COMMUN. MATH. SCI.
Vol. 9, No. 1, pp. 277–287

AN IMPROVEMENT OF THE TYT ALGORITHM FOR $GF(2^M)$ BASED ON REUSING INTERMEDIATE COMPUTATION RESULTS*

YIN LI†, GONG-LIANG CHEN‡, YI-YANG CHEN§, AND JIAN-HUA LI¶

Abstract. Multiplicative inversion plays an important role to Elliptic Curve Cryptosystems. This paper presents an efficient inversion algorithm in $GF(2^m)$ using a normal basis which improves the Itoh-Tsujii (IT) algorithm and the Takagi et al. (TYT) algorithm . The proposed algorithm reduces the number of required multiplications by decomposing $m-1$ into several factors plus a remainder and by reusing intermediate computation values. It is proved that the decomposition of $m-1$ can be made simpler, but requires even fewer multiplications. Furthermore, a practical algorithm for finding an optimal decomposition of $m-1$ is investigated.

Key words. Multiplicative inverse, IT algorithm, TYT algorithm, optimal decomposition.

AMS subject classifications. 12E30, 68Q10.

1. Introduction

Arithmetic operations in $GF(2^m)$ are usually required in many areas such as public-key cryptography and error-correcting codes [1, 2]. A significant example is offered by the Elliptic Curve Cryptosystem (ECC) [3], which is based on the group of points of an elliptic curve defined over $GF(2^m)$. The basic operation in ECC, i.e., the point addition, is typically expressed in terms of field additions — field multiplications plus a single inversion. However, inversion is much more expensive than other operations in $GF(2^m)$ and many researchers attempt to develop efficient algorithms for carrying out such operation.

The inversion computation is usually carried out using approaches based either on the Extended Euclidean algorithm [4, 5] and its derivatives [6, 7], or Fermat's little theorem [8, 9, 10, 11]. Specially, Fermat's little theorem implies that, for any nonzero value $\alpha \in GF(2^m)$, $\alpha^{-1} = \alpha^{2^m-2}$. If we use α^{2^m-2} instead of α^{-1}, the inversion computation can be performed by iterative squaring operations and multiplications, requiring $m-1$ squaring operations and $m-2$ multiplications [8].

Normal basis representation is usually applied in hardware implementation of finite field arithmetic. In normal basis representation, squaring in $GF(2^m)$ can be accomplished in terms of a simple cyclic shift of the coefficients, which can be realized as a simple rewiring and does not cost any logic gates. Thus, to speed up the inversion computation, it is necessary to reduce the number of required multiplications.

The Itoh-Tsujii [9] (IT) algorithm reduced the number of required multiplications to $\lfloor \log_2(m-1) \rfloor + w(m-1) - 1$ through a clever construction of an addition chain, where $w(m-1)$ denotes the Hamming weight of the binary representation of $m-1$. Takagi et al. [12] (TYT) further improved the IT algorithm by factoring $m-1$ into several factors plus a small remainder and then applied a similar approach to each

*Received: April 7, 2010; accepted (in revised version): July 28, 2010. Communicated by Shi Jin.
†School of Information Security Engineering, Shanghai Jiaotong University, Shanghai, P.R. China (woajian@sjtu.edu.cn).
‡School of Information Security Engineering, Shanghai Jiaotong University, Shanghai, P.R. China (chengl@sjtu.edu.cn).
§School of Information Security Engineering, Shanghai Jiaotong University, Shanghai, P.R. China (channing@sjtu.edu.cn).
¶Department of Electronic Engineering, Shanghai Jiaotong University, Shanghai, P.R. China (lijh888@sjtu.edu.cn).

factor. As a result, the proposed algorithm necessitates even fewer multiplications than the IT algorithm. Chang et al. [13] and Li et al. [14] further extended the TYT algorithm to the composite fields $GF((2^n)^m)$ and $GF(2^m)$ using polynomial representation, respectively.

In this paper, we propose an improved version of the TYT algorithm using a normal basis. Our approach is also based on the strategy which decomposes $m-1$ into several factors plus a remainder, but reduces the required multiplications further by reusing some intermediate computation results. Our approach allows for a more flexible decomposition of $m-1$ to minimize the multiplications required to perform the inversion. Consequently, for some values of m, our algorithm can utilize the simplest decomposition of $m-1$ and requires even fewer multiplications than the original TYT algorithm. Furthermore, a feasible algorithm for finding optimal decomposition of $m-1$ is also discussed.

The rest of this paper is organized as follows: in Section 2, we briefly recall the IT algorithm and the TYT algorithm. Then, an improved version of the TYT algorithm is proposed in Section 3. In Section 4, some propositions about finding optimal decompositions are presented and a related algorithm is discussed. Then, a comparison of our proposal with some others is made in Section 5. Finally, some conclusions are drawn.

2. Previous algorithms and related concepts

In this section, we briefly review the IT algorithm and the TYT algorithm. A normal basis of $GF(2^m)$ over $\mathbb{F}_2$ is defined as follows: for an nonzero element $x \in GF(2^m)$, if $x, x^{2^1}, x^{2^2}, \cdots, x^{2^{m-1}}$ are linearly independent, then the set $\{x, x^{2^1}, x^{2^2}, \cdots, x^{2^{m-1}}\}$ is called a normal basis [1, 2]. We can represent any element $A \in GF(2^m)$ as $A = \sum_{i=0}^{m-1} \alpha_i x^{2^i} = (\alpha_0, \alpha_1, \cdots, \alpha_{m-1})$ where $\alpha_i \in \mathbb{F}_2$. From Fermat's little theorem, $x^{2^m} = x$ holds. Then we have

$$\begin{aligned} A^2 &= (\alpha_0 x + \alpha_1 x^2 + \cdots + \alpha_{m-1} x^{2^{m-1}})^2 \\ &= \alpha_0 x^2 + \alpha_1 x^{2^2} + \cdots + \alpha_{m-1} x^{2^m} \\ &= \alpha_{m-1} x + \alpha_0 x^2 + \cdots + \alpha_{m-2} x^{2^{m-1}}. \end{aligned}$$

Hence, the squaring operation over a normal basis can be easily accomplished through a right cyclic shift of its coefficients.

Let $\beta \in GF(2^m)$ be an arbitrary nonzero element. Based on Fermat's little theorem, the inversion can be carried out as $\beta^{-1} = \beta^{2^m-2} = (\beta^{1+2+2^2+\cdots+2^{m-2}})^2$. Since we work with a normal basis, the cost of the squaring operation can be ignored and it suffices to reduce the number of required multiplications. The IT algorithm [9] computed the exponentiation through repeated squaring of intermediate results (cyclic shift) and multiplication. Consequently, the IT algorithm only requires $\lfloor \log_2 m \rfloor + w(m-1) - 1$ multiplications. This approach can be generalized by the following lemma:

LEMMA 2.1. *[12] Let $\beta \in GF(2^m)^*$ and $t = 1 + 2^a + (2^a)^2 + \cdots + (2^a)^{b-1}$, $(a, b \in \mathbb{N})$. Then there exists an algorithm for computing β^t, which requires $\lfloor \log_2(b) \rfloor + w(b) - 1$ multiplications. Here, $w(b)$ denotes the Hamming weight of the binary representation of b.*

Proof. First, consider the computation of β^{s_k} where

$$s_k = \sum_{i=0}^{2^k-1} (2^a)^i = 1 + 2^a + (2^a)^2 + \cdots + (2^a)^{2^k-1}.$$

Then, there exists a recursion:

$$\begin{aligned}\beta^{s_k} &= \beta^{1+2^a+(2^a)^2+\cdots+(2^a)^{2^k-1}} \\ &= \beta^{1+2^a+\cdots+(2^a)^{2^{k-1}-1}} \cdot \beta^{(2^a)^{2^{k-1}}+\cdots+(2^a)^{2^k-1}} \\ &= (\beta^{s_{k-1}})^{(2^a)^{2^{k-1}}} \cdot \beta^{s_{k-1}}. \end{aligned} \tag{2.1}$$

Hence, if β^{s_k} is computed then, for any s_i less than s_k, β^{s_i} will also be computed. Note that $\beta^{s_0}=\beta$, so that the computation of $\beta^{s_k},\beta^{s_{k-1}},\cdots,\beta^{s_0}$ only requires k multiplications.

Let $b=\sum_{i=1}^{n}2^{k_i}$, with $k_1>k_2>\cdots>k_n$. Then the exponentiation β^t can be rewritten as follows:

$$\beta^t=(\beta^{s_{k_n}})\left(\cdots(\beta^{s_{k_3}})((\beta^{s_{k_2}})(\beta^{s_{k_1}})^{(2^a)^{2^{k_2}}})^{(2^a)^{2^{k_3}}}\cdots\right)^{(2^a)^{2^{k_n}}}. \tag{2.2}$$

Since $k_1>k_i$ for $i=2,\cdots,n$, if we compute $\beta^{s_{k_1}}$ as above, then all the $\beta^{s_{k_i}}$ for $i=2,\cdots,n$ will also be computed, as above. The computation of $\beta^{s_{k_1}}$ requires only $k_1=\lfloor\log_2(b)\rfloor$ multiplications. Note that it still needs $w(b)-1$ more multiplications to go through Equation (2.2). Adding up the two partial complexities, we directly obtain the conclusion. □

Takagi et al.[12] reduced the number of required multiplications using a factorization approach. They first decomposed $m-1$ into several factors plus a small remainder and then applied Lemma 2.1 to each factor. Assume that $m-1=\prod_{i=1}^{k}r_i+h$ and we have the following equation:

$$\begin{aligned}2^m-2 &= 2^{m-1}+2^{m-2}+\cdots+2^{m-h}+2^{m-h}-2 \\ &= \textstyle\sum_{m-h}^{m-1}2^i+2(2^{\prod_{i=1}^{k}r_i}-1) \\ &= \textstyle\sum_{m-h}^{m-1}2^i+2(2^{\prod_{i=1}^{k-1}r_i}-1)\cdot\Big((2^{\prod_{i=1}^{k-1}r_i})^{r_k-1}+ \\ &\qquad +(2^{\prod_{i=1}^{k-1}r_i})^{r_k-2}+\cdots+(2^{\prod_{i=1}^{k-1}r_i})+1\Big) \\ &\cdots \\ &= \textstyle\sum_{m-h}^{m-1}2^i+2(2^{r_1}-1)\cdot\big((2^{r_1})^{r_2-1}+\cdots+(2^{r_1})+1\big)\cdots \\ &\qquad \Big((2^{\prod_{i=1}^{k}r_i})^{r_t-1}+(2^{\prod_{i=1}^{k}r_i})^{r_t-2}+\cdots+(2^{\prod_{i=1}^{k}r_i})+1\Big).\end{aligned}$$

Then the inversion can be computed as

$$\begin{aligned}\beta^{-1}=\beta^{2^m-2}=(\textstyle\prod_{i=1}^{h}\beta^{2^{m-i}})\times(\beta^{2^{m-h-1}-1})^2=(\prod_{i=1}^{h}\beta^{2^{m-i}})\times \\ \beta^{2(2^{r_1}-1)((2^{r_1})^{r_2-1}+\cdots+(2^{r_1})+1)\cdots((2^{\prod_{i=1}^{k-1}r_i})^{r_k-1}+\cdots+(2^{\prod_{i=1}^{k-1}r_i})+1)}.\end{aligned} \tag{2.3}$$

Note that 2^i-th power in the normal basis can be carried out by a simple cyclic shift, so that one can compute the inversion efficiently through the Equation (2.3) and iterative application of Lemma 2.1. In consequence, we only need

$$\sum_{i=1}^{k}(\lfloor\log_2(r_i)\rfloor+w(r_i)-1)+h$$

multiplications to compute the inversion. If the first factor r_1 is decomposed further, one can just follow the same lines as above to compute $\beta^{2^{r_1}-1}$.

Moreover, since there exist several decompositions of $m-1$, Takagi et al. have defined a notion about optimal decomposition which minimizes the number of required multiplications and consists of the fewest computation components. For some values of m, the TYT algorithm even requires fewer multiplications than the IT algorithm.

3. New algorithm

In this section, we will show that more multiplications can be saved during the computation process with a slight modification of Equation (2.3). Then, an improved version of the TYT inversion algorithm based on this inspiration is proposed.

3.1. Main strategy and the algorithm.

Since

$$\begin{aligned} 2^m-2 &= 2^{m-1}+2^{m-2}+\cdots+2^{m-h}+2^{m-h}-2 \\ &= (2^{m-h}(2^h-1))+2(2^{m-h-1}-1), \end{aligned}$$

we have

$$\begin{aligned} \beta^{-1} &= \beta^{2^m-2} \\ &= (\beta^{2^h-1})^{2^{m-h}} \times (\beta^{2^{m-h-1}-1})^2. \end{aligned}$$

Using techniques similar to those presented in the proof of Lemma 2.1, we can save more field multiplications in the computation of β^{2^h-1}. Assume that $m-h-1=\prod_{i=1}^{k} r_i$; then

$$\beta^{2^m-2} = (\beta^{2^h-1})^{2^{m-h}} \times (\beta^{2^{m-h-1}-1})^2 = (\beta^{2^h-1})^{2^{m-h}} \times \left(\beta^{(2^{r_1}-1)((2^{r_1})^{r_2-1}+\cdots+(2^{r_1})+1)\cdots((2^{\prod_{i=1}^{k-1} r_i})^{r_k-1}+\cdots+(2^{\prod_{i=1}^{k-1} r_i})+1)}\right)^2. \tag{3.1}$$

Consider two following cases:

Case 1: If r_1 is not decomposed further.

Assume that $r_1=\sum_{i=1}^{n} 2^{u_i}$ with $u_1>u_2>\cdots>u_n$ and $h=\sum_{i=1}^{\ell} 2^{t_i}$ with $t_1>t_2>\cdots>t_\ell$. Based on the relevant techniques of Lemma 2.1, we calculate s intermediate values:

$$\{\beta^{2^{2^0}-1}, \beta^{2^{2^1}-1}, \cdots, \beta^{2^{2^s}-1}\},$$

where $s=\max\{u_1,t_1\}$. Similar to Equation (2.2), we obtain two equations as follows:

$$\beta^{2^{r_1}-1} = (\beta^{s_{u_n}})\left(\cdots(\beta^{s_{u_3}})((\beta^{s_{u_2}})(\beta^{s_{u_1}})^{2^{2^{u_2}}})^{2^{2^{u_3}}}\cdots\right)^{2^{2^{u_n}}} \tag{3.2}$$

and

$$\beta^{2^h-1} = (\beta^{s_{t_\ell}})\left(\cdots(\beta^{s_{t_3}})((\beta^{s_{t_2}})(\beta^{s_{t_1}})^{2^{2^{t_2}}})^{2^{2^{t_3}}}\cdots\right)^{2^{2^{t_\ell}}}. \tag{3.3}$$

If $r_1 \geqslant h$, we have $u_1 \geqslant t_1$ and $s=u_1=\lfloor \log_2(r_1) \rfloor$ [1]. Then it requires $w(r_1)-1$ and $w(h)-1$ multiplications to go through Equation (3.2) and Equation (3.3), respectively. Therefore, besides the computation of $\beta^{2^{r_1}-1}$, we only need $w(h)-1$ multiplications to compute $(\beta^{2^h-1})^{2^{m-h}}$ while the original approach needs $h-1$.

[1] Such a case can be ensured by selecting suitable r_1 and h.

Furthermore, note that the number of required multiplications corresponding to other factors r_i, $(i=2,\cdots,s)$ is $\lfloor\log_2(r_i)\rfloor+w(r_i)-1$. Then we still need one more multiplication, of $(\beta^{2^h-1})^{2^{m-h}}$ by $\beta^{2^{m-h}-2}$, to obtain the final result. Therefore the number of multiplications required for the inversion computation is

$$\sum_{i=1}^{k}(\lfloor\log_2(r_i)\rfloor+w(r_i)-1)+w(h). \tag{3.4}$$

Case 2: If r_1 is decomposed further.

Assume that r_1 is decomposed as $\prod_{i=1}^{l}t_i+h'$. Then if h' is less than one of the factors in $\{t_1,\cdots,t_l\}$, the main computation strategy with respect to $\beta^{2^{h'}-1}$ will follow the same line as that presented in Case 1. It also costs $w(h)-1$ multiplications. The number of multiplications required by inversion in this case can be obtained by using Equation (3.4) iteratively.

Based on the previous consideration, we can propose a new version of TYT algorithm. Assume that $m-1$ is decomposed into $\sum_{i=0}^{k}r_i+h$. Let $\left[r_{q_i}^{(i)}r_{q_i-1}^{(i)}\cdots r_1^{(i)}r_0^{(i)}\right]_2$ denote the binary representation of r_i $(i=1,2,\cdots,k)$ and $[h_\ell h_{\ell-1}\cdots h_1h_0]_2$ denote the binary representation of h. We first present the pseudo-code of the computation of $\beta^{2^{m-1}-1}$ as follows:

Procedure: Main computation block of new TYT algorithm

Inputs: β and $m-1$
Auxiliary: $(z_0,z_1,z_2,\cdots,z_q)$ where $q=\max\{q_1,q_2,\cdots,q_k\}$.
Outputs: $\beta^{2^{m-1}-1}$
Step 1: $z_0:=\beta$;
Step 2: if r_1 is decomposed further;
Step 3: call this procedure recursively to compute $\beta^{2^{r_1}-1}$ and β^{2^h-1};
Step 4: else for $i=1$ to q_1;
Step 5: $z_i\leftarrow(z_{i-1})^{2^{i-1}}$, $z_i\leftarrow z_i\cdot z_{i-1}$;
Step 6: $\theta\leftarrow z_{q_1}$, $\eta\leftarrow z_\ell$;
Step 7: for $i=q_1-1$ to 0;
Step 8: if $r_i^1=1$ then $\theta\leftarrow\theta^{2^{2^i}}\cdot z_i$; // $\beta^{2^{r_1}-1}$;
Step 9: if $h_i=1$ then $\eta\leftarrow\eta^{2^{2^i}}\cdot z_i$; // β^{2^h-1};
Step 10: $n\leftarrow r_1$, $z_0\leftarrow\theta$;
Step 11: for $j=2$ to k;
Step 12: for $i=1$ to q_j;
Step 13: $z_i\leftarrow(z_{i-1})^{(2^n)^{i-1}}$, $z_i\leftarrow z_i\cdot z_{i-1}$;
Step 14: for $i=q_j-1$ to 0;
Step 15: if $r_i^j=1$ then $\theta\leftarrow\theta^{(2^n)^{2^i}}\cdot z_i$;
Step 16: $z_0\leftarrow\theta$, $n\leftarrow n\cdot r_j$;
Step 17: return $\theta\cdot(\eta)^{2^{m-h-1}}$;

In the end, we only need to calculate the square of $\beta^{2^{m-1}-1}$ and then obtain the inversion of α.

Since $w(h)\le h$, under the same decomposition of $m-1$, our approach would cost fewer multiplications. In practical application, Takagi et al. usually select the remainder h as 1 to minimize the cost of $\prod_{i=1}^{h}\beta^{2^{m-i}}$ in Equation (2.3). However, this restricts the choice of the decomposition corresponding to $m-1$. Conversely, our ap-

proach only costs $w(h)$ multiplications to calculate $\prod_{i=1}^{h}\beta^{2^{m-i}}$. Hence, the choice of h can be more flexible and lead to even more efficient computation.

3.2. An example. As a small example, consider an inversion in $GF(2^{123})$. Let $A\in GF(2^{123})$ be an arbitrary nonzero element. According to Fermat's little theorem, $A^{-1}=(A^{2^{122}-1})^2$. The original TYT algorithm decomposes 122 into $2\times(3\times 20+1)$ and computes the inversion as follows:

$$\begin{aligned} A^{2^{123}-2} &= (A^{2^{122}-1})^2 \\ &= \left((A^{2^{61}-1})^{(2^{61}+1)}\right)^2 \\ &= \left(\left(A^{2^{60}}\cdot(A^{2^{20}-1})^{((2^{20})^2+2^{20}+1)}\right)^{(2^{61}+1)}\right)^2. \end{aligned} \tag{3.5}$$

It requires 9 multiplications, which saves 1 multiplication compared with the IT algorithm. Conversely, we decompose 122 into $3\times 40+2$ and compute:

$$\begin{aligned} A^{2^{123}-2} &= (A^{2^{122}-1})^2 \\ &= \left((A^{2^{122}}\cdot A^{2^{121}})\cdot(A^{2^{40}-1})^{((2^{40})^2+2^{40}+1)}\right)^2 \\ &= \left((A^2\cdot A)^{2^{121}}\cdot(A^{2^{40}-1})^{((2^{40})^2+2^{40}+1)}\right)^2. \end{aligned} \tag{3.6}$$

Note that $2<40$ and $w(2)=1$, we can reuse the intermediate values in computation of $A^{2^{40}-1}$ to calculate $A^2\cdot A$. Consequently, our approach also requires 9 multiplications. Note that Equation (3.6) has fewer computation components than Equation (3.5), which would be more attractive in hardware implementation. More details are presented in Table 3.1.

Power	Rule	Result
$s_0=2^{2^0}-1$	-	A
$s_1=2^{2^1}-1$	$(A^{s_0})^{2^{2^0}}\cdot A^{s_0}$	A^{1+2}
$s_2=2^{2^2}-1$	$(A^{s_1})^{2^{2^1}}\cdot A^{s_1}$	$A^{1+2+2^2+2^3}$
$s_3=2^{2^3}-1$	$(A^{s_2})^{2^{2^2}}\cdot A^{s_2}$	$A^{1+2+\cdots+2^7}$
$s_4=2^{2^4}-1$	$(A^{s_3})^{2^{2^3}}\cdot A^{s_3}$	$A^{1+2+\cdots+2^{15}}$
$s_5=2^{2^4}-1$	$(A^{s_4})^{2^{2^4}}\cdot A^{s_4}$	$A^{1+2+\cdots+2^{31}}$
$s_0'=2^{40}-1$	$(A^{s_5})^{2^{2^3}}\cdot A^{s_3}$	$A^{1+2+\cdots+2^{39}}$
$s_1'=(2^{40})^2-1$	$(A^{s_0'})^{(2^{40})}\cdot A^{s_0'}$	$A^{1+2+\cdots+2^{79}}$
$(2^{40})^3-1$	$(A^{s_1'})^{(2^{40})}\cdot A^{s_0'}$	$r_1=A^{1+2+\cdots+2^{119}}$
$2^{122}-1$	$(A^{s_1})^{2^{121}}\cdot r_1$	$r_2=A^{1+2+\cdots+2^{120}+2^{121}}$
$2^{123}-2$	$(r_2)^2$	A^{-1}

TABLE 3.1. *The inversion computation of A*

4. Optimal decomposition

Since the number of multiplications with respect to the TYT algorithm depend on the decomposition of $m-1$, Takagi et al. have proposed the notion of optimal decomposition as that which minimizes the number of required multiplications and consists

of the fewest computational components. However, they have only summarized some related propositions. In this section, we will try to propose an applicable algorithm about finding optimal decomposition corresponding to our algorithm. Firstly, some lemmas and propositions are proposed.

LEMMA 4.1. *Let* $a=bc$, $a,b,c\in\mathbb{N}$; *then* $\lfloor\log_2 a\rfloor\leq\lfloor\log_2 b\rfloor+\lfloor\log_2 c\rfloor+1$.

Proof. Assume that $2^n\leqslant b<2^{n+1}$ and $2^m\leqslant c<2^{m+1}$, then $2^{n+m}\leqslant ab<2^{n+m+2}$ which concludes the lemma. □

LEMMA 4.2. *[12] If an integer* t *is decomposed as* $2^n\cdot s$, *the computation of* β^{2^t-1}, *such that* $\beta\in GF(2^m)^*$, *using the TYT algorithm costs the same number of multiplications as using the IT algorithm.*

Proof. See Section 3 in [12]. □

PROPOSITION 4.3. *In the optimal decomposition of* $m-1=\prod_{i=1}^{k}r_i+h$, *the remainder* h *should satisfy* $w(h)<w(m-1)-2$.

Proof. Let $m-1$ be 2^n+c such that $c<2^n$. The decomposition of $m-1$ as 2^n+c requires $n+w(c)$ multiplications, which is identical to that required by the IT algorithm, i.e., $\lfloor\log_2(m-1)\rfloor+w(m-1)-1=n+w(c)$. Thus in the optimal decomposition of $m-1$, the remainder h should be smaller than c.

Assume therefore that $m-1=\prod_{i=1}^{k}r_i+h$ and r_1 is not decomposed further. The number of required multiplications under this decomposition is

$$\sum_{i=1}^{k}(\lfloor\log_2(r_i)\rfloor+w(r_i)-1)+w(h). \tag{4.1}$$

Based on Lemma 4.2, it is obvious that $w(r_i)\geq 2$ for $i=1,\cdots,k$. Hence, $\lfloor\log_2(r_i)\rfloor+w(r_i)-1\geq\lfloor\log_2 r_i\rfloor+1$. Note that $h<c$, so that $\lfloor\log_2(m-1)\rfloor=\lfloor\log_2(m-1-h)\rfloor=\lfloor\log_2\prod_{i=1}^{k}r_i\rfloor$. Applying Lemma 4.1 iteratively, we have

$$\begin{aligned}\lfloor\log_2(m-1)\rfloor &= \lfloor\log_2\textstyle\prod_{i=1}^{k}r_i\rfloor\\ &\leq \lfloor\log_2\textstyle\prod_{i=2}^{k}r_i\rfloor+\lfloor\log_2 r_1\rfloor+1\\ &\leq \lfloor\log_2\textstyle\prod_{i=3}^{k}r_i\rfloor+\lfloor\log_2 r_2\rfloor+1+\lfloor\log_2 r_1\rfloor+1\\ &\cdots\\ &\leq \textstyle\sum_{i=1}^{k}(\lfloor\log_2 r_i\rfloor+1)-1.\end{aligned} \tag{4.2}$$

Consequently, $\lfloor\log_2(m-1)\rfloor\leq\sum_{i=1}^{k}(\lfloor\log_2(r_i)\rfloor+w(r_i)-1)-1$. Since the number of required multiplications here should be less than that of the IT algorithm, we have

$$\lfloor\log_2(m-1)\rfloor+w(m-1)-1>\sum_{i=1}^{k}(\lfloor\log_2(r_i)\rfloor+w(r_i)-1)+w(h).$$

Hence $w(h)<w(m-1)-2$. Moreover, in this case it is obvious that the optimal decomposition corresponding to $m-1$ requires at least $\lfloor\log_2 m-1\rfloor+1$ multiplications.

If r_1 is decomposed further, based on analysis similar to that above, the optimal decomposition corresponding to r_1 costs at least $\lfloor\log_2 r_1\rfloor+1$ multiplications. We plug this inequality to Equation (4.2) to conclude the proposition. □

LEMMA 4.4. *[12] When $w(m-1)=2$, the optimal decomposition is $m-1$ itself and the number of required multiplications is $\lfloor \log_2(m-1) \rfloor + 1$.*

Proof. See Section 3 in [12]. □

Based on previous propositions and lemmas, we could present an applicable algorithm for finding the optimal decomposition of $m-1$. To facilitate description, we use the symbol $\psi(n)$ to denote the number of required multiplications in the computation of β^{2^n-1}, for $n \in \mathbb{N}$. The algorithm is stated as follows:

Algorithm *Opdecompose*(n)
(*produce an optimal decomposition of* n)
if $w(n) \leqslant 2$ **then** *return* n
else
for $i=0$ *to* $\lceil \frac{n}{4} \rceil$ *while* $w(i) < w(n)-2$ **do**
compute $Factor(n-i)$ *and save the relative decomposition*
end for
end if
choose the $n-i, i$ *such that the* $\psi(n-i)+w(i)$ *is minimal and return the decomposition*
end *Opdecompse.*

where

Algorithm $Factor(t)$
(*produce the factorization of* t *that minimize* $\psi(t)$)
if $w(t) \leq 2$ *or* t *is a prime number* **then** *return* t
end if
factorize t *as* $r_1, r_2, \cdots, r_\ell$ *which ensures that* $\psi(t)$ *is minimal and that* r_1 *is the maximal number*
if $w(r_1) \geq 3$ **then** *return* $Opdecompose(r_1), r_2, \cdots, r_\ell$
else *return* $r_1, r_2, \cdots, r_\ell$
end if
end *Factor.*

REMARK 4.5. In the procedure of $Opdecompose(n)$, we only search for the remainder i from 0 to $\lceil \frac{n}{4} \rceil$. Clearly, in the previous section we assumed that the remainder i should be less than one factor of $n-i$. Note that the optimal decomposition of $n-i$ does not include a power of 2 as an independent factor. Thus we have $i \leq \frac{n-i}{3}$ and then $i \leq \lceil \frac{n}{4} \rceil$.

We can use a greedy algorithm to factorize the operand in $Factor(t)$. The main strategy is to find the two locally optimal factors in each iteration with the hope of finding the global optimum. After several iterations, we could find a factorization which decreases $\psi(t)$.

Procedure Factorization
Inputs: an integer t
Auxiliary: Q_0, Q_1 and Q_2
Outputs: factorization of t to decrease $\psi(t)$
Step 1: if $w(t) \leq 2$ or t is a prime number then return t;
Step 2: $Q_0 \leftarrow \lfloor \log_2(t) \rfloor + w(t) - 1$;
Step 3: for $i=3$ to $\sqrt{t}$ while $i|t$
Step 4: if $\lfloor \log_2(i) \rfloor + w(i) + \lfloor \log_2 \left(\frac{t}{i}\right) \rfloor + w\left(\frac{t}{i}\right) - 2 \leq Q_0$ then

Step 5: $Q_0 \leftarrow \lfloor \log_2(i) \rfloor + w(i) + \lfloor \log_2\left(\frac{t}{i}\right) \rfloor + w\left(\frac{t}{i}\right) - 2$;
Step 6: $Q_1 \leftarrow i,\ Q_2 \leftarrow \frac{t}{i}$;
Step 7: recursively call this procedure to factorize Q_1 and Q_2;
Step 8: return the factors;

Clearly, the algorithm *Opdecompose*(n) is based on exhaustive approach using previous propositions. Indeed, we have used the MAPLE software to implement the algorithm. It is shown that the optimal decomposition with respect to the number less than 1000 can be found after at most two recursions.

5. Comparison

In this section, we illustrates the improvement obtained with our algorithm compared with original TYT algorithm and the IT algorithm, by collecting some numerical results for m with cryptographic size. These results are summarized in Table 5.1. It is shown that our algorithm uses relative simpler decompositions of $m-1$ but obtains the same improvement as the original TYT algorithm. Moreover, for some values of m, our algorithm even requires fewer multiplications.

m	$m-1$	TYT decomposition	New decomposition	#Mul. [IT]	#Mul. [TYT]	#Mul. [New]
123	122	$2\times(20\times3+1)$	$3\times40+2$	10	9	9
187	186	$6\times(10\times3+1)$	$5\times34+16$	11	10	10
238	237	$3\times(26\times3+1)$	$3\times68+33$	12	11	11
328	327	$3\times(9\times12+1)$	$5\times65+2$	12	11	11
489	488	$8\times(3\times20+1)$	$3\times160+8$	12	11	11
672	671	$11\times(20\times3+1)$	$131\times5+1$	15	13	13
189	188	4×47	$5\times36+8$	11	11	10
396	395	$5\times(6\times13+1)$	$3\times129+8$	12	12	11
384	383	$2\times(38\times5+1)+1$	$3\times5\times25+8$	15	13	12
471	470	47×10	$9\times52+2$	13	13	12
428	427	$7\times(3\times20+1)$	$17\times25+2$	13	12	11
429	428	4×107	$3\times132+32$	12	12	11
500	499	$6\times83+1$	$3\times161+16$	13	12	11

TABLE 5.1. *Comparison of different approaches for multiplicative inversion*

In [15, 16], Cruz-Cortés et al. generalized the IT algorithm using the concept of an addition chain. They argued that obtaining the smallest number of multiplications required to perform inversion is equivalent to the problem of finding the shortest addition chain (SAC) starting with 1 and ending with $m-1$. They utilized the Artificial Immune System (AIS) paradigm to obtain their result.

In fact, finding the shortest addition chain for an integer is an open problem. Now one can get the lengths of the shortest addition chains for all of the integers [2] up to 2^{31}. Considering the cryptographic applications, we have computed all the optimal decomposition of the number between [160,1000]. It is shown that there exist only 105 numbers where the optimal decompositions corresponding to these numbers are greater than the length of SAC (equal to the length of SAC plus one). On the other hand, the optimal decomposition corresponding to other numbers are equal to

[2] `http://wwwhomes.uni-bielefeld.de/achim/addition_chain.html`

the length of SAC. However, our approach would be more attractive in hardware implementation, since the structure of our algorithm is similar to the IT algorithm and it can utilize the former architecture with a slight modification.

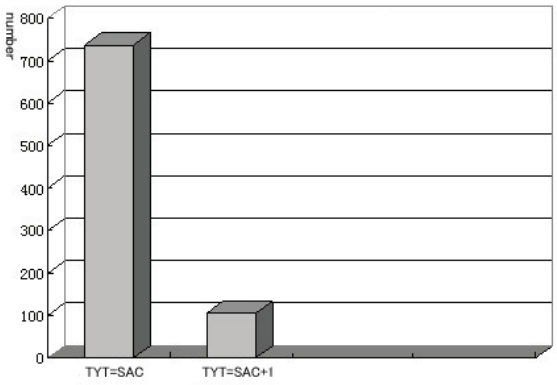

FIG. 5.1. *The distribution of the optimal decomposition*

6. Conclusion

We have proposed an improvement of TYT algorithm for multiplicative inversion in $GF(2^m)$. Analogously to the original algorithm, the parameter $m-1$ is also decomposed into several factors and a remainder. But the decomposition used in our approach takes into account of reuse of intermediate values in the computation. Consequently, our algorithm even requires fewer multiplications than the TYT algorithm and the IT algorithm for some values of m. The new decomposition principle proposed is a good supplement for optimal decomposition mentioned in [12].

The proposed algorithm can be easily modified for multiplicative inversion in $GF(p^m)$, where p is an odd prime. Related results are also attractive for multiplicative inversion using a polynomial basis or a dual basis.

Acknowledgement. The authors would like to thank the anonymous reviewers for their useful and valuable comments. This work is supported by the Natural Science Foundation of China (No.60672068).

REFERENCES

[1] R. Lidl and H. Niederreiter, *Finite Fields*, Cambridge University Press, New York, NY, USA, 1983.
[2] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, Cambridge University Press, New York, NY, USA, 1994.
[3] I. Blake, G.Seroussy and N.P. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, Cambridge, 1999.
[4] J. Guo and C. Wang, *Systolic array implementation of Euclid's algorithm for inversion and division in* $GF(2^m)$, IEEE Trans. Comput., 47(10), 1161–1167, 1998.
[5] D. Hankerson, J.L. Hernandez and A. Menezes, *Software implementation of elliptic curve cryptography over binary fields*, In CHES '00, Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems, London, UK, Springer, 1–24, 2000.
[6] Z. Yan and D.V. Sarwate, *New systolic architectures for inversion and division in* $GF(2^m)$, IEEE Trans. Comput., 52(11), 1514–1519, 2003.

[7] A.P. Fournaris and O. Koufopavlou, *Applying systolic multiplication-inversion architectures based on modified extended Euclidean algorithm for $GF(2^k)$ in elliptic curve cryptography*, Comput. Electr. Eng., 33(5–6), 333–348, 2007.

[8] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura and I.S. Reed, *VLSI architectures for computing multiplications and inverses in $GF(2^m)$*, IEEE Trans. Comput., 34(8), 709–717, 1985.

[9] T. Itoh and S. Tsujii, *A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases*, Inf. Comput., 78(3), 171–177, 1988.

[10] J. Guajardo and C. Paar, *Itoh-Tsujii inversion in standard basis and its application in cryptography*, Codes. Des. Codes Cryptography, 25(2), 207–216, 2002.

[11] F. Rodríguez-Henríquez, G. Morales-Luna, N. Saqib and N. Cruz-Cortés, *Parallel Itoh-Tsujii multiplicative inversion algorithm for a special class of trinomials*, Des. Codes Cryptography, 45(1), 19–37, 2007.

[12] N. Takagi, J.I. Yoshiki and K. Takagi, *A fast algorithm for multiplicative inversion in $GF(2^m)$ using normal bases*, IEEE Trans. Comput., 50(5), 394–398, 2001.

[13] K.Y. Chang, H. Kim, J.S. Kang and H.S. Cho, *An extension of TYT algorithm for $GF((2^n)^m)$ using precomputation*, Inf. Process. Lett., 92(5), 231–234, 2004.

[14] Y. Li, G.L. Chen, Y.Y. Chen and J.H. Li, *An extension of TYT inversion algorithm in polynomial basis*, Inf. Process. Lett., 110(8–9), 300–303, 2010.

[15] N. Cruz-Cortés, F. Rodríguez-Henríquez and C.A. Coello, *On the optimal computation of finite field exponentiation*, Advances in Artificial Intelligence - IBERAMIA 2004, 9th Ibero-American Conference on AI, LNCS, 3315, 747–756, 2004.

[16] N. Cruz-Cortés, F. Rodríguez-Henríquez and C.A. Coello, *An artificial immune system heuristic for generating short addition chains*, IEEE Trans. Evol. Comput., 12(1), 1–24, 2008.