PAPER Special Section on Information and Communication System Security

Efficient Implementation of Pairing-Based Cryptography on a Sensor Node

Masaaki SHIRASE[†a)], **Yukinori MIYAZAKI**[†], **Tsuyoshi TAKAGI**[†], ***Members***, **Dong-Guk HAN**[††], ***and*** **Dooho CHOI**[†††], ***Nonmembers***

SUMMARY Pairing-based cryptography provides us many novel cryptographic applications such as ID-based cryptosystems and efficient broadcast encryptions. The security problems in ubiquitous sensor networks have been discussed in many papers, and pairing-based cryptography is a crucial technique to solve them. Due to the limited resources in the current sensor node, it is challenged to optimize the implementation of pairings on sensor nodes. In this paper we present an efficient implementation of pairing over MICAz, which is widely used as a sensor node for ubiquitous sensor network. We improved the speed of η_T pairing by using a new efficient multiplication specialized for ATmega128L, called *the block comb method* and several optimization techniques to save the number of data load/store operations. The timing of η_T pairing over $GF(2^{239})$ achieves about 1.93 sec, which is the fastest implementation of pairing over MICAz to the best of our knowledge. From our dramatic improvement, we now have much high possibility to make pairing-based cryptography for ubiquitous sensor networks practical.

key words: *η_T pairing, sensor node, ATmega128L, finite field multiplication, assembly implementation*

1. Introduction

The technology of wireless sensor networks (WSNs) has been implemented in practical applications of ubiquitous society. In general, WSN node has low physical protection and does not have secure memory for cryptographic keys. It is thus important to develop secure solutions to these networks.

Firstly, symmetric cryptosystems have been utilized to propose secure WSNs. However, they face the key distribution problem. Due to that, conventional public key such as RSA and elliptic curve cryptosystem (ECC) are considered as alternative proposals. Public key authentication is typically achieved by means of a public key infra-structure (PKI), which issues certificates and requires exchange and memory of large keys. These operations, however, cause high overheads of memory, computation, and communication and, in consequence, are inadequate for WSNs [21]. Because WSN nodes are low power, has a limited battery, and especially with limited resources.

Motivated by the above reasons, Identity-Based Encryption (IBE) [1] based on bilinear pairings comes into the spotlight because it is an exception where a known information that uniquely recognizes users such as email address can be utilized as a public key and thus PKI is unnecessary. Recently, Oliveira *et al.* argued that IBE is idea for WSNs and vice versa [17], [18]. They discussed the synergy between the systems, describe how WSNs can take advantage of IBE.

To make Pairing-Based Cryptography including IBE become truly practical in WSNs, it is necessary to optimize the performance of pairings which are the most significant operation. In [17] Oliveira *et al.* implemented the Tate pairing of a supersingular elliptic curve over $GF(q)$ with 256-bit q ($q^2 \approx 512$ bits) on MICAz, whose timing is about 30 sec. Moreover, Ishiguro *et al.* implemented an η_T pairing over $GF(3^m)$ in 5.8 sec for $m = 97$ [12]. TinyPBC [18] by Oliveira *et al.* and NanoECC [25] by Szczechowiak *et al.* are implementations of η_T pairing over $GF(2^m)$ for $m = 271$. Currently pairing implementations (TinyPBC, TinyTate) are slower than RSA (TinyPK, [27]) and ECC (TinyECC [14], TinyECCK [23]). Therefore, it is a good research challenge to optimize the implementation of Pairing cryptosystems in resource-constrained sensor nodes.

In this paper, we propose an efficient implementation of η_T pairing over $GF(2^{239})$ on MICAz with ATmega128L processor to compare with previous works [12], [17], [18]. The target η_T pairing consists of Addition, Multiplication, Reduction, Square, Inversion, which form 2.3%, 75%, 6.3%, 4.7%, 4.5% in total computation cost. Namely, multiplication is a dominant operation in the η_T pairing. Thus, we first propose a fast multiplication, called *block comb method*, which can dramatically reduce the number of load/store operation. Actually, it is well known that data load and store are very time consuming operation in sensor node. Due to the proposed block comb method, the timing of a multiplication in $GF(2^{239})$ is improved 1.28 from 4.6 msec.

In addition, we improve the squaring, reduction, and inversion. The shift operation is optimized for ATmega128L, which makes the reduction and squaring faster. The degree function in the inversion is also improved. Consequently, we can compute an η_T pairing over $GF(2^{239})$ on MICAz with ATmega128L processor in about 1.93 sec which is, to the best of our knowledge, the most efficient implementation of PBC primitives for MICAz with ATmega128L processor.

The remainder of this paper is organized as follows. In Sects. 2 and 3, we discuss η_T pairings and its implementa-

Manuscript received July 28, 2008.

[†]The authors are with Future University Hakodate (FUN), Hakodate-shi, 041–8655 Japan.

[††]The author is with the Dept. of Mathematics, Kookmin University, Seoul, Korea.

[†††]The author is with Electronics and Telecommunications Research Institute (ETRI), Korea.

a) E-mail: shirase@fun.ac.jp

DOI: 10.1587/transinf.E92.D.909

tion, and target platform that is MICAz ATmega128L. We propose a new *block comb method* in Sect. 4. The detail implementation and results are presented in Sect. 5. Finally, we conclude in Sect. 6.

2. η_T Pairing and Its Implementation on ATmega128L

In this section, we explain the η_T pairing proposed by Barreto *et al.* and how to compute that pairing. The η_T pairing is a fast pairing, and there has been much research on it [5], [6], [16], [22], [24], [25].

2.1 η_T Pairing over Binary Field

In this section, we explain about the η_T pairing over a binary field. Let $GF(2^m)$ be a binary field with the extension degree m, where m is an odd prime. Let E be a supersingular elliptic curve defined over $GF(2^m)$,

$$E : Y^2 = X^3 + X + b, b \in \{0, 1\},$$

Let l be the largest odd prime with $l \,|\, \#E(GF(2^m))$. Then, the η_T paring proposed by Barreto *et al.* [2] is the mapping,

$$\eta_T : E(GF(2^m))[l] \times E(GF(2^m))[l] \to GF(2^{4m})^*,$$

with the bilinearity, namely $\eta_T(sP, tQ) = \eta_T(P, Q)^{st}$, satisfied for any $P, Q \in E(GF(2^m))$ and any integers s, t. The extension degree 4 of $GF(2^{4m})$ over $GF(2^m)$ is the smallest positive integer such that l divides $(2^{km} - 1)$. Such an integer is called the embedding degree.

An element in the 4−th extension $GF(2^{4m})$ is represented as $a_0 + a_1s + a_2t + a_3st$ for $a_0, a_1, a_2, a_3 \in GF(2^m)$, where $s^2 + s + 1 = 0$ and $t^2 + t + s = 0$.

The η_T pairing is computed by Algorithm 1, modified by Shu *et al.* [25], where M of Step 13 is defined as follows.

$$M = \begin{cases} (2^{2m} - 1)(2^m - 2^{(m+1)/2} + 1)2^m \\ \quad \text{if } (m \bmod 8, b) = (1, 0), (3, 1), (5, 1), (7, 0) \\ (2^{2m} - 1)(2^m + 2^{(m+1)/2} + 1)2^m \\ \quad \text{if } (m \bmod 8, b) = (1, 1), (3, 0), (5, 0), (7, 1) \end{cases}$$

Note that A and C in Algorithm 1 belong to $GF(2^{4m})$, where a set $\{1, s, t, st\}$ is a basis of $GF(2^{4m})$ over $GF(2^m)$ with $t^2 = t + 1, s^2 = s + t$. A powering by M of Step 13 is called the final exponentiation, and is computed using multiplications, squarings, and an inversion. The final exponentiation is efficiently computed by the following equations.

$$\begin{aligned} A^{2^{2m}} &= (a_0 + a_2) + (a_1 + a_3)s + a_2t + a_3st, \\ A^{2^m} &= (a_0 + a_1 + a_2) + (a_1 + a_2 + a_3)s, \\ &\quad + (a_2 + a_3)t + a_3st \end{aligned}$$

where $A = a_0 + a_1s + a_2t + a_3st \in GF(2^{4\cdot m})$.

In this paper, we fix parameters ($m = 239, b = 1$) of the η_T pairing. Let l be the largest odd prime with $l \,|\, \#E(GF(2^{239}))$. The size of l and $GF(2^{4\cdot 239})$ are important values for security. Let E' be an elliptic curve used for the η_T pairing over $GF(3^{97})$, and let l' be the largest odd prime with $l' \,|\, \#E'(GF(3^{97}))$. We know that $l \approx l'$ and $2^{4\cdot 239} \approx 3^{6\cdot 97}$, which means that the security of our implementation is equivalent to that of the implementation in Ref. [5], [6], [9], [12], [13].

It is easy to see that Algorithm 1 takes 885 multiplications, 144 squarings, 1 division, and 3226 additions when $m = 239$.

Algorithm 1 Computing the η_T pairing [25]

```
Input: $P = (\alpha,\beta), Q = (x, y) \in E(GF(2^m))$
Output: $\eta_T(P, Q) \in (GF(2^{4m})^*)$
 1: $C \leftarrow 1$
 2: $\alpha \leftarrow \alpha^2 + 1,\ \beta \leftarrow \beta^2 + 1,\ u \leftarrow y + b + 1,\ v \leftarrow x + 1,\ \theta \leftarrow \alpha v$
 3: for $i \leftarrow 0$ to $\frac{m-1}{2}$ do
 4:     $A \leftarrow \beta + \theta + u + (\alpha + v + 1)s + t$
 5:     $C \leftarrow C^2$
 6:     $C \leftarrow C \cdot A$
 7:     if $i < \frac{m-1}{2}$ then
 8:         $\alpha \leftarrow \alpha^4,\ \beta \leftarrow \beta^4,\ u \leftarrow u + v + 1,\ v \leftarrow v + 1,\ \theta \leftarrow \alpha v$
 9:     end if
10: end for
11: $A \leftarrow A + (\alpha^2 + v + 1) + s$
12: $C \leftarrow C \cdot A$
13: $C \leftarrow C^M$
14: return $C$
```

2.2 Previous Implementations

We report previous known implementations of pairing on ATmega128L.

Oliveira *et al.* implemented the Tate pairing [17]. The implementation is named TinyTate. TinyTate implemented uses a finite field $GF(p)$ (p is a 256 bit prime) and a curve $y^2 = x^3 + x$ with the embedding degree 2. It occupies 18,384 bytes of ROM for program, and 1,831 bytes of memory. The timing is about 30 sec.

Oliveira *et al.* implemented the η_T pairing [18]. The implementation is named TinyPBC. TinyPBC uses a finite field $GF(2^{271})$ and a curve $y^2 + y = x^3 + x^2$ with the embedding degree 4, and occupies 47,948 bytes of ROM for program, and 3,235 bytes of memory (2,867 bytes are used as stack). A multiplication in $GF(2^{271})$ is computed by look-up table and the Karatsuba method, and takes about 4.0 msec. The timing of pairing is about 5.5 sec.

Ishiguro *et al.* implemented the η_T pairing [12]. Their implementation uses a finite field $GF(3^{97})$ and a curve $y^2 = x^3 - x + 1$ with the embedding degree 6, and occupies 17,284 bytes of ROM for program, and 628 bytes of memory. A multiplication in $GF(3^{97})$ is computed by the comb method, and takes about 6.2 msec by the comb method. The timing of pairing is about 5.8 sec.

On the other hand, TinyECCK is the fastest implementation of ECC on ATmega128L by Seo *et al.* [23]. A multiplication in $GF(2^{163})$ is computed by the comb method and the window method with the width 4, and takes 2.9 msec. This multiplication method becomes an object of comparison of our proposed method because this paper also uses binary field.

2.3 Our Initial Implementation

We initially implement η_T pairing over $GF(2^{239})$ on ATmega128L by using Algorithm 1. The timing is 5.4 sec that is faster than the implementation of [12].

In the implementation, multiplication is implemented by the comb method with window of the width 3, squaring is implemented by table reference, and inversion is implemented by the extended Euclidean algorithm. Timings of an addition, multiplication, squaring, reduction, and inversion are 0.039 msec, 4.60 msec, 0.176 msec, 0.384 msec, and 246.2 msec, respectively. And, we found multiplications, reductions, squarings, inversion, additions occupied 75%, 6.3%, 4.7%, 4.5%, and 2.3% of whole pairing timing, respectively. Therefore, reduction of multiplication timing is most important for fast implementation of the η_T pairing.

3. Target Platform: MICAz ATmega128L

In this section, we explain ATmega128L, which is a processor for MICAz.

3.1 Architecture of ATmega128L

ATmega128L is a processor of 8-bit word, and its clock frequency is 7.38 MHz. ATmega128L consists of an arithmetic logic unit (ALU), 32 8-bit purpose registers ($R_0 \sim R_{31}$) for intermediate results, data memory of 64 Kbyte for general data, and a program ROM (flash memory) of 64 K locations.

TinyOS is an operating system for WSN, especially MICAz [15] that is often used as a platform for the research of sensor network. NesC language is extension to C language for sensor nodes. We can implement applications on ATmega128L by using NesC on TinyOS.

3.2 Operation on ATmega128L

In this section, we explain how ALU operates instruction.

The ALU operates between general purpose resisters,

$$R_d \leftarrow R_d \; opR_r$$

with one cycle, where op is an operation, and $0 \le d, r \le 31$. The ALU takes 2 cycles to store one word data in register to the memory, and takes 2 cycles to load one word date in the memory to a register. Then, computing $C = A \; op \; B$ takes 7 cycles for data in memory.

$$\begin{array}{ll} \text{load } R_d \leftarrow A & \text{(2 cycles)} \\ \text{load } R_r \leftarrow B & \text{(2 cycles)} \\ R_d \leftarrow R_d \; op \; R_r & \text{(1 cycles)} \\ \text{store } C \leftarrow R_d & \text{(2 cycles)} \end{array}$$

In other words, 1 operation takes 6 cycles for memory access (load/store operation). In general the compiler of ATmega128L generates such code.

Next, consider the multiplication of $GF(2^{239})$, namely 30 words data, $(A_{29}, \cdots, A_0) \cdot (B_{29}, \cdots, B_0)$. Note that the result is 60 words. Load/store operations to compute each partial product $A_i \cdot B_j$ takes 6 cycles. Memory access to compute a multiplication in $GF(2^{239})$ takes 900 (the number of partial products) $\times 6 = 5{,}400$ cycles, if each partial product is individually computed by the above operation.

However, if the number of registers in ATmega128L is not limited, we can write a code, with which the multiplication takes only 240 cycles for memory access, as follows:

$$\begin{array}{ll} \text{load } R_0 \leftarrow A_0 & \text{(2 cycles)} \\ \quad \vdots & \\ \text{load } R_{29} \leftarrow A_{29} & \text{(2 cycles)} \\ \text{load } R_{30} \leftarrow B_0 & \text{(2 cycles)} \\ \quad \vdots & \\ \text{load } R_{59} \leftarrow B_{29} & \text{(2 cycles)} \\ \multicolumn{2}{c}{\text{(each computation of } A_iB_j)} \\ \text{store } C_0 \leftarrow R_{60} & \text{(2 cycles)} \\ \quad \vdots & \\ \text{store } C_{59} \leftarrow R_{119} & \text{(2 cycles).} \end{array}$$

In this method, 5,160 cycles are saved per a computation of a multiplication in $GF(2^{239})$.

Of course, ATmega128L has only 32 registers, and thus we need more than 240 cycles for a multiplication in $GF(2^{239})$. The main purpose of this paper is to propose a multiplication method for $GF(2^{239})$ in which memory access by using 32 registers. Reducing the amount of memory access to compute a multiplication is effective because pairing computation needs many hundreds of multiplications.

4. The Proposed Block Comb Method

In this section, we propose a block comb method for multiplying the η_T pairing efficiently. In this method, the multiplier and multiplicand are divided into blocks and a partial product in each block is performed without memory access. In such a way, a multiplication is efficiently computed.

4.1 The Representation of Finite Field

This section explains how we represent $GF(2^{239})$ suitably for ATmega128L.

$GF(2^{239})$ is represented as $GF(2^{239})/(f(x))$, with the irreducible polynomial $f(x) = x^{239} + x^{36} + 1$. Then, a basis of $GF(2^{239})/GF(2)$ is

$$\{x^{238}, x^{237}, \cdots, x, 1\}$$

and each element A in $GF(2^{239})$ is

$$A = a_{238}x^{238} + a_{237}x^{237} + \cdots + a_1x + a_0, a_i \in GF(2). \tag{1}$$

For simplicity, we represent the right side of Eq. (1) as

$$A = (a_{238}, a_{237}, \cdots, a_1, a_0)$$

Table 1 Block multiplication of size 6.

				A^6_{24}	A^6_{18}	A^6_{12}	A^6_{6}	A^6_{0}
$\times$				B^6_{24}	B^6_{18}	B^6_{12}	B^6_{6}	B^6_{0}
				$A^6_{24}B^6_0$,	$A^6_{18}B^6_0$,	$A^6_{12}B^6_0$,	$A^6_{6}B^6_0$,	$A^6_{0}B^6_0$,
			$A^6_{24}B^6_6$,	$A^6_{18}B^6_6$,	$A^6_{12}B^6_6$,	$A^6_{6}B^6_6$,	$A^6_{0}B^6_6$,	
		$A^6_{24}B^6_{12}$,	$A^6_{18}B^6_{12}$,	$A^6_{12}B^6_{12}$,	$A^6_{6}B^6_{12}$,	$A^6_{0}B^6_{12}$,		
	$A^6_{24}B^6_{18}$,	$A^6_{18}B^6_{18}$,	$A^6_{12}B^6_{18}$,	$A^6_{6}B^6_{18}$,	$A^6_{0}B^6_{18}$,			
$A^6_{24}B^6_{24}$,	$A^6_{18}B^6_{24}$,	$A^6_{12}B^6_{24}$,	$A^6_{6}B^6_{24}$,	$A^6_{0}B^6_{24}$,				
(C^6_{54}, C^6_{48}),	(C^6_{48}, C^6_{42}),	(C^6_{42}, C^6_{36}),	(C^6_{36}, C^6_{30}),	(C^6_{30}, C^6_{24}),	(C^6_{24}, C^6_{18}),	(C^6_{18}, C^6_{12}),	(C^6_{12}, C^6_{6}),	(C^6_{6}, C^6_{0})

in this paper. A can be represented as

$$A = (\underbrace{a_{238}, \cdots, a_{232}}_{A_{29}}, \cdots, \underbrace{a_{15}, \cdots, a_8}_{A_1}, \underbrace{a_7, \cdots, a_0}_{A_0}),$$

by 30 words where each word is 8-bit. Moreover, we integrate s-wordsize as one block. Then A is represented as

$$A = (\underbrace{A_{29}, \cdots A_{(t-1)s}}_{A^s_{(t-1)s}}, \cdots, \underbrace{A_{2s-1}, \cdots, A_s}_{A^s_s}, \underbrace{A_{s-1}, \cdots, A_0}_{A^s_0})$$

where $t = \lceil 30/s \rceil$.

We call $(A_{29}, \cdots, A_0)$ the word representation, and call $(A^s_{(t-1)s}, \cdots, A^s_0)$ the block representation of size s for A.

4.2 Block Multiplication

In this section, we propose the block multiplication in $GF(2^{239})$ on ATmega128L.

In order to compute a multiplication AB for $A, B \in GF(2^{239})$, first, we compute AB as polynomial, next, we compute a reduction AB modulo $f(x)$, where $f(x) = x^{239} + x^{36} + 1$. The polynomial multiplication is an dominant time consuming operation in computing the η_T pairing. We focus on the polynomial multiplication because reduction is already fast.

Let s be the block size. $A = (A^s_{(t-1)s}, \cdots, A^s_s, A^s_0)$, $B = (B^s_{(t-1)s}, \cdots, B^s_s, B^s_0)$, where $t = \lceil 30/s \rceil$. The result C consists of $2t$ blocks.

$$C = (C^s_{(2t-1)s}, \cdots, C^s_s, C^s_0)$$

Table 1 shows the block multiplication of size 6.

Note that there are many orders of computation of partial products $A^6_{is}B^6_{js}$, and there are many choice of the block size s. Indeed, choice of the order of partial products and their block size affect the number of memory access. To sum up, there are two important issues to save the overhead of memory access time:

(1) order of the computation of partial product $A^6_{is}B^6_{js}$
(2) the block size of **s**

4.2.1 The Order of Computation of Partial Product $A^6_{is}B^6_{js}$

Here, we provide the best order for computations of partial product $A^6_{is}B^6_{js}$, that is an answer of the important issue (1).

Let us consider the following order in the case of $s = 6$:

$$\begin{aligned}
&A^6_0B^6_0 \Rightarrow\\
&A^6_6B^6_0 \rightarrow A^6_0B^6_6 \Rightarrow\\
&A^6_{12}B^6_0 \rightarrow A^6_6B^6_6 \rightarrow A^6_0B^6_{12} \Rightarrow\\
&A^6_{18}B^6_0 \rightarrow A^6_{12}B^6_6 \rightarrow A^6_6B^6_{12} \rightarrow A^6_0B^6_{18} \Rightarrow\\
&A^6_{24}B^6_0 \rightarrow A^6_{18}B^6_6 \rightarrow A^6_{12}B^6_{12} \rightarrow A^6_6B^6_{18} \rightarrow A^6_{12}B^6_{18} \Rightarrow\\
&A^6_{24}B^6_6 \rightarrow A^6_{18}B^6_{12} \rightarrow A^6_{12}B^6_{18} \rightarrow A^6_{12}B^6_{24} \Rightarrow\\
&A^6_{24}B^6_{12} \rightarrow A^6_{18}B^6_{18} \rightarrow A^6_{12}B^6_{24} \Rightarrow\\
&A^6_{24}B^6_{18} \rightarrow A^6_{18}B^6_{24} \Rightarrow\\
&A^6_{24}B^6_{24},
\end{aligned}$$

where $\rightarrow$ needs no store operation, $\Rightarrow$ needs 6-word store operation, and 12-word store operation is needed in the last product. If we choose any different order, then the number of operation "$\Rightarrow$" increases. Therefore, the above order for implementing a multiplication AB in $GF(2^{239})$ has the least number of store operations. In this case, the number of memory accesses to compute AB are 8 6-word store operations, one 12-word store operation and 16 12-word load operations. The whole memory accesses takes 720 cycles because one load/store operation takes 2 cycles on ATmega128L.

Next we consider the case of general s. We assume that there are enough registers in a processor to compute each partial product $(C^s_{(i+j+1)s}, C^s_{(i+j)s}) = A^s_{is}B^s_{js}$ without memory accesses after A^s_{is} and B^s_{js} are loaded to the registers. Note that in some cases the store operations to $(C^s_{(i+j+1)s}, C^s_{(i+j)s})$ are omitted or reduced. Suppose that we compute $A^{(i'+1)s}_{i's}B^{(j'+1)s}_{j's}$ after $A^{(i+1)s}_{is}B^{(j+1)s}_{js}$. If $i + j = i' + j'$, then the store operation is omitted. If $(i', j') = (i + 1, j)$ or $(i, j + 1)$, then only low s registers are stored to memory corresponding to $(C^s_{(i+j)s})$. From the above observation, the least number of memory accesses to compute AB require $(2t - 2)$ s-word store operations, one $2s$-word store operation and $(t - 1)^2$ $2s$-word load operations. Then, the whole memory accesses takes

$$(4st^2 - 4st + 4s) \text{ cycles}$$

to compute a multiplication AB in $GF(2^{239})$.

4.2.2 The Choice of Block Size

In this section, we provide the best block size s that is an answer of the important issue (2).

If there are unlimited registers, we know that the larger the block size s is, the less memory access cost is.

(s,t)	the cost of memory accesses (cycles)
$(1,30)$	3,484
$(2,15)$	1,688
$(3,10)$	1,092
$(5,6)$	620
$(6,5)$	504
$(10,3)$	280
$(15,2)$	180
$(30,1)$	120

However, ATmega128L has only 32 registers. Moreover, 6 registers are used for A, B, and C pointers to compute $C = AB$. Thus, only 26 registers are available.

We implement each partial product $A^6_{6i}B^6_{6j}$ by the comb method for efficiency. Note that the window method and Karatsuba method waste registers for precomputation or intermediate data; in these methods the block size is small.

In the case where $A^{(i+1)s}_{is}B^s_{js}$ is computed by the comb method, we need $4s + 1$ registers $(s, s, 2s, 1)$ for A^s_{is}, B^s_{is}, $(C^s_{(i+j+1)s}, C^s_{(i+j)s})$ and as a temporary register. We then choose s such that $4s + 1 \leq 26$, and thus the block size $s = 6$ is best.

4.3 The Proposed Block Comb Method

Algorithm 2 presents the proposed *block comb method* for computing $C = AB$ for $A, B \in GF(2^{239})$. Notations in Algorithm 2 denote the following:

load:	transfer of a data in memory to a register
store:	transfer of a data in a register to memory
move:	transfer of a data in a register to another register
$\ll 1$:	left-shift operation
$\oplus$:	bitwise exclusive-or

First, we allocate registers (R_0 to R_{31}) on ATmega128L to perform the block comb method as follows: the 12 registers $R_0, \cdots R_{11}$ are used for the result $C = AB$, register R_{12} is used for a temporary register of the comb method, the 6 registers $R_{13}, \cdots, R_{18}$ are used for the multiplier A, and the 6 registers $R_{19}, \cdots R_{24}$ are used for the multiplicand B.

Next, we explain each Step in Algorithm 2. Steps 1 to 7 initialize the 12 registers corresponding to C. Steps 8 to 30 correspond to the comb method to compute

$$\underbrace{(A_{6j+5}, \cdots, A_{6j})}_{A^6_{6j}} \cdot \underbrace{(B_{6k+5}, \cdots, B_{6k})}_{B^6_{6k}}. \tag{2}$$

In Step 12 $(A_{6j+5}, \cdots, A_{6j})$ in memory are loaded to $R_{13}, \cdots, R_{19}$, and $(B_{6k+5}, \cdots, B_{6k})$ in memory are loaded to $R_{20}, \cdots, R_{25}$.

Algorithm 2 Block comb method for computing $C = AB$ with block size 6 (A,B in $GF(2^{239})$)

```
Input: Binary polynomials $A = (A_{29}, \cdots, A_0)$, $B = (B_{29}, \cdots, B_0)$ (in
    memory), where A,B in $GF(2^{239})$
Output: $(C_{59}, \cdots, C_0) = AB$ (to memory)
 1: for $i = 0$ to 5 do
 2:     $R_i \leftarrow 0$
 3: end for
 4: for $i = 0$ to 8 do
 5:     for $j = 6$ to 11 do
 6:         $R_j \leftarrow 0$
 7:     end for
 8:     for $j = 0$ to 4 do
 9:         $k = i - j$
10:         if $0 \leq k$ and $k \leq 4$ then
11:             for $l = 0$ to 5 do
12:                 load $R_{13+l} \leftarrow A_{6j+l}$
13:                 load $R_{19+l} \leftarrow B_{6k+l}$
14:             end for
15:             $R_{12} \leftarrow 0$
16:             for $l = 7$ downto 0 do
17:                 $(R_{12}, \cdots, R_0) \leftarrow (R_{12}, \cdots, R_0) \ll 1$
18:                 for $m = 6$ downto 1 do
19:                     for $n = 0$ to 5 do
20:                         if (the $l$-th bit of $R_{18+m}$) = 1 then
21:                             $R_{m+n} \leftarrow R_{m+n} \oplus R_{13+n}$
22:                         end if
23:                     end for
24:                 end for
25:             end for
26:             for $l = 0$ to 11 do
27:                 $R_n \leftarrow R_{n+1}$
28:             end for
29:         end if
30:     end for
31:     for $k = 0$ to 5 do
32:         store $C_{6i+k} \leftarrow R_k$
33:         $R_k \leftarrow R_{k+6}$
34:     end for
35: end for
36: for $i = 0$ to 6 do
37:     store $C_{54+i} \leftarrow R_{6+i}$
38: end for
```

Steps 16 to 25 are the body of the comb method. Note that the result of Eq. (2) is registered in $(R_{12}, \cdots, R_1)$ (not $(R_{11}, \cdots, R_0)$) due to shift operations at Step 17. Then, we need Steps 26 to 28 to make R_{11} (the most significant word), and R_0 (the least significant bit)[†].

In Steps 32, the lowest 6 words $(R_5, \cdots, R_0)$ of the result of the partial product are stored. Recall that the highest 6 words $(R_{11}, \cdots, R_6)$ can be reused in the next iteration, as explained in Sect. 4.2. Last, the highest 6 words are also stored (Steps 36 to 38).

In the following we estimate the efficiency of the proposed comb method. A multiplication of the block comb method with block size 6 takes 504 cycles for memory access. Recall that a straight-forward multiplication, which calls each partial product A_iB_j individually, takes 5,400 cy-

[†]We may omit Steps 22 to 24 in Algorithm 2 if we allow the movement of the most/least significant word of $(R_{12}, \cdots, R_0)$ and modify Algorithm 2.

Table 2 Instructions of ATmega128L used in this paper.

Description	Syntax	Operation	Operand	Program counter	#Clock
Load to Y	ld R_d, Y	$R_d \leftarrow (Y)$	$0 \leq d \leq 31$	$PC \leftarrow PC + 1$	2
	ld $R_d, Y+$	$R_d \leftarrow (Y), Y \leftarrow Y + 1$	$0 \leq d \leq 31$	$PC \leftarrow PC + 1$	2
	ldd $R_d, Y + q$	$R_d \leftarrow (Y + q)$	$0 \leq d \leq 31, 0 \leq q \leq 63$	$PC \leftarrow PC + 1$	2
Store to X	st X, R_r	$(X) \leftarrow R_r$	$0 \leq r \leq 31$	$PC \leftarrow PC + 1$	2
	st $X+, R_r$	$(X) \leftarrow R_r, X \leftarrow X + 1$	$0 \leq r \leq 31$	$PC \leftarrow PC + 1$	2
Logical Shift Left	lsl R_d	$R_d(n + 1) \leftarrow Rd(n)$ $(1 \leq n \leq 6)$, $R_d(0) \leftarrow 0, C \leftarrow Rd(7)$	$0 \leq r \leq 31$	$PC \leftarrow PC + 1$	1
Rotate Left Through Carry	rol R_d	$R_d(0) \leftarrow C, R_d(n + 1) \leftarrow Rd(n)$ $(1 \leq n \leq 6)$, $C \leftarrow R_d(7)$	$0 \leq r \leq 31$	$PC \leftarrow PC + 1$	1
Skip if Bit in Register Set	sbrs R_r, b	if $(R_r(b) = 1) PC \leftarrow PC + 2$ else $PC \leftarrow PC + 1$	$0 \leq r \leq 31, 0 \leq b \leq 7$	$PC \leftarrow PC + 2$ (T) $PC \leftarrow PC + 1$ (F)	2 (T) 1 (F)
Relative Jump	rjmp k	$PC \leftarrow k + 1$		$PC \leftarrow k + 1$ (T)	2
Copy Register	mov R_d, R_r	$R_d \leftarrow R_r$	$0 \leq d, r \leq 31$	$PC \leftarrow PC + 1$	1
Copy Register Word	movw R_d, R_r	$R_{d+1} : R_d \leftarrow R_{r+1} : R_r$	$d, r \in \{0, 2, \cdots, 30\}$	$PC \leftarrow PC + 1$	1
AND immediate	andi R_d, K	$R_d \leftarrow R_d$ and K	$16 \leq d \leq 31, 0 \leq K \leq 255$	$PC \leftarrow PC + 1$	1
Exclusive OR	eor R_d, R_r	$R_d \leftarrow R_d \oplus R_r$	$0 \leq d, r \leq 31$	$PC \leftarrow PC + 1$	1
Swap Nibbles	swap R_d	$R(7 : 4) \leftarrow R_d(3 : 0)$ $R(3 : 0) \leftarrow R_d(7 : 4)$	$0 \leq d, r \leq 31$	$PC \leftarrow PC + 1$	1
Clear Register	clr R_d	$R_d \leftarrow R_d \oplus R_d$	$0 \leq d \leq 31$	$PC \leftarrow PC + 1$	1

R_d:	Destination register
R_r:	Source register
k:	Constant address
K:	Constant data
b:	Bit in the register
X, Y:	Indirect Address Register ($X = R_{27} : R_{26}$ and $Y = R_{29} : R_{28}$)
q:	Displacement for direct addressing (6-bit)
C:	Carry flag
PC:	Program Counter

cles for memory access, as described in Sect. 3.2. Therefore, the proposed block comb method for a multiplication AB in $GF(2^{239})$ is about 10 times faster than the multiplication with the slowest memory access. In the real implementation, a compiler does not usually gives us a code of multiplication with the slowest memory access, and thus the improvement by the proposed scheme becomes smaller — it strongly depends on the underlying compiler. We will demonstrate the effectiveness of the block comb method applied to ATmega128L in Sect. 5.1.

5. Our Implementation

In this section, we explain about the details of our implementation of the η_T pairing over $GF(2^{239})$ on ATmega128L. Refer Table 2 for each instruction used in this section of ATmega128L in details.

5.1 Implementation of Proposed Block Comb Method

We implemented the proposed block comb method (Algorithm 2) by assembly. In the following we explain main three improvements: memory access (Step 8, Step 31, Step 36), if-statement (Step 20), and left-shift (Step 17), where their timing is more than 80% of the whole multiplication of $GF(2^{239})$.

The proposed block comb method with the three improvements below can compute a multiplication in $GF(2^{239})$ in 1.29 msec. The comb method of width 3 in Sect. 2.3 requires 4.60 msec, and thus our implementation of the proposed block comb method is about 3.6 times faster.

5.1.1 Memory Access

Note that 16-bit registers are required for indicating memory address in ATmega128L because the size of the memory is 2^{16} bytes and hence 16-bit indirect address registers are needed for memory addressing. Six registers can be used as 3 16-bit registers, X-register (R_{26} and R_{27}), Y-register (R_{28} and R_{29}), and Z-register (R_{30} and R_{31}). We use registers X, Y, and Z for pointers C, A, and B, respectively. We can write load operations by "ldd" instructions. For example, we can implement the load instruction at Step 8 as follows.

ldd $R_{13+l}, X + (6j + l)$

Note that $X + (6j + l)$ denotes the address of A_{6j+1}. The ldd instruction takes 2 cycles.

We can write store operations by "st" instructions. Note that store operations are sequentially performed to $C_0, C_1, \cdots, C_{59}$ at Steps 31 and 36. We can implement a store operation as follows[†].

st $X+, R_{13+k}$

[†]An std instruction corresponding to ldd is prepared in ATmega128L. However, the std instruction does not support the X register. Therefore, we must use the st instruction for store operations in Algorithm 2.

Table 3 Comparison of timing of pairing on ATmega128L.

	TinyTate [17]	TinyPBC [18]	Ishiguro *et al.* [12]	**Ours**
Language	NesC	NesC	NesC	Assem, NesC
Pairing	Tate ($GF(q)$) $q^2 \approx 512$ bits	η_T ($GF(2^{271})$)	η_T ($GF(3^{97})$)	η_T ($GF(2^{239})$)
ROM (byte)	18,384	47,948	17,284	35,012
memory (byte)	1,831	368	628	568
Timing of pairing (sec)	30.21	5.45	5.79	1.93

5.1.2 If-Statement

We can implement the if-statement at Step 20 by using "sbrs" and "rjmp" instructions as follows.

```
sbrs $R_{19+j}, l$
rjmp Label
eor $R_0, R_{13}$
eor $R_1, R_{14}$
    ⋮
eor $R_5, R_{18}$
Label:
    ⋮
```

In the case of an lth bit of $R_{19+j} = 1$, rjmp is not executed and eor (exclusive or) operations are executed. The sbrs instruction takes two cycles. If the lth bit of $R_{19+j} = 0$ and rjmp is executed, the eor operations are not executed. The sbrs instruction takes one cycle, and the rjmp instruction takes two cycles. The if-statement takes 2.5 cycles on average because the probability of $R_{19+j} = 1$ is 0.5.

5.1.3 Block Left-Shift

We can implement the block left-shift at Step 17 by using "lsl" and "rol" instructions as follows.

```
lsl $R_1$
rol $R_2$
rol $R_3$
    ⋮
rol $R_{12}$
```

Instructions lsl and rol each take one cycle. Thus, the block left-shift takes only 12 cycles in the implementation.

5.2 Other Improvements

The shift operation is often used in the squaring, reduction, and inversion. We implemented the shift operation optimized for ATmega128L.

Let $A_0 = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$ be an 8-bit register. "$(a_3, a_2, a_1, a_0, 0, 0, 0, 0) = A_0 \ll 4$" (left 4-bit shift) is complied by NesC as follows.

```
ldd $R_{20}, Z$ ;; $R_{20} = A_0$
mov $R_{22}, R_{20}$ ;; $R_{22} = A_0$
clr $R_{23}$, ;; $R_{23} = 0$
movw $R_{24}, R_{22}$ ;; $R_{24} = A_0, R_{25} = 0$
swap $R_{24}$ ;; $R_{24} = (a_3, a_2, a_1, a_0, a_7, a_6, a_5, a_4)$
swap $R_{25}$ ;; $R_{25} = 0$
andi $R_{25}, 0xf0$ ;; $R_{25} = 0 \& 0xf0 = 0$
eor $R_{25}, R_{24}$ ;; $R_{25} = R_{25} \wedge R_{24} = R_{24}$
andi $R_{24}, 0xf0$ ;; $R_{24} = A_0 \ll 4$
```

We can optimize it as follows.

```
ldd $R_{20}, Z + 15$ ;; $R_{20} = A_0$
mov $R_{22}, R_{20}$ ;; $R_{22} = A_0$
mov $R_{24}, R_{22}$ ;; $R_{24} = A_0$
swap $R_{24}$ ;; $R_{24} = (a_3, a_2, a_1, a_0, a_7, a_6, a_5, a_4)$
andi $R_{24}, 0xf0$ ;; $R_{24} = A_0 \ll 4$
```

From this we reduce to 6 clocks from 10 clocks. Similarly we can perform the same improvements for left i-bit sift or right i-bit sifts for $i = 1$ to 7.

As a result, the timings of squaring, reduction, and inversion are improved to 0.129 msec, 0.0304 msec, and 166.8 msec from 0.176 msec, 0.0384 msec, and 246.2 msec of our initial implementation in Sect. 2.3, respectively.

5.3 Comparison with Other Works

In Table 3, we show a comparison of implementation of pairing on ATmega128L with previous works. Ours is 3 times faster than the implementation of η_T pairing over $GF(3^{97})$ which has the same security level (see Sect. 2.1), but we used twice larger ROM due to assembly code. TinyPBC has a larger parameter size, and the weighted speed of their implementation over $GF(2^{239})$ is $5.45(239/271)^2 = 4.24$ sec., which is still twice slower than ours. The ROM size of TinyPBC is larger than that of ours because they used a table look-up for Karatuba multiplication in $GF(2^{271})$. TinyTate uses finite field $GF(p)$ with a large prime characteristic, and it is currently much slower than other implementation of pairing on ATmega128L.

Remark 1: The timing of implementation only by NesC (without assembly) can also be improved by the proposed block comb method. The improvement by assembly merely aims at achieving the top timing of pairing implementation at ATmega128L. For example, the block comb method enhances the speed of Ishiguro *et al.* [12] or TinyECCK [23], because it uses the comb method.

6. Conclusion

In this paper we presented an efficient implementation of pairing over a sensor node. We implemented the η_T pairing over $GF(2^{239})$ using MICAz platform with ATmega128L. In order to accelerate the speed of the pairing, we proposed *the block comb method* that is particularly optimized for ATmega128L. Combining with other optimizations for squaring, reduction, and inversion, the timing of the η_T pairing becomes 1.93 sec. This is currently the fastest timing comparing with the previously known implementations of pairing over sensor nodes. Ubiquitous sensor networks now use the pairing-based cryptography in a reasonable time.

Acknowledgements

The work reported in this paper was supported by the IT R&D program of MIC/IITA. [2005-S088-04, Development of Security technology for Secure RFID/USN Service].

References

[1] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," SIAM J. Comput., vol.32, no.3, pp.586–614, 2003.

[2] P. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott, "Efficient pairing computation on supersingular abelian varieties," Des. Codes Cryptogr., vol.42, no.3, pp.239–271, 2007.

[3] P. Barreto, H. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing-based cryptosystems," CRYPTO 2002, LNCS 2442, pp.354–368, 2002.

[4] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," CRYPTO 2005, LNCS 3621, pp.258–275, 2005.

[5] J.-L. Beuchat, M. Shirase, T. Takagi, and E. Okamoto, "An algorithm for the η_T pairing calculation in characteristic three and its hardware implementation," 18th IEEE International Symposium on Computer Arithmetic, ARITH-18, pp.97–104, 2007.

[6] J.-L. Beuchat, M. Shirase, T. Takagi, and E. Okamoto, "A refined algorithm for the η_T pairing calculation in characteristic three," Cryptology ePrint Archive, Report 2007/311, 2007.

[7] I. Duursma and H. Lee, "Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$," ASIACRYPT 2003, LNCS 2894, pp.111–123, 2003.

[8] E. Gorla, C. Puttmann, and J. Shokrollahi, "Explicit formulas for efficient multiplication in $\mathbb{F}_{3^{6m}}$," SAC 2007, LNCS 4876, pp.173–183, 2007.

[9] R. Granger, D. Page, and M. Stam, "On small characteristic algebraic tori in pairing-cased cryptography," LMS Journal of Computation and Mathematics, vol.9, pp.64–85, 2006.

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

[11] IEEE P1363.3, Available at http://grouper.ieee.org/groups/1363/IBC/index.html

[12] T. Ishiguro, M. Shirase, and T. Takagi, "Efficient implementation of pairing on sensor nodes," Applications of pairing-based cryptography, NIST, pp.96–106, 2008.

[13] T. Kerins, W. Marnane, E. Popovici, and P. Barreto, "Efficient hardware for the Tate pairing calculation in characteristic three," CHES 2005, LNCS 3659, pp.412–426, 2005.

[14] A. Liu, P. Kampanakis, and P. Ning, "TinyECC: Elliptic curve cryptography for sensor networks (version 0.3)," http://discovery.csc.csu.edu/software/TinyECC/, 2007.

[15] MICAz Hardware Description, Available at http://www.xbow.jp/

[16] T. Nakajima, T. Izu, and T. Takagi, "Reduction optimal trinomials for efficient software implementation of the η_T pairing," IWSEC 2007, LNCS 4752, pp.44–57, 2007.

[17] L. Oliveira, D. Aranha, E. Morais, F. Daguano, J. López, and R, Dahab, "TinyTate: Identity-based encryption for sensor networks," Cryptology ePrint Archive, Report 2007/020, 2007.

[18] L. Oliveira, M. Scott, J. López, and R. Dahab, "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," Cryptology ePrint Archive, Report 2007/482, 2007.

[19] RFC 5091, Available at http://www3.tools.ietf.org/html/rfc5091

[20] A. Ramachandran, Z. Zhou, and D. Huang, Computing cryptographic algorithms in portable and embedded devices, IEEE Portable, 2007.

[21] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," Commun. ACM, vol.47, no.6, pp.53–57, 2004.

[22] R. Ronan, C. Ó hÉigeartaigh, C. Murphy, T. Kerins, and P. Barreto, "Hardware implementation of the η_T pairing in characteristic 3," Cryptology ePrint Archive, Report 2006/371, 2006.

[23] S. Seo, D-G. Han, H. Kim, and S. Hong, "TinyECCK: Efficient elliptic curve cryptography implementation over GF(2^m) on 8-bit micaz mote," IEICE Trans. Inf. & Syst., vol.E91-D, no.5, pp.1338–1347, May 2008.

[24] M. Shirase, T. Takagi, and E. Okamoto, "Some efficient algorithms for the final exponentiation of η_T pairing," ISPEC 2007, LNCS 4464, pp.254–268, 2007.

[25] C. Shu, S. Kwon, and K. Gaj, "FPGA accelerated Tate pairing based cryptosystems over binary fields," Cryptology ePrint Archive, Report 2006/179, 2006.

[26] P. Szczechowiak, L. Oliveira, M. Scott, M. Collier, and R. Dahab, "NanoECC: Testing the limits of elliptic curve cryptography in Sensor Networks," EWSN 2008, LNCS 4913, pp.305–320, 2008.

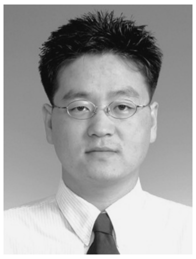
[27] R. Watro, D. Kong, S. Cuti, C. Lynn, and P. Knuus, "TinyPK: Securing sensor networks with public key technology," SASN 2004, pp.59–64, 2004.

Masaaki Shirase received the B.Sc. in mathematics from Ibaraki University in 1994, and M.I.S. and Dr.I.S degrees from JAIST (Japan Advanced Institute of Science and Technology) in 2003 and 2006, respectively. He is currently a Postdoctoral in the School of System Science Information at Future University-Hakodate. His research interests are algorithm and implementation of cryptography.

Yukinori Miyazaki received the Bachelor of Media Architecture from School of Systems Information Science at Future University-Hakodate in 2009. He is currently a master student in Graduate School of Systems Information Science at Future University-Hakodate. He is a student member of IPSJ.

Tsuyoshi Takagi received the B.Sc. and M.Sc. degrees in mathematics from Nagoya University in 1993 and 1995, respectively. He had engaged in the research on network security at NTT Laboratories from 1995 to 2001. He received the Dr.rer.nat degree from Technische Universität Darmstadt in 2001. He was an Assistant Professor in the Department of Computer Science at Technische Universität Darmstadt until 2005. He is currently a Professor in the School of Systems Infomation Science at Future University-Hakodate. His current research interests are information security and cryptography. Dr. Takagi is a member of International Association for Cryptologic Research (IACR).

Dong-Guk Han received his B.S. degree in mathematics from Korea University in 1999, and his M.S. degrees in mathematics from Korea University in 2002, respectively. He received Ph.D. of engineering in Information Security from Korea University in 2005. He was a Post.Doc. in Future University-Hakodate, Japan. After finishing the doctor course, he had been an exchange student in Dep. of Computer Science and Communication Engineering in Kyushu University in Japan from Apr. 2004 to Mar. 2005. He was a senior researcher in Electronics and Telecommunications Research Institute (ETRI), Daejeon, Rep. of Korea. He is currently working as an assistant professor with the Department of Mathematics of Kookmin University, Seoul, Rep. of Korea. He is a member of KIISC, IEEK, and IACR.



Dooho Choi received his B.S. degree in mathematics from Sungkyunkwan University, Seoul, Korea in 1994, and the M.S. and Ph.D. degrees in mathematics from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea in 1996, 2002, respectively. He is currently a senior researcher in Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea from Jan. 2002. His research interests include security technologies of RFID and wireless sensor network. He is an editor of the ITU-T X.1171 (X.nidsec-1).