Malaysian Journal of Mathematical Sciences 7(S): 145-156 (2013)
Special Issue: The 3rd International Conference on Cryptology & Computer Security 2012 (CRYPTOLOGY2012)

MALAYSIAN JOURNAL OF MATHEMATICAL SCIENCES

Journal homepage: http://einspem.upm.edu.my/journal

Big-Digits Representation and its Application in Cryptography

***Shahram Jahani and Azman Samsudin**

School of Computer Sciences,
Universiti Sains Malaysia,
11800 Penang, Malaysia

E-mail: Jahani2001@yahoo.com and azman@cs.usm.my

*Corresponding author

ABSTRACT

The efficiency of number theory based cryptosystems correlatesdirectly to the efficiency of large integer multiplication operation. In this paper, we propose a hybrid of Karatsuba-Classical multiplication algorithm that is based on a Look-up table of "Big-Digits" representation. The Big-Digits representation is a more compact representation compared to the binary representation. Therefore, by using the Big-Digits representation, the number of sub-multiplication operations in a multiplication algorithm will reduce significantly. The results of this study show that the proposed multiplication algorithm, which is based on the Big-Digits representation, is faster than the classical, Karatsuba and the hybrid of Karatsuba-Classical multiplication algorithms in range of the public-key cryptography implementation.

Keywords: Public-key cryptography, Big-Integer calculation, and Karatsuba multiplication algorithm.

1. INTRODUCTION

Number theory based public-key algorithms use multiplication extensively in their operations. Subsequently, many mathematicians and computer scientists have devoted their time in devising efficient multiplying algorithms for large numbers. The best known algorithmsfor multiplying big numbers in order of complexity are: Schoolbook or classical multiplication algorithm (Knuth (1997),Karatsubamultiplication algorithm (Karatsuba and Ofman (1963)), Toom-Cook multiplication algorithm (Cook May (1966), Shönang-Strassenmultiplication algorithm (Schonhage and Strassen (1971), and Fürer (Fürer (2007)) multiplication algorithm.Related

work on big integer multiplication algorithms can also be found in the following literatures (Jedwab and Mitchell (1989), Montgomery (2005), Bodrato (2007), Hars (2007)).

Earlier researchers improved the multiplication algorithms efficiency by introducing algorithms with efficient complexity. However, there are other techniques that can be used to improve the multiplication algorithm efficiency. Manipulating the numbers representation (Reitwiesner (1960), Okeya *et al.* (2004)), representing integers in anew number system (Dimitrov *et al*. (1997); Avanzi *et al.* (2006), Maitra and Sinha (2007)) or using look-up table (Hasan (2000), Wen-Ching *et al.* (2008)) are some of the known techniques that had been used in improving multiplication algorithms.

This paper will first explain the two most important multiplication algorithms in cryptography; classical algorithm and Karatsuba algorithm. This will followed by a brief discussion on binary number representation before the introduction of the Big-Digits concept and the ZOT number representation through definitions and properties. The following section will show how ZOT improves the efficiency of the multiplication algorithms.

Classical Multiplication Algorithm

The usual method of multiplying two numbers in positional numeral system isknown as classical multiplication algorithm (Knuth 1997)which can be stated as follows (see Algorithm 1):

Algorithm 1: Classical Multiplication CL (A,B)

Input: $A = (a_p \dots a_0)_b$

$$B = (b_q \dots b_0)_b$$

Output: $C = (C_{p+q+1} \dots, C_1 C_0)_b$

1. **for** $(i = 0; i \leq q; i{+}{+})$
2. **if** $(i \neq 0)$ do
3. **for** $(j = 0; j \leq p; j{+}{+})$
4. $TMP = C_{i+j} + (a_i \times b_j) + CRRY;$
5. $CRRY = \lfloor TMP/b \rfloor; C_{i+j} = TMP \bmod b;$
6. $C_{q+p+1=CRRY};$
7. **return** C

There are two nested loops as shown in Steps 1 and 3 of Algorithm 1 to scan all q digits of the multiplier and p digits of the multiplicand. The maximum length of output C will be $q + p + 1$and each digit of C will be calculated as follow:

The value of TMP in Step 4,is computed by adding the value of $C_{i+j}$with the value of$CRRY$and the result of multiplying the non-zero digit of the multiplier, a_i, (Step 2) with digitb_j from the multiplicand. The values of quotient and reminder areextracted from *TMP*after divided by *b,* in Step 5 and are saved in c_{i+j} and *CRRY* respectively. In this way, all the output digits from the least to the most significant digit are consecutively computed. The complexity of this algorithm is$O(n^2)$, which is slow for applications such as cryptography.

Karatsuba Multiplication Algorithm

Karatsuba multiplication algorithm (Karatsuba and Ofman (1962)) with complexity $O(n^{1.58})$ is a suitable algorithm for multiplying numbers larger than few hundred digits long (Xianjin and Longshu (2007)). This algorithm performs based on two mechanisms; divide and conquer (Cormen *et al.* (2000), Levitin (2002), Mainzer (2007)) and binary splitting (Brent (1976)).

Equations 1 and 2 describe the multiplication operations, which the classical algorithm and Karatsubaalgorithm are based on, respectively. The number of partial products in each iteration of the Karatsuba algorithm is three while the classical algorithmhas four, which gives Karatsuba algorithm the extra advantage in its calculation.

$$a.b = (a_L.b_L)r^{2n} + (a_R.b_L + a_L.b_R)r^n + (a_R.b_R) \tag{1}$$

$$a.b = (a_L.b_L)r^{2n} + \{(a_R + a_L).(b_L + b_R) - (a_L.b_L) - (a_R.b_R)\}r^n + (a_R.b_R) \tag{2}$$

Algorithm 2 shows the recursive Karatsuba multiplication algorithm in detail. When the length of the numbers that are being multiplied is 1, the multiplication proses is a simple digit by digit multiplication (see Step 2). For numbers larger than 1 digit, the numbers are divided into a lower (a_R) and an upper half (a_L) as shown by Equation 3 before the algorithm is being call again recursively. The algorithm ends after $log_2 n$steps.

$$a = a_L \times r^{n/2} + a_R \text{ , } b = b_L \times r^{n/2} + b_R \tag{3}$$

Since Karatsuba multiplication algorithm run slower for numbers shorter than few hundred digits, some researchers (Von (2002)) had proposed a hybrid approach where Karasuba algorithm is combined with other multiplication methods. Another approach to improve the performance of Karasuba algorithm, is by splitting the numbers into more than 2 segments per iteration. Dan Zuras described 3-way and 4-way variations of the Karatsuba algorithm (Zuras (1994)), and these studies was later extended by M. Sadiq and A. Jawed (Sadiq and Ahmed (2006)) by splitting the numbers into 2-to-ten parts. Related work on Karatsuba algorithm can be found in these literatures (Montgomery (2005), Haining and Hasan (2007), Bernstein (2009)).

Algorithm 2: Karatsuba Multiplication C=KA (A,B)

Input: $A = (a_{n-1} \dots a_0)_r$

$B = (b_{n-1} \dots b_0)_r$

Output: KA (A,B)

1. **If** $n = 1$
2. **return** $A \times B$
3. **else**
4. $a_L = \left\lfloor \frac{a}{r^{n/2}} \right\rfloor$ *and* $a_R = a \bmod r^{n/2}$ // dividing a into two halves
   $b_L = \left\lfloor \frac{b}{r^{n/2}} \right\rfloor$ *and* $b_R = b \bmod r^{n/2}$ // dividing b into two halves
5. $k_0 = KA(a_L, b_L)$
6. $k_1 = KA(a_R, b_R)$
7. $k_2 = KA(a_R + a_L\ ,\ b_R + a_L)$
8. **return** $k_0 \times r^n + (k_2 - k_1 - k_0) \times r^{n/2} + k_1$

2. BIG-DIGITS REPRESENTATION

ZOT representation (Jahani (2009)) is a new representation for integers, which was derived from the binary numbering system. Symbols used in this representation are known as Big-Digits or in short "BD". The different patterns of "0" and "1" symbols are the foundation of ZOT. These patterns are described as follows:

- Big-Zero: A sequence of symbol "0" is identified as Big-Zero or BZ. We represent a BZwith length of $n$as Z_n.For example, $Z_3 = "000" = 0$ and $Z_1 = "0" = 0$.

- Big-One: A sequence of symbol "1" is identified as Big-One or BO. We represent a BO with length of n asO_n.The numerical value of each Big-

One could be obtainedby $O_n = \sum_0^{n-1} 2^i$.For example, $O_1 = \sum_0^{1-1} 2^i$ =1 and $O_7 = \sum_0^{7-1} 2^i = 1111111_2 = 127$.

- Big-Two: A sequence of symbols "10" with extra symbol "1" at the right side of the sequence is called Big-Two or BT. We represent a BT with length of n as T_n.It is clear from the definition that the length of BT is always odd and its numerical value can be obtained from $T_n = \sum_0^{(n-1)/2} 4^i$.For example, $T_5 = \sum_0^{(5-1)/2} 4^i = 10101_2 = 21$.

Big-Digits is not a unique representation. For example, the binary number of "11111" could be represented by O_5 , $O_4\, O_1$, $O_1\, O_4$or $O_3\, O_2$. ZOT representationlimits these varieties to only one representation. To convert a binary number to the ZOT representation the following rules must be considered.

- **Direction of scanning:**The direction of scanning a binary number to search for a new BD does not matter;however right-to-left is preferred.

- **Valid Big-Digit:**A valid Big-Digit in ZOT representation is a Big-Digit, which cannot be extendedwith any symbols, either tothe left or to the right of the BD. There is one exception; when a Big-One and a Big-Two are next to each other. In this situation the common "1" must belongs to BO.For example, the valid representation for "1111010101" is $O_4\, Z_1\, T_5$, not $O_3\, T_7$. More detail on ZOT representation can be found in (Jahani 2009).

For coding purposes ZOTis represented as shown by the following example:

$$\underbrace{11111}_{O_5}\,000\,\underbrace{1010101}_{T_7}\,000\,\underbrace{11111}_{O_5} = O_{(5,18)}\, T_{(7,8)}\, O_{(5,0)}$$

In above example, we can see all BZs disappeared and every non-zero BDs in the representation carry extra one more parameter. The parameters are the length and position of BD in its original binary form. In above example, $T_{(7,8)}$ means there is a BT with length 7 at position 8. This representation will prevent from double scanning of zeros while doing multiplication in ZOT representation.

Implementing Look-Up Table (LUT) in multiplication algorithm has its advantages (Hasan (2000), Mahboob and Ikram (2005), Wen-Ching *et al.* (2008)).To benefit from this technique the ZOT representation is modified to form another variant of ZOT known as ZOT_x, where x is the

upper limit forthe maximum size of non-zero BDs in the representation. In this case, the size of the multiplication LUT will be limited to x^2.The procedure for obtaining ZOT_x representation is similar tothe process of obtaining the ZOT representation, except that the maximum length of BDs must belimited to *x*bits.The following exampleclarifies this concept.

$$A = \overbrace{1111111000010101000111111}^{binary} = \overbrace{O_{(7,18)}\, T_{(5,9)}\, O_{(6,0)}}^{ZOT}$$
$$= \overbrace{O_{(2,23)} O_{(5,18)}\, T_{(5,9)}\, O_{(1,5)} O_{(5,0)}}^{ZOT_5}$$

3. KARATSUBA MULTIPLICATION ALGORITHM WITH ZOT_x REPRESENTATION

The ZOT_x has less non-zero digitsin its representation compared to its original binary representation. Hence, to multiply two ZOT_x numbers, less sub-multiplication operations is required. Classical multiplication algorithm, with some modification, can support the ZOT_x representation; as demonstrated by Algorithm 3.

Algorithm 3: Classical_ZOT_x Multiplication Algorithm

Input: $A = (a_p, \dots, a_1, a_0)_2$
$B = (b_q, \dots, b_1, b_0)_2$

Output: $C = CL - ZOT_x(A, B) = (c_{p+q+1} \dots, c_2, c_1, c_0)_2$

1. $ZOT_x(A) = a^* = a^*_p, \dots, a^*_1, a^*_0$;// where $a^*_n = (a^*_{nt}, a^*_{nl}, a^*_{np})$
2. $ZOT_x(B) = b^* = b^*_q, \dots, b^*_1, b^*_0$;// where $b^*_n = (b^*_{nt}, b^*_{nl}, b^*_{np})$
3. **for** ($i = 0; i \leq p; i{+}{+}$)
4. **for** ($j = 0; j \leq q; j{+}{+}$)
5. $c'_{a^*_{ip}+b^*_{jp}} = c'_{a^*_{ip}+b^*_{jp}} + BigDigitMultiplication(a^*_i, b^*_j)$;
6. C = Convert to binary (C');
7. **return** C

The first modification is the conversion step, converting binary numbers a and b to ZOT_x representation a^* and b^* (see Steps 1 and 2). In these steps, all BDs such as a^*_n,will be denoted by three additional parameters; type denoted bya^*_{nt}, length denoted by a^*_{nl}, and position of BD denoted bya^*_{np}.These conversions are actually the first overhead of the algorithm. The second modification is in Step 5. In this step, the function $BigDigitMultiplication(a^*_i, b^*_j)$ fetches the result of binary multiplication

of two Big-Digits a_i^*andb_j^* from a pre-calculated LUT. This value will be added to digit $c_{a_{ip}^*+b_{jp}^*}$, where $a_{ip}^* + b_{jp}^*$ addresses the position of the digit. Note that, there is no "carry" from the previous calculation being calculated in Step 5. Therefore, the pre-defined memory for each digit of the output must be big enough to support the summation value in Step 5. The third modification is related to the format of the output. Based on to the memory specified for each digit, the base of the output can be defined. For example if we consider *n* bytes for each digits. The base of the output is 2^n. In Step 6, the output is converted to binary.

Algorithm 4 shows the hybridof the Karatsubaalgorithm with the Classical-ZOT_xmultiplication algorithm. The only difference between Algorithms 2 and 4 is in Step 1.In this step, when the size of the numbers reach the cut-off point value, theClassical-ZOT_x multiplication algorithm will be used for the calculation.

Algorithm 4: Hybrid of Karatsubaand Classical-ZOT_x Multiplication Algorithm

Input: $\mathrm{A} = (\mathrm{a}_{n-1} \dots \mathrm{a}_0)_\mathrm{r}$

$\mathrm{B} = (\mathrm{b}_{n-1} \dots \mathrm{b}_0)_\mathrm{r}$

Output: $KA - ZOT_x(A, B)$

1. **If** $\mathrm{n} \leq$ cut_off point
2. **return** $(A \times B)_{Classical\_ZOTx}$
3. **else**
4. $a_L = \left\lfloor \frac{\mathrm{a}}{\mathrm{r}^{n/2}} \right\rfloor$ *and* $a_R = \mathrm{a} \bmod \mathrm{r}^{n/2}$ // dividing *a* into two halves
   $b_L = \left\lfloor \frac{\mathrm{b}}{\mathrm{r}^{n/2}} \right\rfloor$ *and* $b_R = \mathrm{b} \bmod \mathrm{r}^{n/2}$// dividing *b* into two halves
5. $\mathrm{k}_0 = \mathrm{KA}(\mathrm{a}_\mathrm{L}, \mathrm{b}_\mathrm{L})$
6. $k_1 = KA(a_R, b_R)$
7. $k_2 = KA(a_R + a_L\ ,\ b_R + a_R)$
8. **return** $k_0 \times r^n + (k_2 - k_1 - k_0) \times r^{n/2} + k_1$

In the following section,we compare the efficiency of the proposed multiplication algorithm with the existingclassical (CL), Karatsuba (KA) and hybrid of Karatsuba-Classical (KA-CL)multiplication algorithms in range of the public-key cryptography algorithms.

4. RESULTS

According to (Jahani (2009)), the Hamming weight for 32 bits to 32 Kbits random numbers (Matsumoto and Nishimura (1998)) is about 20% while the Hamming weight for binary number is 50%. Therefore, theoretically the number of partial multiplication for classical and Classical-

ZOT_x multiplication algorithm will be about$0.25n^2$and$0.04n^2$,respectively. Subsequently, the classical-ZOT_x multiplication algorithm should be about 6.25 times faster than the classicalmultiplication algorithm. Because of the overhead in converting the binary numbers to the ZOT_x representation andthe call to the function$BigDigitMultiplication$,the actual speed-up ratio is less than what is being speculated above. This paper investigates the effectiveness of combining the Karatsuba algorithm with the Classical-ZOT_x multiplication algorithm.

Random numbers that are being represented by Big-Digits have special distribution, which will help in determining the optimized size for the lookup table. Table1 shows that with higher value of x (the maximum length of Big-digits), more numbers can be converted to ZOT_x. Result in Table 1,which are based on the 50 different 8 Kbits random numbers, indicatesthat 99% of random number has less than 8 non-zero symbols.In general, depending on the application and available memory, we can increase or decrease the value of x. However, the proposed range (x=7)should covers 99% of the numbers. The other 1% of the numbers can be segmented into a few Big-Digits with length less than 7 bits.

TABLE 1: Distribution percentage of Big-Digits in a random numbers (8 Kbits)

x	1	2	3	4	5	6	7
Percentage	37%	67%	84%	92%	96%	98%	99%

Table 2 shows the measured execution time for each algorithm (CL, KA, KA-CL and KA-ZOT_x) within the range of 32 bits to 8 Kbits. The number of random numbers used for each test is 50 and the cut-off points were determined by experimenting with the KA-CL algorithm. The proposed algorithm was tested under the same conditions as other algorithms with the same cut-off points. The machine specification used in the experiment is as follows: AMD Phenom (TM) 9950 Quad-core CPU 2.6 GHz, 3.25GB RAM, Windows XP Professional version 2002 (Service Pack 3) OS and Dev-C++ version 4.9.9.2 compiler.

Table 2 shows that the hybrids algorithms have different cut-off points depending on the length of the number. The cut-off point value increases continuously against the length of numbers within the range of 32 to 128 bits. For numbers in the range of 1 Kbits up to 8 Kbits, the cut-off point stable at 16 bits.

TABLE 2: Execution time (msec) of multiplication algorithms

Algorithm	Length of numbers (bits)								
	32	64	128	256	512	1024	2048	4096	8192
CL	0.007	0.023	0.083	0.344	1.27	4.9	19.2	76.9	308.5
KA	0.021	0.064	0.193	0.592	1.77	5.4	15.9	48.3	142.9
KA-CL	0.008	0.024	0.078	0.267	0.82	2.5	7.6	23.3	71.4
ProposedAlgorithm (x=7)	0.005	0.010	0.020	0.094	0.28	1.4	4.1	13	33.3
Cut-offPoint	16	32	64	32	32	16	16	16	16

The results show that the performance of KA-ZOT_xmultiplication algorithm is better than CL, KA and KA-CL. The speed of KA-ZOT_7 is about1.4 times faster than CL and increases to 9.2 times faster for 8 Kbits numbers. Comparing the execution speed between KA-$ZOT_x$and KA, tells us that KA-$ZOT_x$is about 4.2 times faster for 32 bits number and increases to about 4.3 times faster for 8 Kbits numbers, with some fluctuation in between. Figure 1 also indicates that KA-ZOT_x is relatively faster than KA-CL. KA-ZOT_x is about 1.6 times faster to 2.9 times faster for multiplying numbers in the range of 32 bits to 8Kbits.

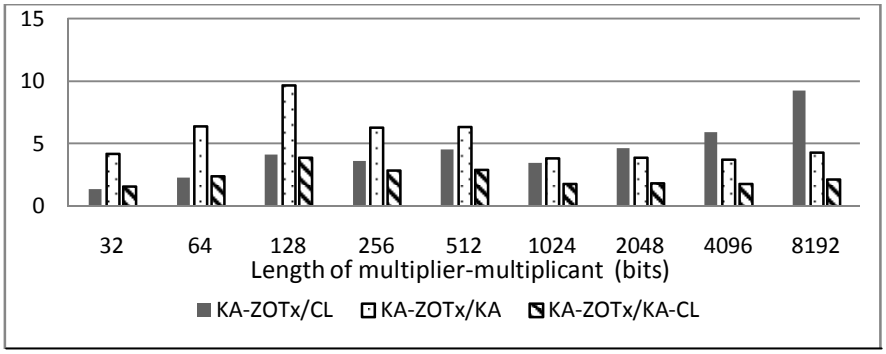

Figure 1: Comparison of KA-ZOT_x multiplication algorithm against CL,KA and KA-CL multiplication algorithms

5. CONCLUSION

In this paper, we proposed a new hybrid multiplication algorithm, combining the Karatsuba multiplication algorithmwith the ZOT_x multiplication algorithm. The proposed Karatsuba-ZOT_x (with x = 7) out-performs all other tested algorithms. The result indicated that Karatsuba-ZOT_x algorithm is about 1.6 (for 32 bits numbers) to 2.9 times faster (for 8192 bits numbers) against the best existing Karatsuba-Classical algorithm.

The finding from this paper indicates that the proposed algorithm is currently the most suitable multiplication algorithm for the use in existing public-key cryptosystems.

ACKNOWLEDGEMENT

The researchers would like to thank the Universiti Sains Malaysia for supporting this research (Project grant: 1001/USM/817059).

REFERENCES

Avanzi, R. M. and Dimitrov, V. 2006. Extending Scalar Multiplication to Double Bases. In: Lai, X., Chen, K. (eds.). *ASIACRYPT 2006. LNCSS.* **4284**: 130-144.

Bernstein, D. J. 2009. *Batch Binary Edwards*. Advances in Cryptology - Crypto 2009. S. Halevi. Berlin, Springer-Verlag Berlin. **5677**: 317-336.

Bodrato, M. 2007. Towards Optimal Toom-Cook Multiplication for Univariate and Multivariate Polynomials in Characteristic 2 and 0. Proceedings of the 1st international workshop on Arithmetic of Finite Fields. Madrid, Spain, Springer-Verlag.

Cook, S. A. 1966. On the Minimum Computation Time of Functions. Mathematics, Harvard University. *Ph.D. Thesis*, Department of Mathematics.

Dimitrov, V. S. *et al*. 1997. Theory and applications for a double-base number system. *Computer Arithmetic, 1997. Proceedings of 13th IEEE Symposium.*

Fürer, M. 2007. Faster integer multiplication. Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. San Diego, California, USA, ACM.

Haining, F. and Hasan, A. 2007. Comments on Five, Six, and Seven-Term Karatsuba-Like Formulae'. *Computers, IEEE Transactions.* **56**(5): 716-717.

Hars, L. 2007. Applications of fast truncated multiplication in cryptography. *EURASIP J. Embedded Syst.* **2007**(1): 3-13.

Hasan, M. A. 2000. Look-up table-based large finite field multiplication in memory constrained cryptosystems. *Computers, IEEE Transactions.* **49**(7): 749-758.

J. Von, J. S. 2002. Fast Arithmetic for Polynomials Over F2 in Hardware. In Proc. *IEEE Information Theory Workshop.*

Jahani, S. 2009. ZOT-M_K: A New Algorithm for Big Integer Multiplication. Department of Computer Science. Penang, Universiti Sains Malaysia. **Msc. Thesis**.

Jedwab, J. and Mitchell, C. J. 1989. Minimum weight modified signed-digit representations and fast exponentiation. *Electronics Letters.* **25**(17): 1171-1172.

Karatsuba, A. and Ofman, Y. 1962. Multiplication of Many-Digital Numbers by Automatic Computers. *Proceedings of the USSR Academy of Sciences*.

Karatsuba, A. and Ofman, Y. 1963. Multiplication of Multidigit Numbers on Automata. *Soviet Physics Doklady (English translation).* **7**(7): 595-596.

Knuth, E. 1997. *The Art of Computer Programming*. Addison-Wesley.

Mahboob, A. and Ikram, N. 2005. Lookup table based multiplication technique for GF(2(m)) with cryptographic significance. *IEE Proceedings-Communications.* **152**(6): 965-974.

Maitra, S. and Sinha, A. 2007. A single digit triple base number system - a new concept for implementing high performance multiplier unit for DSP applications. *Information, Communications & Signal Processing, 2007 6th International Conference.*

Montgomery, P. L. 2005. Five, six, and seven-term Karatsuba-like formulae. *Computers, IEEE Transactions.* **54**(3): 362-369.

Okeya, K. *et al.* 2004. Signed binary representations revisited. Proceedings of .Advances in Cryptology - Crypto 2004. M. Franklin. Berlin, Springer-Verlag Berlin. **3152:** 123-139.

Reitwiesner, G. W. 1960. Binary arithmetic. *Advances in Computers.* **1:**231-308.

Sadiq, M. and J. Ahmed 2006. Complexity Analysis of Multiplication of Long Integers. *Asian Jurnal of Information Technology.* **5**(2): 6-11.

Schonhage, A. and V. Strassen 1971. Schnelle Multiplikation grober Zahlen. *Computing in Science & Engineering.* **7**:139-144.

Wen-Ching, L. et al. 2008. A new look-up table-based multiplier/squarer design for cryptosystems over GF(2^m). *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium* .

Xianjin, F. and Longshu, L. 2007. On Karatsuba Multiplication Algorithm. *Data, Privacy, and E-Commerce, 2007. ISDPE 2007. The First International Symposium* .

Zuras, D. 1994. More on Squaring and Multiplying Large Integers. *IEEE Transactions on Computers.* **43**(8): 899-908.