The Optimal Split Method for Large Integer Multiplication on Low-End Devices

Ren-Junn Hwang, Loang-Shing Huang, and Feng-Fu Su
Department of Computer Science and Information Engineering
Tamkang University
Taiwan (R.O.C.)
junhwang@ ms35.hinet.net

Abstract

This paper proposes a "recursive-balanced-2-way split" method, which is an optimal method for large integer multiplication on mobile devices and smart low-end devices when implementing modern cryptographic applications. The proposed method is based on the divide-and-conquer concept. The proposed method first recursively bisections multiplier and multiplicand in threshold times. Subsequently, classical multiplication calculates the products of the split multiplier and multiplicand blocks. Finally, the products of the blocks are gradually integrated to obtain the product of the large integers. This study demonstrated that the *n*-times recursive-balanced-2-way split method, where *n* is the floor of $\log_2(0.13515\times s)$, obtains the optimal performance in multiplying two *s*-words based on classical multiplication. This method can be implemented by recursive call procedures, and reduces code size and computational cost substantially. The proposed scheme is an energy-saving method to implement security protocols in mobile devices and low-end devices. Therefore, it is suitable for realizing modern public-key cryptosystems on low-end devices, in which the framework is based on modular exponentiation and modular multiplication. The experiment results show that modular exponentiation combined with other modular multiplication methods uses 1.28x-2.10x the computational cost required in the proposed method for the bit length of the modulus from 1024 bits to 4096 bits on Texas Instruments TMS320C55x DSP. Low-end devices based on the proposed method perform security protocols and PKI functions practically and satisfy the security recommendations of NIST.

***Keywords*:** Classical multiplication, Divide-and-conquer, Karatsuba-Ofman method, Public-key cryptosystem.

1 Introduction

Current mobile communication technology has changed our lives with a wide variety of smart low-end devices. The kernel of a smart low-end device is usually one or more than one RISC-like processor. These RISC-like processors, featuring single-precision multiplication, addition, store, and load instructions as their word-based architectures, have special features, such as low computation and low storage, and must be carefully considered when developing particular applications. A number of special applications, especially modern cryptography, require an operation of integers longer than 100 decimal digits. For example, because of the progress in integer factorization and computer power [1], the National Institute of Standards and Technology (NIST) in the United States changed their recommended key sizes for users or devices from 1024-bit to 2048-bit for the RSA digital signature or key transportation and Diffie-Hellman key agreement [2][3]. Such integers are too long to fit into a single word of the modern microprocessor. In addition, large computation is required to realize modern cryptography; therefore, more storage space is required. A particular approach must be determined to achieve reasonable performances on such an application.

Some applications, such as pervasive computing [4] and mobile multimedia [6], need some cryptographic procedures to keep secrets. Numerous security protocols [6][7], , PKI functions (Public Key Infrastructure), and cryptographic methods are based on modular exponentiation computation, including RSA cryptosystem [8], elliptic curve cryptography [9], DSA [10], and Diffie-Hellman Key Exchange [11]. The modular exponentiation procedure requires several modular multiplications. The kernel operation is a large integer multiplication for modular exponentiation or modular multiplication. The proportion of instructions spent on a large integer multiplication is 86% for RSA-1024, 63% for ECC-160, and 90% for ECC-163 [12][13]. Consequently, this study focused on accelerating large integer multiplication operations.

Two methods are suitable for implementing large integer multiplications on modern microprocessors. One method is classical multiplication, which is easy to implement, but exhibits inferior performance when the size of the operand becomes increasingly larger. The other method is the Karatsuba-Ofman method [14], which splits the operand into two equally small blocks and constructs them with classical multiplication. When the operand is sufficiently large, the Karatsuba-Ofman method exhibits superior performance in comparison to direct classical multiplication. However, the performance of large integer multiplication can only improve to a limited extent because the Karatsuba-Ofman method only splits the operand once.

This study researched a number of basic methods to split the operand according to the divide-and-conquer concept, synthesized them into various methods, and analyzed their performances before deciding on the optimal method for large integer multiplications. Moreover, this study also determined the threshold time

to split the operands. Based on these two crucial decisions, large integer multiplication achieved optimal performance on microprocessors. The energy consumption of software is closely related to execution time [15]. The proposed method enables low-end and mobile devices to consume less energy to perform security protocols.

Section 2 presents reviews and analysis of some related works. The proposed method is described in Section 3, followed by the analysis and demonstration of the proposed method in Section 4, and simulations and experiments in Section 5. Finally, Section 6 offers conclusions.

2 Analyses of Related Works

This section introduces two multiplication algorithms. Subsections 2.1 and 2.2 introduce and analyze the classical multiplication and the Karatsuba-Ofman method, respectively.

2.1. Classical Multiplication

Classical multiplication is also called the "Pencil-and-Paper" method. This method is multiplication on operand scanning. The idea of this Pencil-and-Paper method is shown in Algorithm 1 [16].

Classical multiplication is easy to implement because no extra carry bit is present. Therefore, classical multiplication is easy to implement in high-level programming languages that provide a double-precision integer data type. For example, common extensions of the C and C++ programming languages support the unsigned data type for 64-bit integers. Java language provides the long type, which has a precision of 64 bits on all platforms [13].The word complexity of classical multiplication is $O(s^2)$ [17], where s is the word length of the operand.

Algorithm 1. Classical multiplication (Pencil-and-Paper method)

Input: Integers $A = (a_{s-1}a_{s-2}\ldots a_1a_0)_w$, $B = (b_{s-1}b_{s-2}\ldots b_1b_0)_w$.
Output: The product $P = A \times B = (p_{2s-1}p_{2s-2}\ldots p_1p_0)_w$ in radix w representation.
1. $P \leftarrow 0$
2. **for** i from 0 to $s-1$ **do**
3. $u \leftarrow 0$
4. **for** j from 0 to $s-1$ **do**
5. $(u,v) \leftarrow a_j \times b_i + p_{i+j} + u$
6. $p_{i+j} \leftarrow v$
7. **end for**
8. $p_{s+i} \leftarrow u$
9. **end for**

Table 1. The number of base instructions for two s-words classical multiplication [14]

Algorithm	# Mul	# Add	# Load	# Store	#Total
Classical Multiplication	s^2	$4s^2$	$2s^2 + s$	$s^2 + s$	$8s^2 + 2s$

Table 1 [18] shows the number of base instructions for classical multiplication, for which each base instruction requires the same CPU cycle in RISC CPUs such as ARM. Classical multiplication exhibits excellent performance when the operand length is short. However, the performance of classical multiplication degenerates rapidly with the increase of operand length.

2.2. Karatsuba-Ofman Method

Karatsuba and Ofman proposed the Karatsuba-Ofman method [14]. This method reduces the multiplication of two s-word operands to three multiplications of size (s/2), but only at the cost of an increased number of large integer additions or subtractions. Each of these three (s/2)-word multiplications can be used with classical multiplication. According to the Karatsuba-Ofman method, the product $P = A \times B = [A_h \parallel A_l] \times [B_h \parallel B_l]$ can be expressed as the following two equations, where the bit length of one word is w:

$$P = A \times B$$
$$= A_hB_h2^{sw} + (A_hB_l + A_lB_h)2^{sw/2} + A_lB_l2^0 \quad (1)$$
$$= A_hB_h\,2^{sw} + [(A_h + A_l)(B_h + B_0) - A_hB_h - A_lB_l\,]2^{sw/2} + A_lB_l2^0 \quad (2)$$

When the two equations are compared, Equation (2) has one less (s/2)-word multiplication, two more (s/2)-word additions, one more s-word addition/subtraction, and

Algorithm 2. Large integer addition [12]

Input: Integers $A = (a_{s-1}a_{s-2}\ldots a_1a_0)_w$, $B = (b_{s-1}b_{s-2}\ldots b_1b_0)_w$.
Output: (ε, C) where $C = A + B \bmod 2^{sw} = (c_{s-1}c_{s-2}\ldots c_1c_0)_w$ in radix w representation, ε is the carry bit, and 1 word has w bits.
1. $(\varepsilon, c_0) \leftarrow a_0 + b_0$
2. for i from 0 to $s-1$ do
3. $(\varepsilon, c_i) \leftarrow a_i + b_i + \varepsilon$
4. end for
5. return (ε, C)

Table 2. The number of base instructions for a large integer addition/subtraction

Algorithm	# Add/Sub	# Load	# Store	#Total
Multipresion Addition/Subtraction	2s	2s	s	5s

Table 3. the number of classical mul., large integer add./sub. for the Karatsuba-Ofman method

Algorithm	Classical multiplication		Addition/Subtraction	
	Length (word)	#	Length (word)	#
Karatsuba-Ofman method	s/2	3	s	3
			s/2	2

Table 4. The number of base instructions for the Karatsuba-Ofman method

Algorithm	# Mul	# Add/Sub	# Load	# Store	#Total
Karatsuba-Ofman method	$(3/4)s^2$	$3s^2 + 8s$	$(3/2)s^2 + (19/2)s$	$(3/4)s^2 + (11/2)s$	$6s^2 + 23s$

more memory accesses because it contains more elements. For the large value of s, the cost of the additions, subtractions, and memory accesses are insignificant relative to the cost of the multiplications.

The word complexity of the Karatsuba-Ofman method is $O(s^{\log_2 3})$ [17], which exhibits superior performance to classical multiplication. To discover the computational cost of the Karatsuba-Ofman method, this study estimated the number of base instructions for a large integer addition. This study assumed that addition and subtraction operations consume the same computational cost. Algorithm 2 is a large integer addition. Table 2 shows the number of base instructions in Algorithm 2. Table 3 shows the number of classical multiplications and additions/subtractions for the Karatsuba-Ofman method. Finally, Table 4 combines Table 1, Table 2, and Table 3 to demonstrate the number of base instructions for the Karatsuba-Ofman method. A comparison of Tables 1 and 4 reveals that the Karatsuba-Ofman method is more efficient than classical multiplication for large integer multiplication.

3 The Proposed Method

To clearly describe the proposed method, this study defines the following notations:

- $|A|$: denotes the word length of integer A.
- There are w bits in one word.
- A_{high}, A_{low}: denote $(A/2^{(sw/2)})$ and (A mod $2^{(sw/2)}$), respectively, where $|A| = s$.

Modern microprocessors and digital signal processors have internal data pathways and internal registers with a fixed number of bits. A 32-bit processor will process 32-bit words, a 64-bit processor will process 64-bit words. The internal registers are sized to accommodate the word length with a single transfer. Generally, the more bits a processor can handle, the faster it can run. The instructions of microprocessors and digital signal processors are entered or stored in a storage device. Therefore, all word-sized transfers must have the operands in memory residing on word-boundaries. Operations on word boundary are more efficient than those on bit level, because processors possess word-based architectures. This section proposes an optimal method for modern microprocessors and digital signal processors based on the divide-and-conquer concept to accelerate the speed of large integer multiplication. The proposed method recursively bisections multiplier and multiplicand n ($= \lfloor \log_2(0.13515\times s) \rfloor$) times. Subsequently, classical multiplication evaluates the products of the split multiplier and multiplicand blocks. Finally, the products of the blocks are gradually integrated to obtain the product of the large numbers. Algorithm 3 details the proposed method, named "recursive-balanced-2-way split". Subsection 4.2 demonstrates that the proposed $n(= \lfloor \log_2(0.13515\times s) \rfloor)$-times recursive-balanced-2-way split method achieves optimal performance.

Algorithm 3. The recursive-balanced-2-way split

Function: *Mult*(n, A, B)
// **Input:** Integers $A = (a_{s-1}a_{s-2}\ldots a_1a_0)_w$, $B = (b_{s-1}b_{s-2}\ldots b_1b_0)_w$ in radix w representation, $A = A_{high}2^{sw/2} + A_{low}$, $B = B_{high}2^{sw/2} + B_{low}$ where $|A| = |B| = s$, one word has w bits, predetermined $n = \lfloor \log_2(0.13515\times s) \rfloor$.
// **Output:** The 2s-word result of the product of A and B.
If $n > 0$, **then**
// recursively bisection operands.
Height_part= *Mult*($(n-1)$, A_{high}, B_{high});
Low_part= *Mult*($(n-1)$, A_{low}, B_{low});
Return (*Height_part*$\times 2^{sw}$ + [*Mult*($(n-1)$, (A_{high} + A_{low}), (B_{high} + B_{low})) − *Height_part* − *Low_part*] × $2^{sw/2}$ + *Low_part*)
else
// perform the classical multiplication to multiply A and B.
Return ($A \times B$).
Endif.

4 Analysis of the Proposed Method

This section demonstrates that the performance of the proposed method, recursive-balanced-2-way split, is optimal for large integer multiplications on microprocessors and digital signal processors, which perform each word operation in one cycle, such as RISC-like processors. Subsection 4.1 presents the definition and evaluation of two basic methods for splitting the operand: balanced-t-way split and unbalanced-t-way split. Subsection 4.2 demonstrates that the $n(=\lfloor \log_2(0.13515\times s) \rfloor)$-times recursive-balanced-2-way split method achieves optimal performance in multiplying two s-words based on classical multiplication. Without loss of generality, this study paper assumed that both multiplier and multiplicand are s-word large integers, and each word is w bits.

4.1. Two Basic Methods

Two basic methods are used to split the operand, that is, balanced-t-way split and unbalanced-t-way split. Synthesizing these two methods can generate any model for splitting operands for multiplication.

4.1.1. Balanced-t-way Split

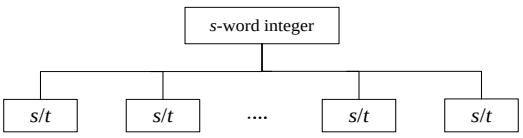

Fig. 1. Balanced-t-way split

Figure 1 shows the framework of splitting an s-word integer into t equal parts. The balanced-t-way split performs Equation (3) to evaluate the product of large

integers A and B from the split blocks. The multiplication, such as A_iB_j, is classical multiplication with (s/t)-words.

$$P = A \times B$$
$$= \left(\sum_{i=0}^{t-1} A_i 2^{i\times\frac{sw}{t}}\right)\times\left(\sum_{j=0}^{t-1} B_j 2^{j\times\frac{sw}{t}}\right)$$
$$= A_{t-1}B_{t-1}2^{(2t-2)(sw/t)} + (A_{t-1}B_{t-2} + A_{t-2}B_{t-1})2^{(2t-3)(sw/t)} + \cdots + (A_{t-1}B_1 + A_{t-2}B_2 + \cdots + A_1B_{t-1})2^{(t)(sw/t)} + (A_{t-1}B_0 + A_{t-2}B_1 + \cdots + A_0B_{t-1})2^{(t-1)(sw/t)} + (A_{t-2}B_0 + A_{t-3}B_1 + \cdots + A_0B_{t-2})\,2^{(t-2)(sw/t)} + \cdots + A_0B_02^0 \quad (3)$$

According to the Karatsuba-Ofman method, each cross term in Equation (3) can be rewritten as the following non-cross term:

$A_iB_j + A_jB_i = (A_i + A_j)(B_i + B_j) - A_iB_i - A_jB_j$, where $i \neq j$.

After transforming each possible cross term, Theorem 1 demonstrates the number of base instructions using balanced-t-way split to compute the large integer multiplication of two s-word integers. Due to the limitations of the number of pages, the paper omitted the proofs of the following theorems. Readers interested in understanding the proofs, please contact the authors.

Theorem 1. Using balanced-t-way split to multiply two s-word integers, the number of base instructions is $\frac{4t+4}{t}s^2 + (21t - 19)s$, where $t \geq 2$.

4.1.2. Unbalanced-t-way Split

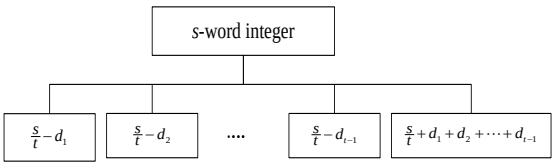

Fig. 2. Unbalanced-t-way split

In the unbalanced-t-way split method, the s-word integer is split into t unequal smaller parts, such as the example shown in Figure 2. Theorem 2 demonstrates the number of base instructions in unbalanced-t-way split.

Theorem 2. The sequence $\{d_1, \cdots, d_{t-1}\}$satisfies the following conditions:
(1) at least one $d_i > 0$
(2) $d_1 + d_2 + \cdots + d_{t-1} > 0$
(3) If $d_i > 0$, then $d_j \geq d_{j-1}$ for $j = i, i-1, \cdots, 2$
(4) If $d_i < 0$, then $d_j \geq d_{j+1}$ for $j = i, i+1, \cdots, t-2$.

Then number of base instructions is $\frac{4t+4}{t}s^2 + \{(21t - 19) + \frac{16}{t}[(t-1)d_1 + (t-2)d_2 + \cdots + d_{t-1}]\}s + \{42[(t-1)d_1 + (t-2)d_2 + \cdots + d_{t-1}] + 8[d_1^2 + 2d_2^2 + \cdots + t(d_1 + d_2 + \cdots + d_{t-1})^2]\}$ using unbalanced-t-way split to multiply two s-word integers.

Proof. See Appendix 2.

Corollary 1. The term $(t-1)d_1 + (t-2)d_2 + \cdots + d_{t-1}$ in Theorem 2 is always greater than 0.

4.2. The Optimal Method

This subsection presents the synthesis of two basic split methods from Subsection 4.1 to generate various multiplication methods. Theorem 7 demonstrates the most efficient method. Two generalized models are used for split way multiplication, as follows:
n-times recursive-balanced-t-way split and n-times recursive-unbalanced-t-way split.

Theorem 3 evaluates the number of base instructions of n-times recursive-balanced-t-way split.

Theorem 3. Using the n-times recursive-balanced-t-way split, the number of base instructions to multiply two s-word integers is $8\left(\frac{t+1}{2t}\right)^n s^2 + 42\left(\frac{t+1}{2}\right)^n s - 40s$, where $t \geq 2$, $n \geq 1$ and $t^n \leq s$.

Theorem 4 finds the n that results in optimal performance of n-times recursive-balanced-t-way split.

Theorem 4. The n-times recursive-balanced-t-way split performs the minimal number of base instructions as $n = \left\lfloor \log_t \frac{4s[\ln(2t) - \ln(t+1)]}{21[\ln(t+1) - \ln 2]} \right\rfloor$.

Using Theorems 3 and 4, Figure 3 plots the curve of the number of base instructions of n-times recursive-balanced-t-way split, that is, $8\left(\frac{t+1}{2t}\right)^n s^2 + 42\left(\frac{t+1}{2}\right)^n s - 40s$ when $n = \log_t \frac{4s[\ln(2t) - \ln(t+1)]}{21[\ln(t+1) - \ln 2]}$, as $1 \leq s \leq 2000$, $t \geq 2$. The recursive-balanced-2-way split,

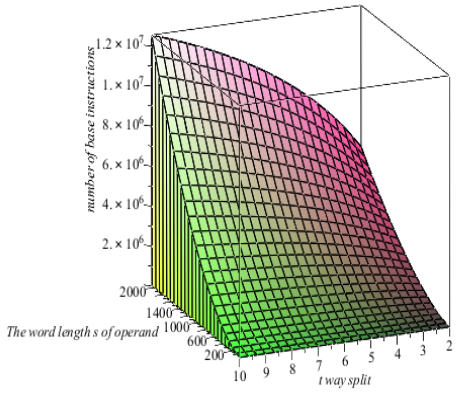

Fig. 3. Plot of the number of base instructions of n-times recursive-balanced-t-way split as $1 \leq s \leq 2000$, $t \geq 2$ and $n = \log_t \frac{4s[\ln(2t) - \ln(t+1)]}{21[\ln(t+1) - \ln 2]}$

that is, (t = 2), results in superior performance in comparison to recursive-balanced-t-way split as t > 2. Theorem 5 demonstrates that using recursive-balanced-2-way split to multiply two s-word integers is more efficient than using recursive-balanced-t-way split as t > 2.

Theorem 5. Using recursive-balanced-2-way split to multiply two long s-word integers is more efficient than using recursive-balanced-t-way split when t > 2.

Subsequently, Lemma 1 and Theorem 6 demonstrate that using recursive-balanced-t-way split is superior to recursive-unbalanced-t-way split, and Theorem 7 concludes that recursive-balanced-2-way split is the optimal split method to achieve optimal performance for implementing the large integer multiplication of two s-word integers by using the classical multiplication.

Lemma 1. Using balanced-t-way split to multiply two s-word integers is more efficient than using unbalanced-t-way split.

Theorem 6. Using recursive-balanced-t-way split to multiply two s-word integers is more efficient than using recursive-unbalanced-t-way split.

Theorem 7. Using $n(= \lfloor \log_2(0.13515 \times s) \rfloor)$-times recursive-balanced-2-way split to multiply two s-word integers is more efficient than using all of the other split methods by using the classical multiplication.

Proof. The results of Theorems 5 and 6 show that the recursive-balanced-2-way split multiplies two long s-word integers is more efficient than the recursive-balanced-t-way split (when t > 2) and the recursive-unbalanced-t-way split. By Theorem 4, the $n(= \lfloor \log_2(0.13515 \times s) \rfloor)$-times recursive-balanced-2-way method achieves optimal performance in multipling two s-word integers by using the classical multiplication.

5 Experiments And Discussions

In order to confirm theoretical results, this study used assembly language to implement a number of experiments on the Code Composer Studio (CCS) platform (TI DSP C55x family platform [19]) without requiring any special coding skills. The TI DSP C5510 family is an RISC-like processor (1 word = 16 bits), which typically has low computational power and memory; therefore, it is included in several smart devices.

5.1. Experiments on the large integer multiplication

These experiments implemented large integer multiplication from 8 words (1 word = 16 bits) to 512 words using classical multiplication, the balanced-2-way (i.e., Karatsuba–Ofman method), balanced-t-way split, recursive-balanced-2-way split, and unbalanced-t-way split. Table 5 shows the comparison among performances of classical multiplication, balanced-2-way split, balanced-t-way split, and recursive-balanced-2-way split. Table 6 shows the results of experiments on the performance of unbalanced methods of splitting the operand.

A comparison of Table 5 with 6 revealed that the proposed recursive-balanced-2-way split exhibited superior performance as the demonstrated result of Theorem 7. The application of the proposed method results in substantial improvement to the performance of large integer multiplication when the size of the operand increases. According to the results of these experiments and the teoretical results of Section 4, the proposed recursive-balanced-2-way split method exhibits optimal performance.

Table 5.Numberof CPU cycles for the large integer multiplication

Length (bit)	Classical mul.	Balanced-2-way	Balanced-4-way	**The proposed scheme[1]**
8192	2,140,139	1,663,872	1,478,400	**657,483**
4096	535,296	419,059	378,752	**215,406**
2048	137,896	116,807	99,264	**69,924**
1024	34,543	30,015	27,104	**22,369**
512	8,669	7,911	7,920	**6,987**
256	2,184	2,081	2,552	**2,081**
128	539	596	924	**539**

NOTE.1. the proposed scheme is $n(= \lfloor \log_2(0.13515 \times s) \rfloor)$-times recursive-balanced-2-way

Table 6. Number of CPU cycles for the multiplication with unbalanced methods

Length (bit)	Unbalanced 2way Split ratio: 1:3	Unbalanced 3way Split ratio: 1:1:2	Unbalanced 4way Split ratio: 1:1:2:4	Unbalanced 5way Split ratio: 1:1:1:1:4
8192	2,883,648	2,293,081	3,030,700	3,460,672
4096	725,843	581,292	771,107	882,464
2048	183,926	149,334	199,492	229,264
1024	47,214	39,339	53,231	61,640
512	12,420	10,837	14,986	17,572
256	3,413	3,210	4,586	5,474
128	1,007	1,053	1,566	1,909

5.2. Experiments on modular multiplication and modular exponentiation

Modular multiplication is the kernel of modular exponentiation, which is the main operation of several security protocols. For computing modular multiplication, this study considered whether multiplication and reduction are separated or integrated. The separated approach first multiplies two operands, and subsequently performs a Montgomery reduction, that is, KCM method [18] while alternating between multiplication and reduction in the integrated approach (i.e., CIOS method [20]). This study used the separated method to implement modular multiplication with the proposed recursive-balanced-2-way method, and compared the performance with other modular multiplication methods in Table 7. As shown in Table 8, modular exponentiation was used to compare other modular multiplication methods with modular multiplication for the proposed

method. Table 9 shows the increase in speed of modular exponentiation with the proposed recursive-balanced-2-way method versus that with other modular multiplication methods. Other modular multiplication methods require 1.28x-2.10x the computational cost required with the proposed recursive-balanced-2-way method for the processors of smart devices. Moreover, 1.58x-1.94x the computational cost is required for other modular multiplication methods versus modular multiplication with the proposed method when the bit length at 3072 satisfies the NIST recommendations. The energy consumption of software is closely related to execution time. The proposed scheme is an energy-saving method to implement security protocols in mobile devices and low-end devices.

Table 7. Number of CPU cycles for integrated Montgomery modular multiplication and the proposed recursive-balanced-2way method

Length (bit)	#Cycles of Algorithms		
	CIOS[1] [20]	KCM[2] [18]	**The proposed scheme[3]**
8192	2,402,948	1,933,718	**927,329**
4096	597,148	448,150	**284,804**
3072	339,255	275,914	**174,928**
2048	151,132	135,133	**88,250**
1024	38,275	35,093	**27,447**

Note:
1. Alternating between multiplication and reduction.
2. Performing Karatsuba–Ofman method to multiply the operand first and then perform the Montgomery reduction [18].
3. Performing the proposed recursive-balanced-2-way method to multiply the operand first and then perform the Montgomery reduction.

Table 8. Number of CPU cycles for modular exponentiation with different modular multiplications where the bit lengths of base, modulus, and exponent are the same length

Length (bit)	#Cycles of Algorithms		
	CIOS [20]	KCM [18]	**The proposed scheme**
4096	3,668,877,312	2,753,433,600	**1,749,835,776**
3072	1,563,287,040	1,271,411,712	**806,068,224**
2048	464,277,504	415,128,576	**271,104,000**
1024	58,790,400	53,902,848	**42,158,592**

Table 9. Speedup of modular exponentiation with the proposed method vs. other modular multiplications where the bit lengths of base, modulus, and exponent are the same length (the CPU Cycles of other modular reductions/ the CPU Cycles of the proposed method)

	1024 bits	2048 bits	3072 bits	4096 bits
CIOS[20]	139%	171%	194%	210%
KCM [18]	128%	153%	158%	157%

6 Conclusions

This paper proposes a recursive-balanced-2-way split method that uses the divide-and-conquer concept to split the operand to ensure that long integer multiplication is executed as quickly as possible. This study demonstrated that the proposed "$n(= \lfloor \log_2(0.13515 \times s) \rfloor)$-times recursive-balanced-2-way split" method achieved the optimal performance for multiplying two s-words based on classical multiplication. This method can be easily implemented by recursive call procedures to substantially reduce the complexity of programming flow. Modular exponentiations with other modular multiplication methods require 1.28x-2.10x the computational cost required with the proposed recursive-balanced-2-way method for TI DSP TMS320C55x family with bit length ranging from 1024 to 4096. The proposed scheme is an energy-saving method to implement security protocols in mobile devices and low-end devices. The proposed method substantially improves the performance of large integer multiplication for processors of low-end devices. Smart low-end devices using the proposed method can perform security protocols and PKI functions practically, and satisfy the security recommendations of NIST.

References

[1] RSA Laboratories, "RSA challenges," Available from URL: http://www.rsasecurity.com/rsalabs/.

[2] NIST, "SP 800-57 Part 1, Recommendation for Key Management," NIST Special Publication 800-57, March 2007, Available from URL: http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf/.

[3] NIST: "SP 800-57 Part 3, Recommendation for Key Management," NIST Special Publication 800-57, August 2008, Available from URL: http://csrc.nist.gov/publications/drafts/800-57-part3/Draft_SP800-57-Part3_Recommendationforkeymanagement.pdf.

[4] S. Silas, K. Ezra, E. B. Rajsingh, "A novel fault tolerant service selection framework for pervasive computing," *Human-centric Computing and Information Sciences* 2012, 2:5.

[5] H. Luo, M. Shyu, "Quality of service provision in mobile multimedia - a survey," *Human-centric Computing and Information Sciences* 2011, 1:5.

[6] M. D. Kettani, B. En-Nasry, "MidM : an Open Architecture for Mobile Identity," *Journal of Convergence*, vol.2, no.2, pp.25- 32, 2011.

[7] Y. Li, L. Xiao, S. Chen, H. Tian, L. Ruan, and B. Yu, "Parallel Point-multiplication based on the Extended Basic Operations on Conic Curves over Ring Z_n," *Journal of Convergence*, vol.2, no.1, pp.69-78, 2011.

[8] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communication on ACM*, vol.21, pp.120-126, 1978.

[9] Koblitz, N., "Elliptic curve cryptosystems," *Math. Comp*. 48, 203-209, 1987.

[10] NIST, "FIPS PUB 186-3, Digital Signature Standard (DSS) ," NIST Publication FIPS 186-3, June 2009, Available from URL: http://csrc.nist.gov/publications/fips/fips186-3/.pdf.

[11] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. On Information Theory*, vol.IT-22, no.6, pp.638-654, Nov. 1976.

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

[13] H. Eberle, S. Shantz, V. Gupta, N. Gura, L. Rarick, and L. Spracklen, "Accelerating Next-Generation Public-Key Cryptosystems on General-Purpose CPUs," *IEEE Macro*, vol. 25, issue 2, pp 52-59, 2005.

[14] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Phys*. Doklagy, vol.7, no.7, pp595-596, 1963.

[15] C. Lederer, R. Mader, M. Koschuch, J. Grobschadl, A. Szekely, and S. Tillich, "Energy-efficient implementation of ECDH key exchange for wireless sensor networks," WISTP 2009, LNCS 5746, pp. 112-127, 2009.

[16] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[17] D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley, 1969, 2nd edition, 3rd edition, 1998.

[18] J. Groβschädl, R. M. Avanzi, E. Savas, and S. Tillich: "Energy-efficient software implementation of large integer modular arithmetic," *CHES 2005, LNCS 3659*, pp.75-90, 2005.

[19] Texas Instruments, "C5000 DSPs: Architecture & Peripheral Features," Available: http://www.ti.com.

[20] C. K. Koc, T. Acar, and B. S. Kaliski: "Analyzing and computing Montgomery algorithms," *IEEE Macro*, 16(3):26-33, 1996.