Optimized Multi-Precision Multiplication for Public-Key Cryptography on Embedded Microprocessors

Hwajeong Seo and Howon Kim

***Abstract*—In this paper, we revisit the previous multi-precision multiplication techniques including "operand-scanning", "hybrid-scanning", "operand-caching", "consecutive operand-caching" and "product-scanning." Particularly, the former four methods execute an intermediate result computation which is process for updating the results with a newly computed result by computing a number of addition operations. This operations is expensive, so efficient implementation is required to boost the performance. For this reason, we propose a novel method, i.e., "Carry-Once", which reduces the number of intermediate result computation by size of result accumulation. The main idea is gathering carry values and updating the values at once. This method improves all multi-precision multiplication techniques having intermediate result computation and show performance enhancement in terms of speed by up to 2.5%, compared with best known results.**

***Index Terms*—Multi-precision multiplication, public-key cryptography, carry-once method, embedded microprocessors.**

I. Introduction

Public key cryptography methods such as RSA [1], ECC [2], and pairing [3] involve computation-intensive arithmetic operations; in particular, multiplication accounts for most of the execution time of microprocessors. Several technologies have been proposed to reduce the execution time and computation cost by decreasing the number of clock cycles. A row-wise method called "operand scanning" is used for short looped programs. This method loads all operands in a row.

The alternative Comba method is a common schoolbook method that is also known as the "product scanning method." This method computes all partial products in a column [4].

The "hybrid scanning" combines the useful features of operand scanning and product scanning. By adjusting row and column width, the number of operand accesses and result updates are reduced. This method has an advantage over a microprocessor equipped with many general purpose registers [5]. At CHES'11, the "operand caching" was proposed [6]. This method significantly reduces the number of load operations, which are regarded as expensive operations, via caching of operands.

At WISA'12, the "consecutive operand caching" was proposed [7] which enhances previous "operand caching" with fully caching required operands, so it does not reload operands throughout the computations.

Manuscript received November 4, 2012; revised January 18, 2013.

The authors are with the Department of Computer Engineering, Pusan National University, South Korea (e-mail: hwajeong@pusan.ac.kr, howonkim@pusan.ac.kr).

Previous multi-precision multiplication methods were paying more attention on reducing the expensive memory access. However, multiplication also spends a number of clock cycles for accumulation of intermediate result value.

In this paper, we propose a novel multiplication method that highly optimizes the number of addition instructions required for intermediate result update. The remainder of this paper is organized as follows. In Section II, we describe previous multi-precision multiplication techniques and in Section III, we introduce a novel intermediate result update method, "Carry-Once." In Section IV, we describe the performance evaluation in terms of number of addition instructions. Finally, Section V concludes the paper.

II. Multi-Precision Multiplication Techniques

In this section, we introduce various multi-precision multiplication techniques, including "operand-scanning", "hybrid-scanning", "operand-caching", "consecutive operand-caching" and "product-scanning."

Each method has unique features for reducing the number of load and store instructions, and they have similar intermediate result update process which reloads intermediate results and accumulates them with newly computed result in the column except "product-scanning."

This operation is expensive, so efficient implementation is important for high performance. To enhance the performance, we present an advanced intermediate result update method, "Carry-Once" which reduces the number of required addition instructions. This method is described with multiplication method.

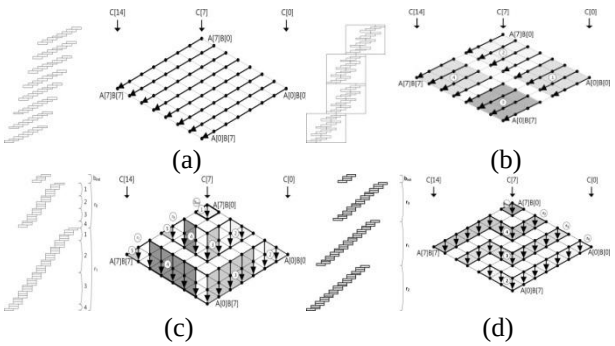

Fig. 1. Multi-precision multiplication techniques. (a)operand-scanning. (b)hybrid-scanning. (c)operand-caching. (d)consecutive operand-caching [6], [9].

To describe the multi-precision multiplication method, we use the following notations. Let A and B be two m-bit operands that are multiple-word arrays. Each operand is written as follows: $A=(A[n-1],...,A[2],A[1],A[0])$ and $B=(B[n-1],...,B[2], B[1], B[0])$. The division of operand-size (m)

DOI: 10.7763/IJCCE.2013.V2.183

by word-size (w) represents the number of elements (n) in the operand array. The result of multiplication is twice as large as operand $C=(C[2n\text{-}1]...\ C[2], C[1], C[0])$.

For clarity, we describe the method using a multiplication structure and rhombus form, as shown in Fig. 1. Each point represents a multiplication $A[i] \times B[j]$. The rightmost corner of the rhombus represents the lowest indices ($i, j=0$), whereas the leftmost represents corner the highest indices ($i, j=n\text{-}1$). The lowermost side represents result indices $C[k]$, which ranges from the rightmost corner ($k=0$) to the leftmost corner ($k=2n\text{-}1$).

A. Operand-Scanning Method

This method consists of two parts, i.e., inner and outer loops. In the inner loop, operand $A[i]$ holds a value and computes the partial products with all multiple values of the multiplicand $B[j]$ ($j=0...n\text{-}1$). In the outer loop, the index of operand $A[i]$ increases by a word-size and then the inner loop is executed. Fig. 1 (a) shows the "operand-scanning" method. The arrows indicate the order of computations which are executed from the rightmost corner to the leftmost corner.

B. Hybrid-Scanning Method

This method combines the useful features of "operand scanning" and "product-scanning" [5]. Multiplication is performed on a block scale using "product scanning." The number of rows within the block is defined as d and inner block partial products follow the "operand-scanning" rule. Therefore, this method reduces the number of *load* instructions by sharing the operands within the block. As the number of available registers increases, the size of row (d) increases and, the number of memory accesses is reduced by the number of shared operands. Fig. 1 (b) shows the "hybrid" method in the case of ($d=4$).

C. Operand-Caching Method

This method follows the "product-scanning" method, but it divides the calculation into several row sections [6]. By reordering the sequences of inner and outer row sections, previously loaded operands in working registers are reused for the next partial products. Although a few *store* instructions are added, but the number of required *load* instruction is reduced. The number of row section is given by $r = \lfloor n/e \rfloor$, and e denotes the number of words used to cache digit in the operand. Fig. 1 (c) shows the "operand caching" method in the case of ($e=3$). Given $n=8$, the number of row section is $r = \lfloor 8/3 \rfloor = 2$.

D. Consecutive-Operand-Caching Method

This method is based on "operand-caching", so it can perform multiplication with reduced number of memory accesses for operand *load* instructions by caching operands. However, previous method has to reload operands whenever a row is changed, which generates unnecessary overheads. To overcome these shortcomings, this method divided the rows and re-scheduled the multiplication sequences to find a contact point among rows that share the common operands for partial products. This method can cache the operands, when a row is changed. A detailed example is shown in Fig. 1 (d). The size of the caching operand e and the number of elements n are set to 2 and 8, respectively. The value e is determined by the number of working registers in the platform. The number of rows is $r=4$, following the notation $r = \lfloor n/e \rfloor$. Given the number of working registers w, the value is $w=3+2e$. Three working registers are used for accumulating the intermediate results obtained from the partial products.

E. Product-Scanning Method

The "product-scanning" method is not efficiently computed with "Carry-Once" method, so we don't describe the method in main contents.

III. Proposed Method

In the section, we present a novel method, "Carry-Once" method for multi-precision multiplication having accumulation of intermediate results. Firstly, we provide main idea of "Carry-Once" method and then apply the method to previous multiplication techniques including "Operand-Scanning", "Hybrid-Scanning", "Operand-Caching" and "Consecutive Operand-Caching."

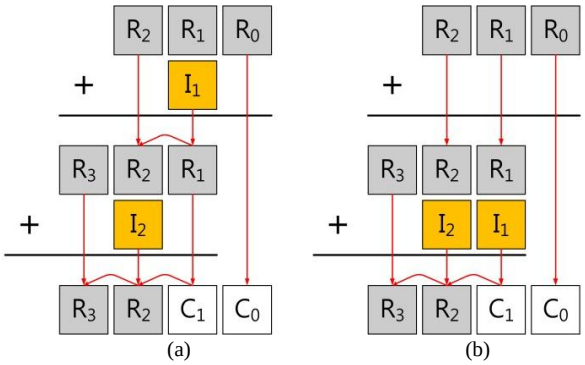

Fig. 2. Carry-once method, R, I, C represent computed result, reloaded intermediate result and result of memory, respectively.

A. Carry-Once Method

The idea of "Carry-Once" is gathering the intermediate results and updating the values at once. Fig. 2 describes "Carry-Once" in detail. The case (a) is normal implementation of carry operation, in which computed three result values (R_0, R_1, R_2) are updated with intermediate result I_1 and least result (R_0) is stored into memory C_0 directly. Therefore, intermediate result I_1 is added to R_1 and carry of previous addition is added to R_2. In the next step, same process is conducted after one word size shifted to the left. Finally, four addition operations are required for case (a). Case (b) describes "Carry Once", in which updating results (R_0, R_1, R_2) with intermediate results are not computed at the first process. In next step, intermediate results (I_1, I_2) are updated with result values (R_0, R_1, R_2) at once. Therefore, case (b) updates intermediate results with three addition operations and this uses one less addition operation than normal technique. The advantage of method is computed with following equations. Let i, s, and b be size of computed result, reloaded intermediate result, and block ($\lfloor i/s \rfloor$). If $\lfloor i/s \rfloor = i/s$, the advantage is $(s\text{-}1) \times b = sb\text{-}b$ in a row. If $\lfloor i/s \rfloor \neq i/s$, the advantage is $(s\text{-}1) \times b + (i\text{-}bs\text{-}1) = i\text{-}b\text{-}1$.

Following sections present detailed flow of multiplication.

B. Multi-Precision Multiplication with Carry-Once Method

In the section, we introduce practical usages of "Carry-Once" for previous multi-precision multiplication methods. The used parameters, *n* and *c* are set to 8 and 2 respectively.

1) Operand-Scanning Method: "Operand-scanning" is described in Fig. 3 (a) and red dots represent intermediate result computations. This method contains numerous computations. From second row to last row, all result values are influenced by previous computed results. By reducing the number of computations with "Carry-Once" we can expect high performance. The number of intermediate result is $n(n-1)=n^2$. And required number of addition is double number of intermediate result, $2(n^2-n)$.
2) Hybrid-Scanning Method: "Hybrid-scanning" is described in Fig. 3 (b). In the example, *d* is defined as four and process is separated into four sub-multiplications. Block No. 1 does not have duplicated part with previous computation, so we don't need to compute intermediate results. From block No. 2 to 4, update processes are conducted. Particularly, Block No. 2 and 4 compute intermediate result with half size of row, because remaining half parts don't have previously computed results. Let number of sub-block be $q=n/d$ (*n* is divisible by *d*). The number of intermediate computation is determined by size of row d and its cost is $(q\text{-}1)(2dq+q+1)$. Required number of addition is $2(q\text{-}1)(2dq+q+1)$.
3) Operand-Caching Method: "Operand-caching" is shown in Fig. 3 (c). In the example *e* is a size of caching operand and the number of row is $\lfloor n/e \rfloor$. The computation is executed from b_{init} to last row r_1. In the row computations, intermediate result computation is executed to update previous intermediate results. The cost of operation is determined by parameter *n* and *e*, because the parameters determine the size of rows and number of rows. If condition is $\lfloor n/e \rfloor = n/e$, the number of operation is $\sum_{k=1}^{r} 2e(k-1) = r^2e - re$ and addition is $2(r^2e-re)$ Otherwise, the number of operation is $2(n-er)r + \sum_{k=1}^{r} 2e(k-1) = 2nr - r^2e - re$ and addition is $2(2nr-r^2e-re)$
4) Consecutive Operand-Caching Method: "Consecutive operand-caching" is shown in Fig. 3 (d). In the example *e* is a size of caching operand. The process has similar structure of "operand-caching." Therefore, similarly the cost of the operation is determined by parameter *n* and *e*. If condition is $\lfloor n/e \rfloor = n/e$, the number of operation is $nr+re\text{-}2n$; addition is $2nr+2re\text{-}4n$. Otherwise, the number of operation is $nr+re\text{-}n+e$, addition is $2(nr+re\text{-}n+e)$.

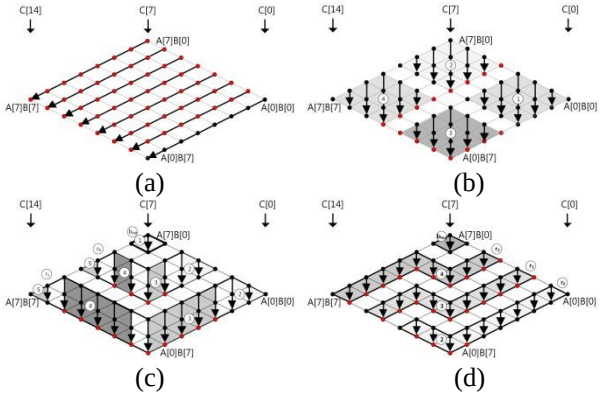

Fig. 3. Intermediate result computation of multi-precision multiplications. (a)operand-scanning. (b)hybrid-scanning. (c)operand-caching. (d)consecutive operand-caching [6], [9].

IV. Result

In this section, we show the complexity of proposed intermediate result computation and estimated result over 8-bit microprocessor depending on different parameters including size of element and caching registers.

A. Complexity of Proposed Method

Fig. 4 shows performance enhancement of proposed method. The graph represents reduced complexity compared with basic implementation which doesn't use optimization techniques. By increasing size of "carry once (Y-axis)," the number of intermediate operation is reduced in case of operand scanning and hybrid scanning. Otherwise, "(Consecutive) operand caching," we set size of "carry-once" to 2 and adjust a size of caching operands (Y-axis). In the both graphs, X-axis represents size of multiplication.

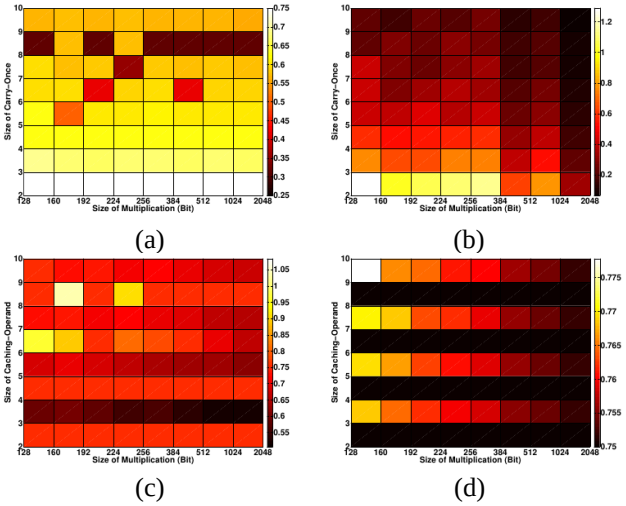

Fig. 4. Performance enhancement of proposed method in carry addition method(Proposed/Basic). (a)operand-scanning. (b)hybrid-scanning. (c)operand-caching. (d)consecutive operand-caching.

1) Operand-Caching: "Operand scanning" is suitable structure for adopting proposed method. Therefore, performance enhancement is significantly higher than any other methods. The following equation is cost for intermediate update process. $(n-1)(\lfloor \frac{n}{c} \rfloor + n)$ The cost is highly relied on size of Carry-Once, *c* so the cost is reduced by increasing the size *c*.

2) Hybrid-Scanning: "Hybrid scanning" is determined by width of column (d) which sets size of caching operands. As the width of column increases, longer size of Carry-Once is possible, so performance increases together with d. The cost of intermediate reload is determined by following conditions. If condition is $d+1>c$, the number of operation is $2(w-1)\left(\left\lfloor \frac{d+1}{c} \right\rfloor + d + 2\right) + (w-1)^2\left(\left\lfloor \frac{2d+1}{c} \right\rfloor + 2d + 2\right)$ If condition is $2d+1 > c \geq d+1$, the number of operation is $2(w-1)(d+1) + (w-1)^2\left(\left\lfloor \frac{2d+1}{c} \right\rfloor + 2d + 2\right)$. If condition is $c \geq 2d+1$ the number of operation is $2(w-1)(d+1) + (w-1)^2(2d+1)$

3) Operand-Scanning: In "operand-caching," the size of "carry-once" is set to value two, because "operand-caching" uses a number of registers for retaining operands, so size is restricted to two. The performance is higher when n is divisible with e and e is even, because it perfectly suits to size of "carry-once." The cost is determined by characteristic of divisibility and even. If condition is $\lfloor n/e \rfloor = n/e$ and e is odd, the number of operation is $2(\frac{3}{2}\sum_{k=1}^{r-1} 2(e-1)k + 2(r-1)) = \frac{3}{2}(e-1)r(r-1) + 4(r-1)$. If condition is $\lfloor n/e \rfloor = n/e$ and e is even, the number of operation is $2\left(\frac{3}{2}\sum_{k=1}^{r-1} ek\right) = \frac{3}{2}(e-1)r(r-1)$. If condition is $\lfloor n/e \rfloor \neq n/e$ and e is odd, the number of operation is $2((n-re)r + \sum_{k=1}^{r-1}(e-1)k)\frac{3}{2} + 2(r-1)) =$

$$2((2(n-re)r + r(r-1)(e-1))\frac{3}{4} + 2(r-1)).$$

4) If condition is $\lfloor n/e \rfloor \neq n/e$ and e is even, the number of operation is

$$3((n-re)r + \sum_{k=1}^{r-1} ek = 3r(n-re) + \frac{3}{2}er(r-1)$$

5) Consecutive Operand-Caching: In "consecutive operand caching," cost of intermediate reload is determined with same conditions of "operand caching." This method shows much more obvious contrasts between characteristic of e, whether it is odd or even number, because even number divides number of element completely. If condition is $\lfloor n/e \rfloor = n/e$ and e is odd, the number of operation is $2\left(\frac{3}{2}(e-1) + \frac{3}{4}(n+e-1)(r-2) + (r-2)\right)$. If condition is $\lfloor n/e \rfloor = n/e$ and e is even, the number of operation is $2\left(\frac{3}{2}e + 2 + \frac{3}{4}(n+e)(r-2)\right)$. If condition is $\lfloor n/e \rfloor \neq n/e$ and e is odd, the number of operation is $2\left(\frac{3}{2}(e-1) + 2 + \frac{3}{4}(n+e-1)(r-1) + (r-1)\right)$. If condition is $\lfloor n/e \rfloor \neq n/e$ and e is even, the number of operation is $2\left(\frac{3}{2}e + \frac{3}{4}(n+e)(r-1)\right)$.

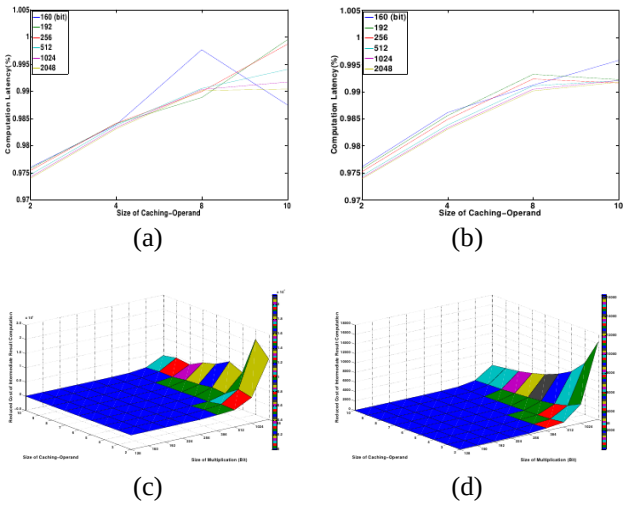

Fig. 5. Implementation result over 8-bit microprocessor(Proposed / Previous). (a)Operand-Caching. (b)Consecutive Operand-Caching. The number of reduced cost of intermediate result computation. (c)Operand-Caching. (d)Consecutive Operand-Caching.

B. Evaluation Result with Embedded Microprocessors

We implemented 160-bit multiplication over ATmega128. Our method shows better results in all multiplication methods compared with best known results. The detailed instruction counts are available in Table I.

TABLE I: INSTRUCTION COUNTS FOR A 160-BIT MULTIPLICATION ON THE ATMEGA128

Method	Load	Store	Mul	Add	Others	Total
Hybrid-Scanning						
Gura [5]	167	40	400	1,360	552	3,106
Uhsadel[8]	238	40	400	986	539	2,881
Scott[9]	200	40	400	1,263	108	2,651
Liu[10]	200	40	400	1,194	391	2,865
This Paper	168	40	400	1,336	79	2,629
Operand-Caching						
Michael[6]	80	60	400	1,240	70	2,395
This Paper	70	60	400	1,230	70	2,365
Consecutive Operand-Caching						
Seo[7]	70	60	400	1,240	56	2,356
This Paper	70	60	400	1,230	56	2,346

The "(Consecutive) Operand Caching" methods show best results in microprocessor, so we evaluate the methods with various sizes of bit. The detailed latency of "Operand Caching" and "Consecutive Operand Caching" is described in Table II. For clarity, we provide performance comparison between previous and proposed methods in Fig. 5 (a) and (b). The enhancement is measured from 0.5 % to 2.5 %. These features are generated with advantages from reduced number of intermediate result computation and detailed information is described in Fig. 5 (c) and (d).

TABLE II: CONSECUTIVE OPERAND-CACHING MULTIPLICATION RESULTS USING PROPOSED METHOD WITH DIFFERENT PARAMETERS

Operand Caching				
Size	e=2	e=4	e=8	e=10
160	3,821	2,917	2,507	2,365
192	5,475	4,187	3,537	3,467
256	9,671	7,411	6,275	6,115
512	38,295	29,427	24,987	24,171
1024	152,375	117,235	99,659	96,125
2048	607,863	467,955	397,995	383,471
Consecutive Operand Caching				
Size	e=2	e=4	e=8	e=10
160	3,674	2,838	2,472	2,346
192	5,296	4,088	3,490	3,437
256	9,428	7,272	6,200	6,128
512	37,796	29,128	24,800	24,205
1024	151,364	116,616	99,248	95,975
2048	605,828	466,696	397,136	384,058

V. CONCLUSION

The previous best known method reduced the number of memory accesses efficiently. However intermediate result computation is also expensive operation for multi-precision multiplication, so we focused more on intermediate result process. Improvement of the process enhances performance of ordinary multi-precision multiplications excluding "product-scanning", so the method affects wide range of multiplication methods. In this paper, we presented a novel method, i.e., "carry-once", for intermediate result computation of multi-precision multiplication. This method efficiently reduces an instruction counts by gathering several carry values and updating the values at once. This is efficiently implemented and show higher performance than previous result by up to 2.5 %.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0026621).

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Comm. ACM 21*, vol. 2, 1977, pp. 120-126.

[2] D. Hankerson, A. Menezes, and S.Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag, 2004.

[3] M. Scott, "Implementing cryptographic pairings," *Pairing-Based Cryptography Pairing* 2007, vol. 4575, pp. 177-196, 2007.

[4] P. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM Systems Journal* vol. 29, no. 4, pp. 526-538, 1990.

[5] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," *LNCS*, CHES, vol. 3156, pp. 119-132, 2004.

[6] M. Hutter and E. Wenger, "Fast Multi-precision Multiplication for Public-Key Cryptography on Embedded Microprocessors," *LNCS*, CHES, vol. 6917, pp. 459-474, 2011.

[7] H. Seo and H. Kim, "Multi-precision Multiplication for Public-Key Cryptography on Embedded Microprocessors," presented at WISA, 2012.

[8] L. Uhsadel, A. Poschmann, and C. Paar, "Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes," presented at the 4th European Workshop on Security and Privacy in Ad-hoc and Sensor Networks, ESAS 2007, Cambridge, UK, July 2-3, 2007.

[9] M. Scott and P. Szczechowiak. (2007). Optimizing Multiprecision Multiplication for Public Key Cryptography. Cryptology ePrint Archive, Report 2007/299, [Online]. Available: http://www.eprint.iacr.org/

[10] Z. Liu, J. Grobshadl, and I. Kizhvatov, "Efficient and Side-Channel Resistant RSA Implementation for 8-bit AVR Microcontrollers," *Workshop on the Security of the Internet of Things - SOCIOT 2010*, presented at the 1st International Workshop, Tokyo, Japan, Nov. 29. IEEE Computer Society, Los Alamitos, 2010.

Hwajeong Seo received the BSEE degree from Pusan National University, Pusan, Republic of Korea in 2010, and he received the MS degree program in department of Computer Engineering at Pusan National University in 2012. He is Ph. D course in same major and university. His research interests include sensor networks, information security, Elliptic Curve Cryptography, and RFID security.

Howon Kim received the BSEE degree from Kyungpook National University, Daegu, Republic of Korea, in 1993 and the MS and PhD degrees in electronic and electrical engineering from the Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea, in 1995 and 1999, respectively. From July 2003 to June 2004, he studied with the COSY group at the Ruhr-University of Bochum, Germany. He was a senior member of the technical staff at the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Republic of Korea. He is currently working as an associate professor with the Department of Computer Engineering, School of Computer Science and Engineering, Pusan National University, Pusan, Republic of Korea. His research interests include RFID technology, sensor networks, information security, and computer architecture. Currently, his main research focus is on mobile RFID technology and sensor networks, public key cryptosystems, and their security issues. Dr. Kim is a member of the IEEE, and the International Association for Cryptologic Research (IACR).