

An Algorithm for Inversion in $\text{GF}(2^m)$ Suitable for Implementation Using a Polynomial Multiply Instruction on $\text{GF}(2)$

Katsuki Kobayashi, Naofumi Takagi, and Kazuyoshi Takagi

Department of Information Engineering, Graduate School of Information Science, Nagoya University,
Furo-cho, Chikusa-ku, Nagoya, 464-8603 Japan.

Email: {katsu,ntakagi,ktakagi}@takagi.i.is.nagoya-u.ac.jp

Abstract

An algorithm for inversion in $\text{GF}(2^m)$ suitable for implementation using a polynomial multiply instruction on $\text{GF}(2)$ is proposed. It is based on the extended Euclid's algorithm. In the algorithm, operations corresponding to several contiguous iterations of the VLSI algorithm proposed by Brunner et al. is represented as a matrix. They are calculated at once through the matrix efficiently by means of a polynomial multiply instruction on $\text{GF}(2)$. For example, in the case where the word size of a processor and m are 32 and 571, respectively, the algorithm calculates inversion with about the half number of instructions of the conventional algorithm on the average.

1 Introduction

Galois field $\text{GF}(2^m)$, i.e. extension field of $\text{GF}(2)$, plays important roles in many applications, such as error-correcting codes and cryptography. In widely used elliptic curve cryptography (ECC), the value of m is large, i.e. 163 or more [1, 2]. Among basic arithmetic operations in $\text{GF}(2^m)$, multiplicative inversion is the most time-consuming. Therefore, there is a demand for a fast algorithm for inversion in $\text{GF}(2^m)$. In recent years, several instruction set extensions for cryptosystem have been proposed [3–6]. These instruction set extensions can increase performance of a processor with small cost. They include a polynomial multiply instruction on $\text{GF}(2)$. In this paper, we propose a new algorithm for inversion in $\text{GF}(2^m)$ which is suitable for implementation using a polynomial multiply instruction on $\text{GF}(2)$.

The algorithm to be proposed is based on the extended Euclid's algorithm. In the algorithm, operations corresponding to several contiguous iterations of the conventional algorithm for VLSI implementation [7] are represented as a matrix which is obtained only with single-word

operations. Then, they are calculated at once through the matrix efficiently by means of a polynomial multiply instruction on $\text{GF}(2)$. For example, in the case where the word size of a processor and m are 32 and 571, respectively, the algorithm calculates inversion with about the half number of polynomial multiply instructions on $\text{GF}(2)$ and XOR instructions for the conventional algorithm evaluated in [8] on the average.

This paper is organized as follows. In the next section, we show conventional algorithms for inversion in $\text{GF}(2^m)$ based on the extended Euclid's algorithm and a polynomial multiply instruction on $\text{GF}(2)$. In Sect. 3, we propose a fast algorithm for inversion in $\text{GF}(2^m)$ using polynomial multiply instruction on $\text{GF}(2)$, which is based on the extended Euclid's algorithm. In Sect. 4, we evaluate the algorithm. In Sect. 5, we make several discussions.

2 Preliminaries

2.1 Multiplicative Inverse in $\text{GF}(2^m)$

Let

$$G(x) = x^m + g_{m-1}x^{m-1} + \cdots + g_1x + 1$$

be an irreducible polynomial on $\text{GF}(2)$. Then, we can represent an element in $\text{GF}(2^m)$ defined by $G(x)$ as

$$A(x) = a_{m-1}x^{m-1} + \cdots + a_1x + a_0,$$

where $a_i \in \text{GF}(2)$ [9].

Addition and subtraction of two elements in $\text{GF}(2^m)$ are defined as polynomial addition and subtraction on $\text{GF}(2)$, respectively. Thus, both addition and subtraction are executed by exclusive-OR operation for every coefficient. Multiplication in $\text{GF}(2^m)$ is defined as a polynomial multiplication modulo $G(x)$. The multiplicative inverse $A^{-1}(x)$ of $A(x)$ in $\text{GF}(2^m)$ is defined as the element that satisfies

$$A(x) \cdot A^{-1}(x) = 1,$$

where “ \cdot ” denotes multiplication in $\text{GF}(2^m)$.

| j | $R_j(x)$ | $U_j(x)$ | $W_j(x)$ | $Q_j(x)$ |
|-----|---------------------------|---|---------------|-----------------------|
| -1 | $x^7 + x^6 + x^3 + x + 1$ | 0 | 1 | — |
| 0 | $x^6 + x^4$ | 1 | 0 | — |
| 1 | $x^5 + x^4 + x^3 + x + 1$ | $x + 1$ | 1 | $x + 1$ |
| 2 | $x^4 + x^3 + x^2 + 1$ | x^2 | $x + 1$ | $x + 1$ |
| 3 | 1 | $\frac{x^3 + x + 1}{x^7 + x^6 + x^3 + x + 1}$ | $x^2 + x + 1$ | x |
| 4 | 0 | $x^7 + x^6 + x^3 + x + 1$ | $x^6 + x^4$ | $x^4 + x^3 + x^2 + 1$ |

Figure 1. An example of inversion by Algorithm 1 ($m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$ and $A(x) = x^4 + x^2$).

2.2 Algorithm for Inversion in $\text{GF}(2^m)$ based on Extended Euclid's Algorithm

Euclid's algorithm for polynomial calculates the greatest common divisor (GCD) polynomial of two polynomials. The algorithm can be extended for calculating the two polynomials [9], $U(x)$ and $W(x)$, that satisfy

$$\text{GCD}(A(x), B(x)) = U(x) \times A(x) + W(x) \times B(x).$$

The extended Euclid's algorithm is as follows, where “ \div ” denotes the operation that calculates a quotient polynomial.

[Algorithm 1]

(Extended Euclid's Algorithm)

$$R_{-1}(x) := B(x); U_{-1}(x) := 0; W_{-1}(x) := 1;$$

$$R_0(x) := A(x); U_0(x) := 1; W_0(x) := 0;$$

$$j := 0;$$

repeat

$$j := j + 1;$$

$$Q_j(x) := R_{j-2}(x) \div R_{j-1}(x);$$

$$R_j(x) := R_{j-2}(x) - Q_j(x) \times R_{j-1}(x);$$

$$U_j(x) := U_{j-2}(x) - Q_j(x) \times U_{j-1}(x);$$

$$W_j(x) := W_{j-2}(x) - Q_j(x) \times W_{j-1}(x);$$

until $R_j(x) = 0$;

output $R_{j-1}(x)$, $U_{j-1}(x)$ and $W_{j-1}(x)$ as the results.

$$\left(\begin{array}{l} R_{j-1}(x) = \text{GCD}(A(x), B(x)) \\ \quad = U_{j-1}(x) \times A(x) + W_{j-1}(x) \times B(x) \end{array} \right)$$

□

Let $G(x)$ be the irreducible polynomial with degree m that defines the field, and $A(x)$ be the polynomial representation of an element in the field. Since the greatest common divisor polynomial of $A(x)$ and $G(x)$ is 1, we can obtain the multiplicative inverse $A^{-1}(x)$ of $A(x)$ as $U_{j-1}(x) \bmod G(x)$ by replacing $B(x)$ with $G(x)$ in Al-

gorithm 1. This is explained as follows.

$$\text{GCD}(A(x), G(x)) = U_{j-1}(x) \times A(x) + W_{j-1}(x) \times G(x)$$

$$1 \equiv U_{j-1}(x) \times A(x) \pmod{G(x)}$$

$$A^{-1}(x) \equiv U_{j-1}(x) \pmod{G(x)}$$

Figure 1 shows an example of inversion by Algorithm 1, where $m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$ and $A(x) = x^6 + x^4$. In this example, the inverse $A^{-1}(x)$ is $U_3(x)$.

It is reported by Hankerson et al. that an algorithm based on the extended Euclid's algorithm is faster than the other algorithms [8]. The algorithm evaluated in [8] is as follows, where $\text{deg}(\cdot)$ is the function to calculate the degree of a polynomial.

[Algorithm 2]

(Algorithm for Inversion in $\text{GF}(2^m)$ Based on Extended Euclid's Algorithm [8])

$$S(x) := G(x); V(x) := 0;$$

$$R(x) := A(x); U(x) := 1;$$

while $\text{deg}(R(x)) \neq 0$ **do**

$$\delta := \text{deg}(S(x)) - \text{deg}(R(x));$$

if $\delta < 0$ **then**

$$\text{temp} := S(x); S(x) := R(x); R(x) := \text{temp};$$

$$\text{temp} := V(x); V(x) := U(x); U(x) := \text{temp};$$

$$\delta := -\delta;$$

end if

$$S(x) := S(x) - x^\delta \times R(x);$$

$$V(x) := V(x) - x^\delta \times U(x);$$

end while

output $U(x)$ as the result.

$$(U(x) = A^{-1}(x))$$

□

Note that, in Algorithm 1, since calculation of $U_j(x)$ is independent of calculation of $W_j(x)$, inversion can be calculated without calculation of $W_j(x)$. Note also that, $R_j(x)$ and $U_j(x)$ can be calculated by the two most recent polynomials of the series $R_j(x)$ and $U_j(x)$, respectively. Therefore, Algorithm 2 keeps only four polynomi-

| $R(x)$ | $S(x)$ | $U(x)$ | $V(x)$ | δ |
|---------------------------|---------------------------|---------------|---------------------|----------|
| $x^6 + x^4$ | $x^7 + x^6 + x^3 + x + 1$ | 1 | 0 | |
| $x^6 + x^4$ | $x^6 + x^5 + x^3 + x + 1$ | 1 | x | 1 |
| $x^6 + x^4$ | $x^5 + x^4 + x^3 + x + 1$ | 1 | $x + 1$ | 0 |
| $x^5 + x^4 + x^3 + x + 1$ | $x^5 + x^2 + x$ | $x + 1$ | $x^2 + x + 1$ | -1 |
| $x^5 + x^4 + x^3 + x + 1$ | $x^4 + x^3 + x^2 + 1$ | $x + 1$ | x^2 | 0 |
| $x^4 + x^3 + x^2 + 1$ | 1 | x^2 | $x^3 + x + 1$ | -1 |
| 1 | $x^3 + x^2 + 1$ | $x^3 + x + 1$ | $x^6 + x^3 + x + 1$ | -4 |

Figure 2. An example of inversion by Algorithm 2 ($m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$ and $A(x) = x^4 + x^2$).

als, $R(x)$, $S(x)$, $U(x)$, and $V(x)$, that correspond to $R_j(x)$, $R_{j-1}(x)$, $U_j(x)$, and $U_{j-1}(x)$, respectively. Figure 2 shows an example of inversion by Algorithm 2, where $m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$ and $A(x) = x^4 + x^2$.

Next, we explain the algorithm for inversion in $\text{GF}(2^m)$ based on the extended Euclid's algorithm proposed by Brunner et al. [7]. Although the algorithm is developed for VLSI implementation, we can use it to develop an algorithm suitable for implementation using the polynomial multiply instruction on $\text{GF}(2)$ as we will describe in the next section. For VLSI implementation, the algorithm tests only the m -th coefficients of two polynomials in the calculation of GCD, thus the polynomials are multiplied by some power of x relative to proper ones. The algorithm is as follows, where $\{O1, O2\}$ means that two operations, $O1$ and $O2$, are performed in parallel. r_m and s_m denote the m -th coefficients of $R(x)$ and $S(x)$, respectively. δ holds the difference of $\deg^*(R(x))$ and $\deg^*(S(x))$, where $\deg^*(\cdot)$ denotes the upper bound of the degree of the proper one.

[Algorithm 3]

(VLSI Algorithm for Inversion in $\text{GF}(2^m)$ [7])

```

 $S(x) := G(x); V(x) := 0;$ 
 $R(x) := A(x); U(x) := 1;$ 
 $\delta := 0;$ 
for  $i = 1$  to  $2m$  do
  if  $r_m = 0$  then
     $R(x) := x \times R(x);$ 
     $U(x) := x \times U(x);$ 
     $\delta = \delta + 1;$ 
  else
    if  $s_m = 1$  then
       $S(x) := S(x) - R(x);$ 
       $V(x) := V(x) - U(x);$ 
    end if
     $S(x) := x \times S(x);$ 
    if  $\delta = 0$  then
       $\{R(x) := S(x), S(x) := R(x)\};$ 
       $\{U(x) := x \times V(x), V(x) := U(x)\};$ 
       $\delta := 1;$ 
    else

```

```

     $U(x) := U/x;$ 
     $\delta := \delta - 1;$ 
  end if
end if
end for

```

output $U(x)$ as the result.
 $(U(x) = A^{-1}(x))$

□

Although the VLSI algorithm proposed in [7] calculates $U(x)$ and $V(x)$ with modulo $G(x)$ arithmetics, we do not need modulo reductions by using $(m + 1)$ -bit registers to hold $U(x)$ and $V(x)$. Figure 3 shows an example of inversion by Algorithm 3, where $m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$, and $A(x) = x^4 + x^2$. When it is implemented in software directly, the algorithm needs multiword shifts in each iteration because shift amount in each iteration is always 1-bit.

2.3 Polynomial Multiply Instruction on $\text{GF}(2)$

In this paper, we consider the typical multiply instruction on $\text{GF}(2)$ that was proposed in [3–6]. We call it MULGF2 instruction as in [3]. MULGF2 instruction calculates the 2-word product from two 1-word operands, rs and rt , and writes the product to two special registers, HI and LO as shown in Fig.4.

A polynomial multiplier on $\text{GF}(2)$ can be very simply realized as an AND-array followed by an XOR-tree, i.e. “carry-free” version of an integer multiplier. Since there are no carry propagation, the multiplier is fast and small. In addition, we can combine this multiplier with an integer

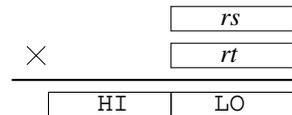


Figure 4. MULGF2 instruction.

| i | $R(x)$ | $S(x)$ | $U(x)$ | $V(x)$ | δ |
|-----|-------------------------------|-------------------------------|-------------------|-------------------------------|----------|
| | $x^6 + x^4$ | $x^7 + x^6 + x^3 + x + 1$ | 1 | 0 | 0 |
| 1 | $x^7 + x^5$ | $x^7 + x^6 + x^3 + x + 1$ | x | 0 | 1 |
| 2 | $x^7 + x^5$ | $x^7 + x^6 + x^4 + x^2 + x$ | 1 | x | 0 |
| 3 | $x^7 + x^6 + x^5 + x^3 + x^2$ | $x^7 + x^5$ | $x^2 + x$ | 1 | 1 |
| 4 | $x^7 + x^6 + x^5 + x^3 + x^2$ | $x^7 + x^4 + x^3$ | $x + 1$ | $x^2 + x + 1$ | 0 |
| 5 | $x^7 + x^6 + x^5 + x^3$ | $x^7 + x^6 + x^5 + x^3 + x^2$ | x^3 | $x + 1$ | 1 |
| 6 | $x^7 + x^6 + x^5 + x^3$ | x^3 | x^2 | $x^3 + x + 1$ | 0 |
| 7 | x^4 | $x^7 + x^6 + x^5 + x^3$ | $x^4 + x^2 + x$ | x^2 | 1 |
| 8 | x^5 | $x^7 + x^6 + x^5 + x^3$ | $x^5 + x^3 + x^2$ | x^2 | 2 |
| 9 | x^6 | $x^7 + x^6 + x^5 + x^3$ | $x^6 + x^4 + x^3$ | x^2 | 3 |
| 10 | x^7 | $x^7 + x^6 + x^5 + x^3$ | $x^7 + x^5 + x^4$ | x^2 | 4 |
| 11 | x^7 | $x^7 + x^6 + x^4$ | $x^6 + x^4 + x^3$ | $x^7 + x^5 + x^4 + x^2$ | 3 |
| 12 | x^7 | $x^7 + x^5$ | $x^5 + x^3 + x^2$ | $x^7 + x^6 + x^5 + x^3 + x^2$ | 2 |
| 13 | x^7 | x^6 | $x^4 + x^2 + x$ | $x^7 + x^6$ | 1 |
| 14 | x^7 | x^7 | $x^3 + x + 1$ | $x^7 + x^6$ | 0 |

Figure 3. An example of inversion by Algorithm 3 ($m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$ and $A(x) = x^6 + x^4$).

multiplier with little cost as described in [10, 11].

3 An Algorithm for Inversion in $\text{GF}(2^m)$ Suitable for Implementation Using a Polynomial Multiply Instruction on $\text{GF}(2)$

In this section, we propose an algorithm for inversion in $\text{GF}(2^m)$ suitable for implementation using a polynomial multiply instruction on $\text{GF}(2)$. The algorithm is based on Algorithm 3, and processes multiple bits in each iteration for fast calculation.

First, we replace the operation “ $U(x) := U(x)/x$,” with the operation “ $V(x) := x \times V(x)$,” in Algorithm 3. By this modification, the final result will become “ $U(x) = x^m \times A^{-1}(x)$ ”, instead of the desired value “ $A^{-1}(x)$ ” [12].

Next, we describe a matrix which represents operations corresponding to several contiguous iterations of Algorithm 3. In Algorithm 3, $R(x)$, $S(x)$, $U(x)$, and $V(x)$ are calculated according to the values of the m -th coefficients of $R(x)$ and $S(x)$. In the same way, we can decide the operations for updating $R(x)$, $S(x)$, $U(x)$, and $V(x)$ in several contiguous iterations of Algorithm 3 according to the values of several coefficients of $R(x)$ and $S(x)$. Therefore, we represent the operations to update $R(x)$, $S(x)$, $U(x)$, and $V(x)$ in several contiguous iterations in a matrix, as proposed in [13] for Montgomery modular inversion case.

The operations in k times iteration of Algorithm 3 can be calculated as follows, where $H(x)$ is the (2×2) matrix which is obtained by coefficients from degree $(m - k + 1)$ to m of $R(x)$ and $S(x)$, and the elements in $H(x)$ are

polynomials on $\text{GF}(2)$ with degree k or less.

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := H(x) \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix}; \quad (1)$$

$$\begin{pmatrix} U(x) \\ V(x) \end{pmatrix} := H(x) \times \begin{pmatrix} U(x) \\ V(x) \end{pmatrix}; \quad (2)$$

We explain how to obtain the matrix $H(x)$ by using an example, where $m = 7$, $G(x) = x^7 + x^6 + x^3 + x + 1$, $A(x) = x^6 + x^4$, and $k = 3$. In this case, the initial values of variables are $R(x) = x^6 + x^4$, $S(x) = x^7 + x^6 + x^3 + x + 1$, $U(x) = x$, and $V(x) = 0$. In the first iteration of Algorithm 3, the following operations are executed:

$$\begin{aligned} R(x) &:= x \times R(x); \\ U(x) &:= x \times U(x); \end{aligned}$$

These operations can be represented in matrices as follows:

$$\begin{aligned} \begin{pmatrix} R(x) \\ S(x) \end{pmatrix} &:= \begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix}; \\ \begin{pmatrix} U(x) \\ V(x) \end{pmatrix} &:= \begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} U(x) \\ V(x) \end{pmatrix}; \end{aligned}$$

By the above operations, we get $R(x) = x^7 + x^5$, $S(x) = x^7 + x^6 + x^3 + x + 1$, $U(x) = x^2$, and $V(x) = 0$. In the same way, in the next iteration of Algorithm 3, the following operations are executed:

$$\begin{aligned} S(x) &:= x \times (S(x) + R(x)); \\ V(x) &:= x \times (U(x) + V(x)); \end{aligned}$$

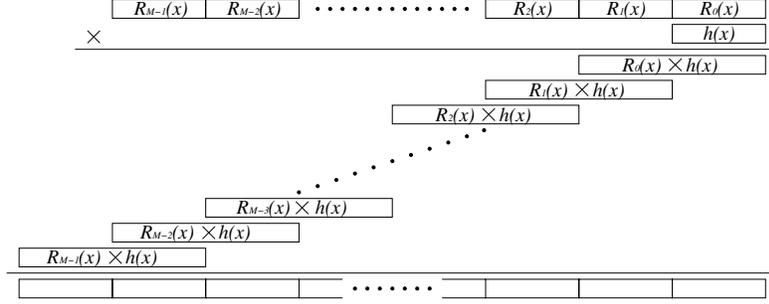


Figure 5. (multi-word × single-word)-multiplication.

These operations can be represented in matrices as follows:

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := \begin{pmatrix} 1 & 0 \\ x & x \end{pmatrix} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix};$$

$$\begin{pmatrix} U(x) \\ V(x) \end{pmatrix} := \begin{pmatrix} 1 & 0 \\ x & x \end{pmatrix} \times \begin{pmatrix} U(x) \\ V(x) \end{pmatrix};$$

By the above operations, we get $R(x) = x^7 + x^5$, $S(x) = x^7 + x^6 + x^4 + x^2 + x$, $U(x) = x$, and $V(x) = x^2$. Then, in the next iteration of Algorithm 3, the following operations are executed:

$$\{R(x) := x \times (S(x) + R(x)), S(x) := R(x)\};$$

$$\{U(x) := x \times (V(x) + U(x)), V(x) := U(x)\};$$

These operations can be represented in matrices as follows:

$$\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := \begin{pmatrix} x & x \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix};$$

$$\begin{pmatrix} U(x) \\ V(x) \end{pmatrix} := \begin{pmatrix} x & x \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} U(x) \\ V(x) \end{pmatrix};$$

The operations in the above three iterations of Algorithm 3 can be calculated at once by using the following matrix $H(x)$.

$$\begin{aligned} H(x) &= \begin{pmatrix} x^3 + x^2 & x^2 \\ x & 0 \end{pmatrix} \\ &= \begin{pmatrix} x & x \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ x & x \end{pmatrix} \times \begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix} \end{aligned}$$

For the word-level description of the algorithm, we partition a polynomial representation on $\text{GF}(2^m)$ into polynomials with degree $(w-1)$. Since the degree of a polynomial representation is up to m , we can represent an element in $\text{GF}(2^m)$ by polynomials with degree $(w-1)$ as follows:

$$R(x) = R_{M-1}(x)x^{(M-1)w} + \dots + R_1(x)x^w + R_0(x)$$

$$R_i(x) = r_{iw+w-1}x^{w-1} + \dots + r_{iw+1}x + r_{iw},$$

where $M = \lceil (m+1)/w \rceil$ and $r_j = 0$ for $j > m$. The matrix $H(x)$, whose elements are polynomials on $\text{GF}(2)$ with

the degree less than w , can be calculated from $R_{M-1}(x)$ and $S_{M-1}(x)$ by only single-word operations, where w is the word size of a processor.

Equations (1) and (2) include (multi-word × single-word)-multiplication. The multiplication is calculated as follows.

$$\begin{aligned} R(x) \times h(x) &= (R_{M-1}(x)x^{(M-1)w} + \dots + R_1(x)x^w + R_0(x)) \times h(x) \\ &= R_{M-1}(x) \times h(x)x^{(M-1)w} + R_{M-2}(x) \times h(x)x^{(M-2)w} \\ &\quad + \dots + R_1(x) \times h(x)x^w + R_0(x) \times h(x) \end{aligned}$$

Using MULGF2 instruction this calculation is calculated by M MULGF2 and $(M-1)$ word XOR instructions as shown in Fig. 5. Therefore, we can calculate them efficiently by using MULGF2 instruction.

An algorithm for inversion in $\text{GF}(2^m)$ suitable for implementation using a polynomial multiply instruction on $\text{GF}(2)$ is as follows. In the algorithm, we assume that the degrees of the initial values of $R(x)$ and $S(x)$ are $(Mw-1)$ instead of m . This assumption does not affect the result.

[Algorithm 4]

(The Proposed Algorithm)

```

S(x) := G(x); V(x) := 0;
R(x) := A(x); U(x) := x;
M := ⌈(m+1)/w⌉;
deg_r := Mw - 1;
deg_s := Mw - 1;
while deg_r > 0 do
  C(x) := R_{M-1}(x);
  D(x) := S_{M-1}(x);
  if C(x) = 0 then
    R(x) := x^w × R(x);
    U(x) := x^w × U(x);
    deg_r := deg_r - w;
  else
    H(x) :=  $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ ;
    j := 1;

```

```

while  $j < w$  and  $deg\_r > 0$  do
   $j := j + 1$ ;
  if  $c_{w-1} = 0$  then
     $C(x) := x \times C(x)$ ;
     $H(x) := \begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix} \times H(x)$ ;
     $deg\_r := deg\_r - 1$ ;
  else
    if  $deg\_r = deg\_s$  then
       $deg\_r := deg\_r - 1$ ;
      if  $d_{w-1} = 1$  then
         $temp := C(x)$ ;
         $C(x) := x \times (C(x) - D(x))$ ;
         $D(x) := temp$ ;
         $H(x) := \begin{pmatrix} x & x \\ 1 & 0 \end{pmatrix} \times H(x)$ ;
      else
         $temp := C(x)$ ;
         $C(x) := x \times D(x)$ ;
         $D(x) := temp$ ;
         $H(x) := \begin{pmatrix} 0 & x \\ 1 & 0 \end{pmatrix} \times H(x)$ ;
      end if
    else
       $deg\_s := deg\_s - 1$ ;
      if  $d_{w-1} = 1$  then
         $D(x) := x \times (C(x) - D(x))$ ;
         $H(x) := \begin{pmatrix} 1 & 0 \\ x & x \end{pmatrix} \times H(x)$ ;
      else
         $D(x) := x \times D(x)$ ;
         $H(x) := \begin{pmatrix} 1 & 0 \\ 0 & x \end{pmatrix} \times H(x)$ ;
      end if
    end if
  end while
   $\begin{pmatrix} R(x) \\ S(x) \end{pmatrix} := H(x) \times \begin{pmatrix} R(x) \\ S(x) \end{pmatrix}$ ;
   $\begin{pmatrix} U(x) \\ V(x) \end{pmatrix} := H(x) \times \begin{pmatrix} U(x) \\ V(x) \end{pmatrix}$ ;
end if
end while

```

output $\begin{cases} U(x)/x^{Mw} & \text{if } deg_r = 0 \\ V(x)/x^{Mw} & \text{otherwise} \end{cases}$ *as the result.*

□

Note that, we initialize $U(x)$ to x instead of 1. Hereby, since a result becomes $A^{-1}(x) \times x^{Mw}$ we can avoid a multi-word shift at the end of the algorithm. Although it would seem that $U(x)$ and $V(x)$ need $2M$ -word, since $U(x)$ and $V(x)$ are just multiplied by some power of x , $U(x)$ and

$V(x)$ are $(M+1)$ -word or less. Figure 6 shows an example of inversion by Algorithm 4, where $m = 7$, $w = 4$, $G(x) = x^7 + x^6 + x^3 + x + 1$, and $A(x) = x^4 + x^2 + x$.

4 Evaluation

We have evaluated the proposed algorithm by comparing the number of MULGF2 and XOR instructions of it with that of Algorithm 2. We assume that MULGF2 instruction has a single-cycle latency. We count the number of MULGF2 and XOR instructions in the following operations in Algorithm 2, since they need (multi-word \times single-word)-multiplication.

$$\begin{aligned} R(x) &:= R(x) - x^j \times S(x); \\ U(x) &:= U(x) - x^j \times V(x); \end{aligned}$$

In the same way, we count the number of MULGF2 and XOR instructions in the equations (1) and (2), in Algorithm 4. In addition, we also count the number of MULGF2 and XOR instructions in operations for calculating $H(x)$ which is in the form

$$H(x) := \begin{pmatrix} h_{00}(x) & h_{01}(x) \\ h_{10}(x) & h_{11}(x) \end{pmatrix} \times H(x);$$

Table 1 shows the number of MULGF2 and XOR instructions of the above operations. These figures are the average of inversion of 1000 random elements, and we used NIST-recommended irreducible polynomials [2]. In the case where $w = 32$ and $w = 16$, Algorithm 4 can calculate inversion faster than Algorithm 2 in almost all m on the average. Especially, in the case where m and w are 571 and 32, respectively, the algorithm calculates inversion with about the half number of instructions of Algorithm 2 on the average. Table 1 also shows no advantage when either the number of words or the word size is small. In these cases, it seems that the cost of the calculation for the matrix $H(x)$ is bigger than the benefit from the calculation at once through the matrix.

5 Discussion

In the proposed algorithm, from the most significant words of $R(x)$ and $S(x)$, we can obtain the matrix $H(x)$ whose elements are polynomials on GF(2) with degree w or less. Therefore, modifying a polynomial multiply instruction to calculate $w \times (w+1)$ -bit multiplication, the proposed algorithm will become faster.

We can reduce the calculation time for the matrix $H(x)$ by using a look-up table. In this case, the table has $w \cdot 2^{2(w-1)}$ entries, and each entry is $4w \cdot \lceil \log_2 w \rceil$ -bit. For large w , we can obtain the matrix $H(x)$ by calculating it from a smaller table instead of direct table look-up. For

| $R(x)$ | $S(x)$ | $U(x)$ | $V(x)$ | $C(x)$ | $D(x)$ | deg_r | deg_s | $H(x)$ |
|-------------------------------|---------------------------|---|-------------------|-----------------|-----------------|---------|---------|--|
| $x^6 + x^4$ | $x^7 + x^6 + x^3 + x + 1$ | x | 0 | $x^2 + 1$ | $x^3 + x^2$ | 7 | 7 | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ |
| | | | | $x^3 + x$ | $x^3 + x^2$ | 6 | 7 | $\begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix}$ |
| | | | | $x^3 + x$ | $x^3 + x^2$ | 6 | 6 | $\begin{pmatrix} x & 0 \\ x^2 & x \\ x^3 + x^2 & x^2 \\ 0 & 0 \end{pmatrix}$ |
| $x^7 + x^6 + x^5 + x^3 + x^2$ | $x^7 + x^5$ | $x^4 + x^3$ | x^2 | $x^3 + x^2 + x$ | $x^3 + x$ | 5 | 6 | $\begin{pmatrix} x & 0 \\ 1 & 0 \\ 0 & 1 \\ x & x \end{pmatrix}$ |
| | | | | $x^3 + x^2 + x$ | x^3 | 5 | 5 | $\begin{pmatrix} 1 & 0 \\ 0 & x \\ x & x \end{pmatrix}$ |
| | | | | x^3 | $x^3 + x^2$ | 4 | 5 | $\begin{pmatrix} x^2 + x & x \\ 1 & 0 \\ x^2 + x & x^2 \\ x^3 + x^2 + x & x^3 \end{pmatrix}$ |
| $x^7 + x^6 + x^5 + x^3$ | x^3 | x^6 | $x^7 + x^5 + x^4$ | $x^3 + x^2 + x$ | 0 | 4 | 4 | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & x \\ 1 & 0 \end{pmatrix}$ |
| | | | | 0 | $x^3 + x^2 + x$ | 3 | 4 | $\begin{pmatrix} 0 & x \\ 1 & 0 \\ 0 & x^2 \\ 1 & 0 \end{pmatrix}$ |
| | | | | 0 | $x^3 + x^2 + x$ | 2 | 3 | $\begin{pmatrix} 0 & x^3 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$ |
| x^6 | $x^7 + x^6 + x^5 + x^3$ | $x^{10} + x^8 + x^7$ | x^6 | x^2 | $x^3 + x^2 + x$ | 1 | 3 | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ x & 0 \end{pmatrix}$ |
| | | | | x^3 | $x^3 + x^2 + x$ | 0 | 2 | $\begin{pmatrix} x & 0 \\ 0 & 1 \end{pmatrix}$ |
| x^7 | $x^7 + x^6 + x^5 + x^3$ | $\frac{x^{11} + x^9 + x^8}{\Rightarrow (x^{11} + x^9 + x^8)/x^8 = x^3 + x + 1}$ | x^6 | | | 0 | 3 | |

Figure 6. An example of inversion by Algorithm 4 ($m = 7$, $w = 4$, $G(x) = x^7 + x^6 + x^3 + x + 1$, and $A(x) = x^4 + x^2$).

Table 1. Comparison of Algorithm 2 and Algorithm 4 by Using MULGF2 instruction.

| m | w | M | Algorithm 2 | | | Algorithm 4 | | | Algorithm 4 Algorithm 2 [%] |
|-----|-----|-----|-------------|-----------|------------|-------------|-----------|------------|--------------------------------|
| | | | #MULGF2 | #XOR | Total | #MULGF2 | #XOR | Total | |
| 163 | 8 | 21 | 2551.990 | 5751.183 | 8303.173 | 4429.690 | 7435.794 | 11865.484 | 142.90 |
| | 16 | 11 | 1341.678 | 2900.299 | 4241.977 | 1776.748 | 2374.208 | 4150.956 | 99.45 |
| | 32 | 6 | 733.744 | 1475.552 | 2209.296 | 1092.291 | 939.702 | 2031.993 | 91.97 |
| | 64 | 3 | 437.356 | 781.916 | 1219.272 | 861.820 | 498.054 | 1359.874 | 111.53 |
| 233 | 8 | 30 | 5171.630 | 11782.952 | 16954.582 | 8498.470 | 14869.154 | 23367.624 | 137.82 |
| | 16 | 15 | 2686.084 | 5938.826 | 8624.910 | 2967.834 | 4339.090 | 7306.924 | 84.72 |
| | 32 | 8 | 1433.177 | 3005.143 | 4438.320 | 1608.673 | 1563.986 | 3172.659 | 71.48 |
| | 64 | 4 | 811.507 | 1550.121 | 2361.628 | 1189.397 | 772.566 | 1961.963 | 83.08 |
| 283 | 8 | 36 | 7606.275 | 17408.760 | 25015.035 | 12114.088 | 21526.934 | 33641.022 | 134.45 |
| | 16 | 18 | 3929.496 | 8770.623 | 12700.119 | 4046.623 | 6190.736 | 10237.359 | 80.61 |
| | 32 | 9 | 2069.663 | 4420.616 | 6490.279 | 1903.184 | 2005.679 | 3908.863 | 60.23 |
| | 64 | 5 | 1150.530 | 2265.673 | 3416.203 | 1519.226 | 1002.331 | 2521.557 | 73.81 |
| 409 | 8 | 52 | 15796.999 | 36401.858 | 52198.857 | 24358.389 | 44466.958 | 68825.347 | 131.85 |
| | 16 | 26 | 8090.381 | 18302.942 | 26393.323 | 7594.821 | 12405.278 | 20000.099 | 75.78 |
| | 32 | 13 | 4196.562 | 9195.477 | 13392.039 | 3167.822 | 3729.663 | 6897.485 | 51.50 |
| | 64 | 7 | 2262.231 | 4673.211 | 6935.442 | 2246.086 | 1664.397 | 3910.483 | 56.38 |
| 571 | 8 | 72 | 30726.338 | 71092.647 | 101818.985 | 45937.205 | 85235.844 | 131173.049 | 128.83 |
| | 16 | 36 | 15624.255 | 35652.145 | 51276.400 | 13611.722 | 23396.826 | 37008.548 | 72.17 |
| | 32 | 18 | 8019.095 | 17888.957 | 25908.052 | 5130.520 | 6652.412 | 11782.932 | 45.48 |
| | 64 | 9 | 4220.452 | 9012.505 | 13232.957 | 3055.222 | 2579.882 | 5635.104 | 42.58 |

example, in the case where $m = 7$, $w = 8$, $G(x) = x^7 + x^6 + x^3 + x + 1$, and $A(x) = x^4 + x^2 + x$. First, according to coefficients from degree 4 to 7 of $C(x)$ and $D(x)$, we can obtain the matrix

$$H_1(x) = \begin{pmatrix} x^3 + x^2 & x^2 \\ x & 0 \end{pmatrix}$$

from the smaller table. Then, we can update $C(x)$, $D(x)$ using $H_1(x)$, and obtain the matrix

$$H_2(x) = \begin{pmatrix} x^2 + x & x^2 \\ x^3 + x^2 + x & x^3 \end{pmatrix}.$$

Finally, we can calculate $H(x)$ by $H_1(x)$ and $H_2(x)$ as follows:

$$\begin{aligned} H(x) &= H_2(x) \times H_1(x) \\ &= \begin{pmatrix} x^5 & x^4 + x^3 \\ x^6 + x^4 + x^3 & x^5 + x^4 + x^3 \end{pmatrix} \end{aligned}$$

6 Concluding Remarks

We have proposed an algorithm for inversion in $\text{GF}(2^m)$ suitable for implementation using a polynomial multiply instruction on $\text{GF}(2)$. In the algorithm, operations corresponding to several contiguous iterations of the VLSI algorithm proposed by Brunner et al. is represented as a matrix. They are calculated at once through the matrix. In the case where the word size of a processor and m are 32 and 571, respectively, the algorithm calculates inversion with about the half number of polynomial multiply instructions on $\text{GF}(2)$ and XOR instructions for the conventional algorithm for software implementation on the average. We can accelerate the proposed algorithm by using look-up table.

References

- [1] IEEE P1363 Working Group, *IEEE P1363/D13 (Draft Version 13): Standard Specifications for Public Key Cryptography*, Nov. 1999. [Online]. Available: <http://grouper.ieee.org/groups/1363/>
- [2] National Institute of Standards and Technology, *FIPS PUB 186-2: Digital Signature Standard (DSS)*. pub-NIST, Jan. 2000. [Online]. Available: <http://www.itl.nist.gov/fipspubs/fip186-2.pdf>
- [3] J. Großschädl and E. Saveş, "Instruction set extensions for fast arithmetic in finite fields $\text{GF}(p)$ and $\text{GF}(2^m)$," in *Proceeding of Cryptographic Hardware and Embedded Systems – CHES 2005*, Aug.29–Sept.1 2005, pp. 133–147.
- [4] A. M. Fiskiran and R. B. Lee, "Evaluating instruction set extensions for fast arithmetic on binary finite fields," in *Proceeding of International Conference on Application-specific Systems, Architectures and Processors – ASAP 2004*, Sept.27-29 2004, pp. 125–136.
- [5] S. Tillich and J. Großschädl, "Accelerating AES using instruction set extensions for elliptic curve cryptography," in *International Conference on Computational Science and Its Applications – ICCSA 2005*, 2005, pp. 665–674.
- [6] H. Eberle, A. Wander, N. Gura, and S. Chang-Shantz, "Architectural extensions for elliptic curve cryptography over $\text{GF}(2^m)$," *Sun Labs Open House*, 2004. [Online]. Available: <http://research.sun.com/sunlabsday/docs.2004/Micro.pdf>
- [7] H. Brunner, A. Curiger, and M. Hofstetter, "On computing multiplicative inverses in $\text{GF}(2^m)$," *IEEE Trans. Comput.*, vol. 42, no. 8, pp. 1010–1015, Aug. 1993.
- [8] D. Hankerson, J. Hernandez, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," in *Proceeding of Cryptographic Hardware and Embedded Systems – CHES 2000*, Aug.17–18 2000, pp. 1–24.
- [9] IEEE P1363 Working Group, *IEEE P1363/D13 (Draft Version 13): Standard Specifications for Public Key Cryptography, Annex A: Number-Theoretic Background*, Nov. 1999. [Online]. Available: <http://grouper.ieee.org/groups/1363/>
- [10] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, pp. 449–460, Apr 2003.
- [11] J. Garcia and M. J. Schulte, "A combined 16-bit binary and dual galois field multiplier," in *Signal Processing Systems, 2002. (SPIS '02). IEEE Workshop on*, Oct.16–18 2002, pp. 63–68.
- [12] J. Guo and C. Wang, "Systolic array implementation of Euclid's algorithm for inversion and division in $\text{GF}(2^m)$," *IEEE Trans. Comput.*, vol. 47, no. 10, pp. 1161–1167, Oct. 1998.
- [13] T. Kobayashi and H. Morita, "Fast modular inversion algorithm to match any operation unit," *IEICE Trans. Fund.*, vol. E82-A, no. 5, pp. 733–740, May 1999.