Computing the Modular Inverse of a Polynomial Function over $GF(2^P)$ Using Bit Wise Operation

Rajaram Ramasamy and Amutha Prabakar Muniyandi

(Corresponding author: Rajaram Ramasamy)

Department of Computer Science and Engineering, Thiayagarajar College of Engineering
Thiruparankundram, Madurai, Tamil Nadu, India, 625 015, India (Email: rrajaram@tce.edu)

(Received May 26, 2008; revised Dec. 4, 2008; and accepted Feb. 10, 2009)

Abstract

Most public key crypto systems use finite field modulo arithmetic. This modulo arithmetic is applied on real numbers, binary values and polynomial functions. The computation cost is based on how it works with minimum use of scarce resources like processor and memory We have implemented the determination of the multiplicative inverse of a polynomial over $GF(2^p)$ with minimum computational cost. The "Extended Euclidean Algorithm" (EEA) has been demonstrated to work very well manually for integers and polynomials. However polynomial manipulation cannot be computerized directly. We have implemented the same by using simple bit wise shift and XOR operations. In small applications like smart cards, mobile devices and other small memory devices, this method works very well. To the best of our knowledge, the proposed algorithm seems to be the first, efficient and cost effective implementation of determining the multiplicative inverse of polynomials over $GF(2^p)$ using computers. As this is a pioneering work, the results could not be compared with that of any previous work.

Keywords: Advance encryption standard, extended Euclidean algorithm, multiplicative inverse

1 Introduction

Euclid proposed an algorithm to determine the multiplicative inverse of polynomials over $GF(Z^p)$ [1, 5, 14]. In modern cryptography, finite fields and number theory play a major role. Some basic operations in finite field are essential to develop encryption algorithm like Advance Encryption Standard [6] and ECC [11]. Advanced Encryption Standard largely relies on S-Box values to introduce non-linearity in the encryption process. Using a row and a column value, expressed in Hexadecimal notation, accesses an element of the S-Box. The hexadecimal integer is representative of a character in a message to be encrypted. Evaluation of the corresponding elemental value of the S-Box is quite a circumlocutory process.

The integer is first expressed as a polynomial in x, say, $95_H = 10010101 \rightarrow x^7 + x^4 + x^2 + 1$. Then its multiplicative inverse in $GF(2^8)$ is determined. The polynomial to represent $GF(2^8)$ is a carefully selected prime number in the range of 2^8 to 2^9. Say, 283=100011101 $\rightarrow$ $x^8 + x^4 + x^3 + x + 1$. This is called an irreducible polynomial [8, 14]. It is so chosen such that it has a unique multiplicative inverse. Then we apply the Extended Euclidean Algorithm (EEA) to these polynomials to evaluate the multiplicative inverse. This means, we have to determine the multiplicative inverse of $x^7 + x^4 + x^2 + 1$ with respect to $x^8 + x^4 + x^3 + x + 1$. Manual operation on the EEA is quite easy and straightforward. But how do we implement the process by using computers? This is what this paper does. It proposes an algorithm, which implements it efficiently and cost effectively. It implements the algorithm in C/C++ for two different cases. This is the first attempt at proposing an algorithm to determine the multiplicative inverse of a polynomial over the $GF(2^8)$ finite plane. Therefore the results got through this method could not be compared with past works.

We organize this paper as follow: In Section 2 papers that have appeared on this topic are surveyed. In Section 3, the proposed algorithm is explained. Section 4 defines the problem. Section 5 defines the proposed algorithm. Section 6 implements the proposed method. Section 7 gives the concluding remarks and discusses future scope of this problem.

2 Related Work

Stallings [14] has used the Extended Euclidean Algorithm to solve linear Diophantine equations, GCDs, and module inverses. Ever since Diffie and Hellman [3] developed the prototype of modern cryptography; most public key cryptosystems are based on finite fields with modular arithmetic constituting basic operations. Modular multiplications, modular exponentiations, and modular inverses are performed in RSA cryptosystems [13], the US Government Digital Signature Algorithm [10], the Diffie Hellman Key Exchange Scheme [10]. Among the basic operations [4, 7], computing modular inverses involving polynomials is the most complex. This has engaged the attention of many researchers [1, 2, 9, 15].

In 1997, Calvez et al. [1] proposed a variation on the Euclidean Algorithm, which determines the greatest common divisors (GCDs) and inverses of polynomials. In 2004, Goupil and Palicot [5] introduced another variation on this algorithm to reduce the number of operations to a large extent. Inspired by their work, Liu [9] proposes a variation on the EA, which uses only simple modulo operators (subtraction operations), to compute the modular inverses. This variant only modifies the initial values and the termination condition of the EA. Therefore it is as simple as the EA. However one drawback is that the input of this variant is twice the size of bit length as the input of the EA.

Liu's algorithm only deals with number system [9]. It is not applicable for polynomial functions. Polynomials with coefficients other than 1 are difficult to implement in computers [9]. The reason for this is due to occurrence of some negative or fractional value coefficients. The proposed algorithm is based on polynomials with 1 or 0 as coefficients such as $x^7 + x^5 + x^3 + x + 1$.

In 1997, the American Government [1] decided to replace DES with an efficient encryption algorithm. The National Institute of Standards and Technology (NIST) announced a common note to cryptographers for development of a new algorithm. Earlier the DES was developed using Federal encryption standard. In January 1997 NIST solicited a new symmetric algorithm based on 128-bit block of message using 128−, 192−, 256−bit keys. Cryptographers from different parts of the world submitted their proposals. Fifteen of these proposals met the NIST specifications. Based on this, NIST organized a conference to deliberate on all the proposed methods. After nineteen months of evaluation, NIST recommended five algorithms like MARS, RC6, Rijndael, Serpent, and Twofish. Then after one year of study, in October 2000, National Institute of Standard and Technology recommended the Rijndael algorithm as best suited for AES. The Rijndael algorithm is combination of security, performance, efficient, implement ability, and flexibility. After one more year of evaluation, in November 2001, the Department of Commerce officially declared Rijndael algorithm as the de facto Advance Encryption Standard.

3 Problem Description

In the field of information security some of the security algorithms are designed by using the finite field $GF(2^p)$ [2]. AES and ECC are the two important encryption techniques that use algorithms based on finite field arithmetic. The finite field $GF(2^p)$ is representative of a polynomial function with respect to one variable x, as follows:

$$GF(2^p) = x^{p-1} + x^{p-2} + \cdots + x^2 + x^1.$$

For example, $GF(2^3) = x^2 + x + 1$.

The above-mentioned $GF(2^3)$ is finite field with respect to 3. In AES, the S-box generation is designed by using irreducible polynomial in $GF(2^8)$. The strength of the AES is dependent on the non-linearity introduced in evaluating the S-Box values.

Suppose we want to generate S-box value for $2A$ with respect to $GF(2^8) = x^8 + x^4 + x^3 + x + 1$. First, we have to determine the multiplicative inverse of $2A$ in $GF(2^8)$. $2A$ in binary form is 0010 1010, which in polynomial representation in x is $(x^5 + x^3 + x)$. In manual procedure the Extended Euclidean Algorithm or its shortened version can be directly applied to polynomials to evaluate the multiplicative inverse.

The multiplicative inverse of $2A(00101010)$, expressed as a polynomial $(x^5 + x^3 + x)$, over $GF(2^8)$ is calculated manually using the abridged Euclidean Algorithm [1].

The manual operation shows that the multiplicative inverse of $(x^5 + x^3 + x)$ over $(x^8 + x^4 + x^3 + x + 1)$ is $(x^7 + x^4 + x^3)$.

In general terms this algorithm determines multiplicative inverse of $B(x)$ modulo $M(x)$, if the degree of $B(x)$ is less than the degree of $M(x)$; or alternatively we say that $gcd[M(x), B(x)] = 1$. If $M(x)$ is an irreducible polynomial, then it has no factor other than itself or 1, so that $\gcd[M(x), B(x)] = 1$.

However the computer cannot be coded to deal with these polynomial functions straightaway. They need to be handled in an indirect way. This paper proposes the technique to manipulate polynomials by the Extended Euclidean Algorithm.

4 Proposed Algorithm

In our proposed method, we have converted the polynomial function into decimal and its equivalent binary values. Both number systems figure in the computations. The proposed algorithm is given below.

Procedure Multiplicative Inverse $((A_3[], B_3[\,]))$

Binary value $A_3[\,], B_3[\,]$
Begin
$C_1 = 0$; $A_2 = 0$; $B_2 = 0$;
while $(B_3 > 1)$ //**Step 1** **do**
  $Q = 0$;
  $Temp = B_3$;
  do
    $Q_1 = 1$;
    do
    $B_3 = B_3 <<$**LinearLeftShift**
    $Q_1 = Q_1 * 2$;
    until$(A_{3MSB} == B_{3MSB})$
    $Q = Q + Q_1$;
    $A_3 = A_3[\,] \oplus B_3[\,]$;
    $B_3 = Temp$;
  until$(A_3 > Temp || BitSize(C) >= BitSize(Temp))$

Table 1: Calculation of multiplicative inverse of 2A by using abridged Euclidean algorithm for polynomials

	A1	B2	Quotient
$x^8+x^4+x^3+x+1$	1	0	-
x^5+x^3+x	0	1	x^3+x
$x^4+x^3+x^2+x+1$	-	x^3+x	x+1
x^3+x+1	-	$x^4+x^3+x^2+x+1$	x+1
x	-	x^5+x^3+x+1	x^2+1
1	-	$x^7+x^4+x^3$	x

```
      A2 = B2;
    B3 = A3; //Remainder part of A3/B3
    A3 = Temp;
    N = BitSize(Q); //Binary Bit Size of Q
    Temp = B2; C2 = 0; //Step 2
    do
        C2 = 0d;
        If(QN == 1) //Testing if N^th bit of Q is 1
        C1 = B2 << N - 1; //Linear left shift by N - 1 times
        C2 = C2⊕C1;
        End if
        N − −;
      until(N >= 1)
    B2 = C2;
    B2 = B2 ⊕ A2; // Multiplicative Inverse
    A2 = Temp;
end while
end
```

The above algorithm works for any polynomial function over $GF(2^p)$. In next section, we give the implementation details.

5 Implementation of Our Algorithm

We have implemented this approach for real time computation with minimum requirement. The polynomial functions are handled in the form of binary and decimal values. To convert the polynomial into decimal value, the x in the polynomial function is replaced by 2, because the base value for $GF(2^p)$ is 2. For example, $x^8+x^4+x^3+x+1 = 283$ and $x^5+x^3+x = 42$.

Our task is to determine $(x^5+x^3+x)^{-1}$ modulo $(x^8+x^4+x^3+x+1)$. As the computer cannot directly handle the polynomials, we use the numerical equivalent to the base 2. The proposed method of computation is illustrated below.

Iteration 1:

Multiplicative inverse of $2A$ $(2A_H = 42_{10})$ in $GF(2^8)$.
$C_1 = C_2 = A_2 = B_2 = 0$
Step 1.
$A_3 = 283 = 100011011$
$B_3 = 42 = 000101010$
$Q_1 = 1$ and $Q = 0$
$Temp = B_3 = 42, B_2 = 0$
$A_3 \rightarrow 100011011$
$B_3 \rightarrow 000101010$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, $B_3 <<$ linear left shift by 1 bit $Q_1 = Q_1 * 2 = 1 * 2 = 2$

$A_3 \rightarrow 100011011$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, $B_3 <<$ linear left shift by 1 bit

$B_3 \rightarrow 001010100$ $Q_1 = Q_1 * 2 = 2 * 2 = 4$
$A_3 \rightarrow 100011011$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, $B_3 <<$ linear left shift by 1 bit

$B_3 \rightarrow 010101000$ $Q_1 = Q_1 * 2 = 4 * 2 = 8$
$A_3 \rightarrow 100011011$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$

$B_3 \rightarrow 101010000$ $Q = Q + Q_1 = 8 + 0 = 8$
$A_3 \rightarrow 001001011$ Decimal value of 001001011 = 75
$A_3 = 75; B_3 = Temp = 42;$
First Condition (A_3=75)>($Temp$ = 42) //TRUE
Second Condition $BitSize$(75)>$BitSize$(42) //TRUE
$Q = 8$ and $Q_1 = 1$
$A_3 \rightarrow 1001011$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, $B_3 <<$ linear left shift by 1 bit

$B_3 \rightarrow 0101010$ $Q_1 = Q_1 * 2 = 1 * 2 = 2$
$A_3 \rightarrow 1001011$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$

$B_3 \rightarrow 1010100$ $Q = Q + Q+_1 = 8 + 2 = 10$
$A_3 \rightarrow 0011111$ Decimal value 0011111 = 31
$A_3 = 31;$ $B_3 = Temp = 42;$
First Condition (A_3=31) > ($Temp$ = 42) //FALSE
Second Condition $BitSize$(31) > $BitSize$(42) //FALSE
$A_2 = B_2$
$B_3 = A_3$
$A_3 = Temp$
$Q = 10 and B_3 = 31$
$B_2 = 1 = 0001$
$Q = 10 = 1010$

Step 2.
$N = BitSize(Q) = 4 C_2 = 0000 Temp = B_2$

N	Q_N	$C_1 = B_2 << N$	$C_2 \oplus C_1$	$\rightarrow$	C_2
3	**1**	**1000**	**0000⊕1000**	$\rightarrow$	**1000**
2	**0**	**-**	**-**	$\rightarrow$	**1000**
1	**1**	**0010**	**1000⊕0010**	$\rightarrow$	**1010**

0	**0**	**-**	**-**	$\rightarrow$	**1010**

$B_2 = C_2 = 1010$
$B_2 = B_2 \oplus A_2 = 1010 \oplus 0000 = 1010$
$A_2 = Temp = 0001$

At end of the iteration 1: $Q = 10, A_2 = 1, A_3 = 42, B_2 = 10, B_3 = 31$.

While $B_3 = 31$ and $B_3 \neq 1$ do Step 1 and Step 2 until $B_3 = 1$. When the B_3 value reaches 1 then execution will stop. The final B_2 value is the multiplicative inverse value. The value may be expressed as a polynomial function. In our case $B_2 = 152_{10} = 10011000_2$ That is $(x^5+x^3+x)^{-1}$ modulo $(x^8+x^4+x^3+x+1) = x^7+x^4+x^3$

Step 1 is used to calculate Q, A_3 and B_3 values and Step 2 is use to calculate B_2 value. Likewise, the remaining iterations lead to the calculation of the Multiplicative Inverse. The complete iterations are worked in Appendix A. This is a case where both the conditions, namely, $C > B_3$ and $Bitsize(C) == Bitsize(B_3)$ are fully met. Table 2 shows the result of the computations.

6 Performance Analysis

The program for computerizing the algorithm was developed in C++. The size of the file containing the program is 5.99 kB. It occupied a disk space of 8.192 kB. A typical smart card MEAP (Multifunctional Embedded Application Platform) processor has the following specification: 250-333MHz, 20MB RAM and 270-400 MB disk space. Therefore this program can be easily embedded in any smart card device. The table gives the execution times for two inputs, run in the above environment.

Manual input implies the input is entered during the program execution time. In automated input, the input is already incorporated in the program itself and there is no need for human intervention.

7 Conclusion

We have implemented the Extended Euclidean Algorithm for polynomials for practical use. This implementation can be easily extended for determining the elements of the S-Box used in Advance Encryption Standard algorithm. The method is easy and compact enough to adopt for smaller applications like smart card, information security in mobile device and security in small memory device. Our algorithm is efficient for determining the multiplicative inverse of polynomials over $GF(2^P)$. However for more general case of $GF(Z^P)$, a lot of further research is to be done. As future extension of this work, it is proposed to extend our computing algorithm to handle $GF(Z^P)$ also. To the best of our knowledge, this computerized method of handling polynomials using Extended Euclidean Algorithm is proposed for the first time. Therefore a comparative analysis with existing work is not possible. Further work may strive to implement the same approach for implementation in hardware for real time applications.

Acknowledgements

The authors are grateful to the management of Thiagarajar College of Engineering Madurai, India, for granting permission to undertake this research work. They express their gratitude to Smart and Secure Project sponsored by the National Technical Research Organization, Government of India, for providing financial support. Our thanks are due to the Head of the Department of Computer Science and Engineering of Thiagarajar College of Engineering for allowing us the use of the laboratories and computing facilities.

References

[1] L. C. Calvez, S. Azou, and P. Vilbe, "Variation on Euclid's algorithm for polynomials," *Electronics Letters*, vol. 33, no. 11, pp. 939-940, 1997.

[2] A. K. Daneshbeh and M. A. Hasan, "A class of unidirectional bit serial systolic architectures for multiplicative inversion and division over GF(2m)," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 370-380, 2005.

[3] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644-654, 1976.

[4] V. Z. Gathen and J. Gerhard, *Modern Computer Algebra*, 2nd Edition, *Cambridge University Press*, 2003.

[5] A. Goupil and J. Palicot, "Variation on variation on Euclid's algorithm," *IEEE Transactions on Signal Processing Letters*, vol. 11, no. 5, pp. 457-458, 2004.

[6] J. N. Jr, "Analysis of Venkaiah et al.'s AES design," *International Journal of Network Security*, vol. 9, no. 3, pp. 285-289, 2009.

[7] D. E. Knuth, *The Art of Computer Programming*, vol. 2, 3rd Edition, Addison-Wesley, Reading, MA, 1997.

[8] S. Landau, "Polynomial in the Nation's service: Using algebra to design the advanced encryption standard," *American Mathematical Monthly*, vol. 111, pp. 89-117, Feb. 2004.

[9] C. L. Liu, G. Horng, and H. Y. Liu, "Computing the modular inverse is as simple as computing the GCDs," *International Journal of Finite Fields and Their applications*, vol. 14, pp. 65-75, 2008.

[10] National Institute Fro Standards and Technology, *Digital Signature Standard (DSS)*, Federal Register, 56:169, Aug. 1991.

[11] R. R. Ramasamy, M. A. Prabakar, M. I. Devi, and M. Suguna, "Knapsack based ECC encryption and decryption," *International Journal of Network Security*, vol. 9, no. 3, pp. 218-226, 2009.

Table 2: Calculation of multiplicative inverse of $2A_H = 42_{10}$ over 283_{10} by using the proposed computerized algorithm

Iteration	Q	A_2	A_3	B_2	B_3
0	-	0	283	1	42
1	10	1	42	10	31
2	3	10	31	31	11
3	3	31	11	43	2
4	5	43	2	152	1

Table 3: Execution time needed for our proposed algorithm

Algorithm	Manual I/P	Automated I/P
Our Proposed Algorithm	3816.3 ms	140.2 ms

[12] R. L. Rivest, A. Shamir, and L. Adlelman, "A method for obtaining digital signatures and public key cryptosystems," *Communications Of the ACM*, vol. 21, no. 2, pp. 120-126, 1978.

[13] K. H. Rosen, *Elementary Number Theory and Its Application*, 4th Edition, Addison-Wesley, Reading, MA, 2000.

[14] W. Stallings, *Cryptography and Network Security Principles and Practices*, 4th Edition, Prentice-Hall India, 2006.

[15] J. Zhou, "Fast algorithms for determining the minimal polynomials of sequences with period kn Over $GF(P^m)$," *International Journal of Network Security*, vol. 7, no. 1, pp. 38-41, 2008.

Appendix A

Iteration 1

Multiplicative inverse of 2A in $GF(2^8)$. ($2A_H =$ 4210)

$C_1 = C_2 = A_2 = B_2 = 0$

Step 1.

$A_3 = 283 = 100011011$
$B_3 = 42 = 000101010$
$Q_1 = 1 and Q = 0$
$Temp = B_3 = 42, B_2 = 0$
$A_3 \rightarrow 100011011$
$B_3 \rightarrow 000101010$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, $B_3 <<$ linear left shift by 1 bit $Q_1 = Q_1 * 2 = 1 * 2 = 2$

$A_3 \rightarrow 100011011$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, $B_3 <<$ linear left shift by 1 bit

$B_3 \rightarrow 001010100$ $Q_1 = Q_1 * 2 = 2 * 2 = 4$
$A_3 \rightarrow 100011011$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, $B_3 <<$ linear left shift by 1 bit

$B_3 \rightarrow 010101000$ $Q_1 = Q_1 * 2 = 4 * 2 = 8$
$A_3 \rightarrow 100011011$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$

$B_3 \rightarrow 101010000$ $Q = Q + Q_1 = 8 + 0 = 8$
$A_3 \rightarrow 001001011$ Decimal value of 001001011 = 75
$A_3 = 75; B_3 = Temp = 42;$
First Condition (A_3=75)>($Temp$ = 42) //TRUE
Second Condition $BitSize$(75)>$BitSize$(42) //TRUE
$Q = 8$ and $Q_1 = 1$
$A_3 \rightarrow 1001011$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, $B_3 <<$ linear left shift by 1 bit

$B_3 \rightarrow 0101010$ $Q_1 = Q_1 * 2 = 1 * 2 = 2$
$A_3 \rightarrow 1001011$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$
$B_3 \rightarrow 1010100$ $Q = Q + Q_1 = 8 + 2 = 10$
$A_3 \rightarrow 0011111$ Decimal value 0011111 = 31
$A_3 = 31; B_3 = Temp = 42;$
First Condition (A_3=31) > (Temp = 42) //FALSE
Second Condition $BitSize$(31) >= $BitSize$(42) //FALSE
$A_2 = B_2$
$B_3 = A_3$
$A_3 = Temp$
$Q = 10 and B_3 = 31$
$B_2 = 1 = 0001$
$Q = 10 = 1010$

Step 2.

$N = BitSize(Q) = 4 C_2 = 0000 Temp = B_2$. See in table a

N	Q_N	$C_1 = B_2 << N$	$C_2 \oplus C_1$	$\rightarrow$	C_2
3	**1**	**1000**	**0000$\oplus$1000**	$\rightarrow$	**1000**
2	**0**	**-**	**-**	$\rightarrow$	**1000**
1	**1**	**0010**	**1000$\oplus$0010**	$\rightarrow$	**1010**
0	**0**	**-**	**-**	$\rightarrow$	**1010**

$B_2 = C_2 = 1010$
$B_2 = B_2 \oplus A_2 = 1010 \oplus 0000 = 1010$
$A_2 = Temp = 0001$

At end of the Iteration 1: $Q = 10, A_2 = 1, A_3 = 42, B_2 = 10, B_3 = 31.$
While($B_3 > 1$) **do** next iteration

Iteration 2

Step 1.

A_3=42=101010
B_3=31=011111
Q_1=1 and Q=0
$Temp = B_3$ $B_2 = 10$ $A_2 = 1$
$A_3 \rightarrow 101010$
$B_3 \rightarrow 011111$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, B_3 << linear left shift by 1 bit $Q_1 = Q_1 * 2 = 1 * 2 = 2$

$A_3 \rightarrow 101010$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$

$B_3 \rightarrow 111110$ $Q = Q + Q_1 = 0 + 2 = 2$
A_3 010100 Decimal value of 010100 = 20
A_3= 20; B_3=$Temp$=31;
First Condition (A_3=20) > ($Temp$ = 31) //FALSE
Second Condition $BitSize(20) >= BitSize(= 31)$ //TRUE
$Q = 2$ and $Q_1 = 1$
$A_3 \rightarrow 10100$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$

$B_3 \rightarrow 11111$ $Q = Q + Q_1 = 2 + 1 = 3$
$A_3 \rightarrow 01011$ Decimal value 01011 = 11
A_3= 11; B_3=$Temp$=31;
First Condition (A_3=11) > (Temp = 31) //FALSE
Second Condition $BitSize(11) >= BitSize(31)$ //FALSE
$A_2 = B_2$
$B_3 = A_3$
$A_3 = Temp$
$Q = 3$ and $B_3 = 11$
$B_2 = 10 = 1010$
$Q = 3 = 11$

Step 2.
$N = BitSize(Q) = 2$ $C_2 = 0000$ Temp $= B_2$. See in table b.

N	Q_N	$C_1 = B_2 << N$	$C_2 \oplus C_1$	$\rightarrow$	C_2
1	1	10100	00000⊕10100		1000
0	1	1010	10100⊕01010		11110

B_2=C_2=11110 = 30
B_2=$B_2 \oplus A_2$=11110⊕00001 = 11111 = 31
$A_2 = Temp = 1010 = 10$

At end of the Iteration 2: $Q = 3$, A_2=10, A_3=31, B_2=31, B_3=11.
While(B_3 >1) **do** next iteration.

Iteration 3

Step 1.
$A_3 = 31 = 11111$
$B_3 = 11 = 01011$
$Q_1 = 1$ and Q =0
$Temp = B_3$ $B_2 = 31$ $A_2 = 10$
$A_3 \rightarrow 11111$
$B_3 \rightarrow 01011$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, B_3 << linear left shift by 1 bit

$Q_1 = Q_1 * 2 = 1 * 2 = 2$
$A_3 \rightarrow 11111$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$

$B_3 \rightarrow 10110$ $Q = Q + Q_1 = 0 + 2 = 2$
$A_3 \rightarrow 01001$ Decimal value of 01001 = 9
A_3= 9; B_3=$Temp$=11;
First Condition (A_3=9) > ($Temp$ = 11) //FALSE
Second Condition $BitSize(9) >= BitSize(11)$ //TRUE
A_3=9; B_3=$Temp$=11;
$Q = 2$ and $Q_1 = 1$
$A_3 \rightarrow 1001$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$

$B_3 \rightarrow 1011$ $Q = Q + Q_1 = 2 + 1 = 3$
$A_3 \rightarrow 0010$ Decimal value 0010 = 2
A_3= 2; B_3=Temp=11;
First Condition (A_3=2) > ($Temp$ = 11) //FALSE
Second Condition $BitSize(2) >= BitSize(11)$ //FALSE
$A_2 = B_2$
$B_3 = A_3$
$A_3 = Temp = 11$
$Q = 3$ and $B_3 = 2$
$B_2 = 31 = 11111$
$Q = 3 = 11$

Step 2.
$N = BitSize(Q) = 2$ $C_2 = 0000$ $Temp = B_2$. See in table c

N	Q_N	$C_1 = B_2 << N$	$C_2 \oplus C_1$	$\rightarrow$	C_2
1	1	111110	000000⊕111110		111110
0	1	111110	111110⊕011111		100001

$B_2 = C_2 = 100001 = 33$
$B_2 = B_2 \oplus A_2 = 100001 \oplus 001010 = 101011 = 43$
$A_2 = Temp = 11111 = 31$

At end of the Iteration 3: $Q = 3$, $A_2 = 31$, $A_3 = 11$, $B_2 = 43$, $B_3 = 2$.
While(B_3 ¿ 1) **do** next iteration.

Iteration 4

Step 1.
$A_3 = 11 = 1011$
$B_3 = 2 = 0010$
$Q_1 = 1$ and Q =0
$Temp = B_3$ $B_2 = 43$ $A_2 = 31$
$A_3 \rightarrow 1011$
$B_3 \rightarrow 0010$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, B_3 << linear left shift by 1 bit

$Q_1 = Q_1 * 2 = 1 * 2 = 2$
$A_3 \rightarrow 1011$ 1^{st} bit of A_3 Not Equal to 1^{st} bit of B_3
so, $B_3 <<$ linear left shift by 1 bit

$B_3 \rightarrow 0100$ $Q_1 = Q_1 * 2 = 2 * 2 = 4$
$A_3 \rightarrow 1011$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$

$B_3 \rightarrow 1000$ $Q = Q + Q_1 = 0 + 4 = 4$
$A_3 \rightarrow 0011$ Decimal value of 0011 = 3
A_3= 3; B_3=$Temp$=2;
First Condition (A_3=3) > (Temp = 2) //TRUE
Second Condition $BitSize(3) >= BitSize(2)$ //TRUE
$Q = 4$ and $Q_1 = 1$
$A_3 \rightarrow 11$ 1^{st} bit of A_3 Equal to 1^{st} bit of B_3
so, $A_3 = A_3 \oplus B_3$

$B_3 \rightarrow 10$ $Q = Q + Q_1 = 4 + 1 = 5$
$A_3 \rightarrow 01$ Decimal value 01 = 1
A_3= 1; B_3=$Temp$=2;
First Condition (A_3=1) > ($Temp$ = 2) //FALSE
Second Condition $BitSize(1) >= BitSize(2)$ //FALSE
$A_2 = B_2$
$B_3 = A_3$
$A_3 = Temp = 2$
$Q = 5$ and B_3= 1
$B_2 = 43 = 101011$
$Q = 5 = 101$

Step 2.
$N = BitSize(Q) = 2$ $C_2 = 0000$ $Temp = B_2$

N	Q_N	$C_1 = B_2 << N$	$C_2 \oplus C_1$	$\rightarrow$	C_2
2	**1**	**10101100**	000000⊕10101100		10101100
1	**0**	**-**	**-**		10101100
0	**1**	**101011**	10101100⊕101011		10000111

$B_2 = C_2 = 10000111 = 71$
$B_2 = B_2 \oplus A_2 = 10000111 \oplus 11111 = 10011000 = 152$
$A_2 = Temp = 101011 = 43$

At end of the Iteration 4: $Q = 5$, $A_2 = 43$, $A_3 = 2$, $B_2 = 152$, $B_3 = 1$.
As $B_3 = 1$, Multiplicative Inverse of $2A$ in $GF(2^8)$ is 152=10011000→$(X^7 + X^4 + X^3)$

R. Rajaram Ramasamy Dean of CSE/IT, Thiagarajar College of Engineering, has BE degree in Electrical and Electronics Engineering from Madras University in 1966. He secured the M Tech degree in Electrical Power Systems Engineering in 1971 from IIT Kharagpur, and the Ph.D. degree on Energy Optimization from Madurai Kamaraj University in 1979. He and his research scholars have published/presented more that 45 research papers in Journals and Conferences. Eight of his scholar secured the Ph.D. degree in computer science and communications areas. Two have submitted thesis and awaiting their results. Six are currently pursuing their Ph.D. research in Anna University with his guidance. His current areas of interest are Mobile Agents, Cryptography and Data Mining. He has published more than 13 text books on Computer languages and Basi c Communications. He attended the International Seminar on Solar Energy at University of Waterloo, Canada during 1978. He has served the Makerere University at Uganda during 1977-1978 and University of Mosul during 1980-1981. He secured two best technical paper awards from the Institution of Engineers India and one from Indian Society for Technical Education. He has travelled to Malaysia, London, Paris, Belgium New York, Toronto, Nairobi.

M. Amutha Prabakar received the B. E. degree in Computer Science and Engineering, in 2003; the M. E. in Computer Science and Engineering, in 2005. He had worked as a lecturer in the department of Computer Science and Engineering, R. V. S. College of Engineering and Technology, India from 2004-2007. Now he is working as Lecturer in Department of Information Technology, Thiagarajar College of Engineering, Madurai. His current research interests include Cryptography and Security.