83, pp. 21-40, 1989.
12. S. T. J. Fenn., M. G. Parker., M. Benaissa., D. Taylor.: Bit-serial Multiplication in $GF(2^m)$ using irreducible all one polynomials. IEE. Proc. Comput. Digit. Tech. vol. 144, no. 6, Nov. 1997.
13. H. S. Kim.: Serial AOP Arithmetic Architecture for Modular Exponentiation. Ph. D. Thesis, Kyungpook National Univ. 2002.
14. P. Pal. Choudhury., R. Barua.: Cellular Automata Based VLSI Architecture for Computing Multiplication and Inverses in $GF(2^m)$. IEEE 7th International Conference on VLSI Design. Jan. 1994.
15. D. E. Knuth.: The Art of Computer Programming. Volume 2: Seminumerical Algorithms. Addison-Wesley, Reading, Massachusetts, 2nd edition. 1998.