A Fast Implementation of η_T Pairing in Characteristic Three on Intel Core 2 Duo Processor

MITSUNARI Shigeo *

January 14, 2009

Abstract

We present an efficient implementation of η_T pairing on Intel Core 2 Duo processor. The processing speed of our implementation achieves 92 μsec over $\mathbb{F}_3^{97}$ and 553 μsec over $\mathbb{F}_3^{193}$ on 2.6GHz processor.

Keywords: η_T pairing, parallel computing, efficient implementation, SSE

1 Introduction

Weil and Tate pairings are important tools for generating many interesting encryption protocols. An η_T pairing proposed by Barreto *et al.* [1] is one of the fastest pairings over $\mathbb{F}_{3^m}$. This paper describes an efficient improvement of the η_T pairing algorithm in characteristic three for Intel Core 2 Duo processor.

First, we try to improve a multiplication of $\mathbb{F}_{3^m}$, which is the most timing-consuming operation in the η_T pairing. We deal with the left-to-right Comb [2] method for the multiplication. The word size parameter for the Comb method is usually selected to 32 or 64 according to CPU to minimize the number of addition and byte shift. Note that a byte shift is faster than an addition using SSE on Intel Core 2 Duo processor, and moreover a bit shift is very slower than

*Cybozu Labs, Inc.

addition on the same platform. In this paper we therefore modify the left-to-right Comb method to reduce the number of bit shift by increasing the number of addition.

Let A, B be an m-degree polynomial of $\mathbb{F}_3[X]$, and consider a multiplication $C \leftarrow A \cdot B$, where the resulting C is an $2m$-degree polynomial. The left-to-right Comb method shifts B and adds it to C base on the coefficient of A. In order to implement a shifting B we require to split B into two registers of word size, which is relatively slow in software implementation. Note that Intel Core 2 Duo processor equips an efficient byte rotation (faster than addition). We then modify the Comb method to rotate the resulting C instead of B, and then the number of slow bit shift operations can be decreased. These modifications make the speed of multiplication about twice faster.

The second improvement is to deploy the two cores of Intel Core 2 Duo processor. The Duursma-Lee loop of the η_T pairing has an auxiliary variable depending on the previous computed variables. However we can split Duursma-Lee loop into two loops of half size by precomputing the sequence of cubes (and cube roots) of input variables. Then Intel Core 2 Duo processor can simultaneously compute each half loop respectively, and the time of Duursma-Lee loop without the final exponentiation can be improved by twice.

We implemented the η_T pairing according these improvements and the speed achieves 92 μsec over $\mathbb{F}_3^{97}$ and 553 μsec over $\mathbb{F}_3^{193}$ on 2.6GHz Intel Core 2 Duo processor. The timing is about four times faster than that of [5], which is the fastest to the best of our knowledge. The full source code of our implementation is available from `http://homepage1.nifty.com/herumi/crypt/pairing.html`.

2 Modified Comb method for SSE

Finite field $\mathbb{F}_{3^m}$ is the set of all polynomials represented by $\mathbb{F}_{3^m} = \mathbb{F}_3[X]/f(X)$, where $f(X)$ is an irreducible polynomial in $\mathbb{F}_3[X]$ of degree m. We select $f(x) = x^{97} + x^{12} + 2$ for $m = 97$ and $f(x) = x^{193} + x^{12} + 2$ for $m = 193$. An element $A(x) \in \mathbb{F}_{3^m}$ can be represented as

$$A(x) = (a_{m-1}, a_{m-2}, \cdots, a_1, a_0), a_i \in \mathbb{F}_3 \text{ for all } i = 0, 1, \cdots, m-1.$$

Let w a word size of CPU (or its divisor), and b be a width size for the precomputed window. Here, we define an element $\tilde{A}_i(x)$ of $\mathbb{F}_{3^m}$ as follows:

$$\tilde{A}_i(x) = (0, \cdots, 0, a_{\min(i+b-1,(\mathbf{ceil}(i/w)+1)w-1)}, \cdots, a_{i+1}, a_i).$$

For example, let $w = 32$ and $b = 3$, then

$$\tilde{A}_0(x) = a_2x^2 + a_1x + a_0,$$

$$\tilde{A}_{27}(x) = a_{29}x^2 + a_{28}x + a_{27},$$

$$\tilde{A}_{30}(x) = a_{31}x + a_{30},$$

$$\tilde{A}_{32}(x) = a_{34}x^2 + a_{33}x + a_{32}.$$

Algorithm 1 is the left-to-right Comb Method in $\mathbb{F}_{3^m}$ [2]. Here tbl$[T(x)]$ is a precomputed table, which consists of $A(x)T(x)$ for all polynomials $T(x) \in \mathbb{F}_{3^m}$ of degree $< b$.

Algorithm 1 : Left-to-right Comb method with window of width b[2]

input: $A(x)$, $B(x) \in \mathbb{F}_{3^m}$
output: $C(x) \leftarrow A(x)B(x)$
1. Compute tbl$[T(x)] \leftarrow A(x)T(x)$ for all $T(x) \in \mathbb{F}_{3^m}$ of degree $< b$
2. $n_1 \leftarrow \mathbf{ceil}(w/b)$
   $n_2 \leftarrow \mathbf{ceil}(m/w)$
3. $C(x) \leftarrow 0$
4. `for` $i \leftarrow 0$ `to` $n_1 - 1$ `do`
5. `for` $j \leftarrow 0$ `to` $n_2 - 1$ `do`
6. $C(x) \leftarrow C(x) + \text{tbl}[\tilde{B}_{iw+j}(x)]x^{jw}$
7. `end for`
8. `if` $j \neq n_1 - 1$ `then` $C(x) \leftarrow C(x)x^b$
9. `end for`

From our experiment, the most efficient implementation of Algorithm 1 on Intel Core 2 Duo processor is to choose $b = 3$ for $m = 97$ and $m = 193$, and thus we fix $b = 3$ in the following. In Step 6 we load a part of precomputed table tbl$[T(x)]$ based on the information of $T(x) = \tilde{B}_{iw+j}(x)$. We show the number of operations of bit shift and addition in Table 1 at Algorithm 1. Computing tbl$[\tilde{B}_{iw+j}(x)]x^{jw}$ requires $(w/8)$-byte shift operation and $C(x) \leftarrow C(x)x^b$ requires b-bit shift.

Ordinary, we select $w = 32$ or 64 because the number of addition is minimum. But, Intel Core 2 Duo processor has 128-bits SSE registers and `palignr` mnemonics which reduces the cost of byte shift among SSE registers.

Acoording to the number of mnemonics of bit/byte shift of temporary $2m$-bit data and addition in Table 2, $2m$-bit shift is very slow, then we selected $w = 8$ to reduce the operation.

Table 1: The number of mnemonics except for move in Algorithm 1

	$m = 97$		$m = 193$	
w	bit shift	add and byte shift	bit shift	add and byte shift
8	2	36	2	72
16	5	36	5	72
32	10	33	10	66

Table 2: The number of mnemonics of operation except for moving by SSE

m	byte shift	byte rotation	bit shift	addition in $\mathbb{F}_{3^m}$
97	4	4	16	6
193	6	6	30	12

Next, to implement an addition between $2m$-bit data and m-bit data in Algorithm 1 requires to split m-bit data into two data and then the number of addtion increases.

Because byte rotation is faster than addition according to Table 2 on Intel Core 2 Duo processor, we modify Algorithm 1 as Algorithm 2 to avoid splitting data. Moreover, by connecting byte shift and bit shift at *l*.9–10 in Algorithm 2, the cost can be more reduced.

3 Parallel computation of loop

We show the main loop of η_T pairing proposed Barreto *et al.* [1] at Algorithm 3. Because x_p and y_p are not dependent on f and g, then we can compute the sequence of cubes and cube roots before computing the loop. And then, we can split the remain loop into two loops of half size (the first half and the latter half). We show the modified algorithm at Algorithm 4.

Because the loops of *l*.2 and *l*.3, *l*.8–12 and *l*.13–17 in Algorithm 4 are independent from each other respectively, then Intel Core 2 Duo processor can simultaneously compute each loop respectively.

4 Results and comparisons

We implemented the η_T pairing according to these improvements and show our timing at Table 3 and compare them with Table 1 [4] for $m = 97$(using SSE) and with Table 6 [5] for $m = 193$. Our timing(single) means the score by one core and our timing(multi) by two cores. Our timing(multi) is about four times faster even if considering the difference of frequency among CPUs.

Algorithm 2 : Modified left-to-right Comb method for SSE

input: $A(x)$, $B(x) \in \mathbb{F}_{3^m}$
output: $C(x) \leftarrow A(x)B(x)$

1. Compute tbl$[T(x)] \leftarrow A(x)T(x)$ for all $T(x) \in \mathbb{F}_{3^m}$ of degree $< b$
2. $n_1 \leftarrow$ **ceil**(w/b) where $w = 8$
   $n_2 \leftarrow$ **ceil**(m/w)
3. $C(x) \leftarrow 0$
4. `for` $i \leftarrow 0$ `to` $n_1 - 1$ `do`
5. `for` $j \leftarrow 0$ `to` $n_2 - 1$ `do`
6. $C(x) \leftarrow C(x) + \text{tbl}[\tilde{B}_{iw+j}(x)]$
7. $C(x) \leftarrow w$-bit right rotation of $C(x)$
8. `end for`
9. $C(x) \leftarrow wn_2$-bit left rotation of $C(x)$
10. `if` $j \neq n_1 - 1$ `then` $C(x) \leftarrow C(x)x^b$
11. `end for`

Algorithm 3 : main loop of η_T pairing[1]

x_p, y_p, x_q, $y_q \in E(\mathbb{F}_{3^m})[l]$; the l-torsion subgroup of $E(\mathbb{F}_{3^m})$
f, $g \in \mathbb{F}_{3^{6m}}$, $\sigma^2 = -1$, $\rho^3 = \rho + 1$

1. `for` $i \leftarrow 0$ `to` $\frac{m-1}{2}$ `do`
2. $u \leftarrow x_p + x_q + 1$
3. $g \leftarrow y_p y_q \sigma - u^2 - u\rho - \rho^2$
4. $f \leftarrow fg$
5. $x_p \leftarrow x_p^{1/3}$, $y_p \leftarrow y_p^{1/3}$, $x_q \leftarrow x_q^3$, $y_q \leftarrow y_q^3$
6. `end for`

Algorithm 4 : modified loop for parallel computation

$x_p[0] \leftarrow x_p$, $y_p[0] \leftarrow y_p$, $x_q[0] \leftarrow x_q$, $y_q[0] \leftarrow y_q$

1. `for` $i \leftarrow 0$ `to` $\frac{m+1}{2} - 1$ `do`
2. $x_p[i+1] \leftarrow x_p[i]^{1/3}$, $x_q[i+1] \leftarrow x_q[i]^3$
3. $y_p[i+1] \leftarrow y_p[i]^{1/3}$, $y_q[i+1] \leftarrow y_q[i]^3$
4. `end for`
5. $n \leftarrow \frac{m+1}{2}$, $f_1 \leftarrow f$
6. $u \leftarrow x_p[n/2] + x_q[n/2] + 1$
7. $f_2 \leftarrow y_p[n/2]y_q[n/2]\sigma - u^2 - u\rho - \rho^2$
8. `for` $i \leftarrow 0$ `to` $n/2 - 1$ `do`
9. $u \leftarrow x_p[i] + x_q[i] + 1$
10. $g \leftarrow y_p[i]y_q[i]\sigma - u^2 - u\rho - \rho^2$
11. $f_1 \leftarrow f_1 g$
12. `end for`
13. `for` $i \leftarrow n/2 + 1$ `to` $n - 1$ `do`
14. $u \leftarrow x_p[i] + x_q[i] + 1$
15. $g \leftarrow y_p[i]y_q[i]\sigma - u^2 - u\rho - \rho^2$
16. $f_2 \leftarrow f_1 g$
17. `end for`
18. $f \leftarrow f_1 f_2$

Table 3: Running times for multiplication and η_T pairing(μsec)

m	97		193	
	mul	pairing	mul	pairing
[4]($m = 97$), [5]($m = 193$)[1]	0.500	479.63	1.822	2611
our timing(single)[2]	0.181	149	0.624	975
our timing(multi)[2]	0.181	92	0.624	553

[1]AMD Opteron 275 (2.2GHz), gcc 4.1.2 using Linux x86 (64-bit)
[2]Intel Core 2 Duo (2.6GHz), Visual Studio 2008 SP1 using Windows Xp (32-bit)

5 Acknowledgment

The authors would like to thank Takagi Tsuyoshi, Jean-Luc Beuchat and Eiji Okamoto for their valuable comments.

References

[1] P. Barreto, S. Galbraith, C. Ó hÉigeartaigh and M. Scott, "Efficient pairing computation on supersingular abelian varieties." Designs, Codes and Cryptography 42(3), pp.239–271, 2007.

[2] D. Hankerson, A. Menezes and S. Vanstone, "Guide to elliptic curve cryptography," Springer-Verlag, 2004.

[3] M. Yoshitomi, T. Takagi, S. Kiyomoto, and T. Tanaka, "Efficient Implementation of the Pairing on Mobilephones using BREW," 20 Aug 2007 (`http://eprint.iacr.org/2007/340`).

[4] M. Shirase, Y. Kawahara, T. Takagi, E. Okamoto, "Universal η_T Pairing Algorithm over Arbitrary Extension Degree," The 8th International Workshop on Information Security Applications, WISA 2007, LNCS 4867, pp.1–15, Springer-Verlag, 2007.

[5] Y. Kawahara, K. Aoki, and T. Takagi, "Faster Implementation of η_T Pairing over GF(3^m) Using Minimum Number of Logical Instructions for GF(3)-Addition," Pairing 2008, LNCS 5209, pp.282–296, 2008.