КРИПТОЛОГІЯ / CRYPTOLOGY

INTEGER MULTIPLICATION ALGORITHM WITH DELAYED CARRY MECHANISM FOR PUBLIC KEY CRYPTOSYSTEMS

Vladyslav Kovtun, Andrew Okhrimenko

National Aviation University, Ukraine

KOVTUN Vladyslav Yu., Candidate of Science (PhD)

Date and place of birth: 1978, Kirovograd, Ukraine.
Education: Kharkiv Military University, 2000.
Current position & Functions: Associate Professor at IT-Security Dept since 2010.
Research interests: information security, quick arithmetic transformations in Galois fields, public key cryptosystems, cryptanalysis of cryptographic transformation with public-key.
Publications: over 40 scientific publications, papers in domestic & foreign scientific journals, international conferences proceedings, patents etc.
E-mail: vladislav.kovtun@gmail.com

OKHRIMENKO Andrew O., Candidate of Science of NAU (PhD of NAU)

Date and place of birth: 1990, Vasilkiv, Kyiv region, Ukraine.
Education: National Aviation University, 2012.
Current position & Functions: Postgraduate student.
Research interests: information security, public key cryptosystems, network security, risk assessment, risk management.
Publications: over 40 scientific publications including papers, conference proceedings, international conferences materials, certificates of copyright registration etc.
E-mail: andrew.okhrimenko@gmail.com

Abstract. *Authors have offered the approach to increase performance of software implementation of integer multiplication algorithm, for 32-bit and 64-bit platforms. The approach relies on delayed carry mechanism of significant bit in sum accumulation. This strategy allows preventing necessity to consider the significant bit carry at the each iteration of the sum accumulation loop. The delayed carry mechanism enables to reduce the total number of additions and apply the modern parallelization technologies effectively.*

Key words: *multiplication of integers, software implementation, cryptographic transformation, cryptosystem, parallelism, delayed carry.*

1. Introduction

The cryptographic transformations with public key revolutionized from Diffie and Hellman consideration [1] to modern algebraic curves cryptosystems. However transformations were stayed permanent – with operations in the number field . The integer multiplication takes the special place in number field operations, see Fig. 1. Among the urgent problems of public key cryptosystems improvements is the increasing of software performance and hardware implementation. One approache to increase the performance of cryptosystems is an increasing the performance of multiplication operations in the finite field arithmetic.

The problem of increase of arithmetic operation in number fields is actively investigated by many scientists, as evidence by the significant publications of this field [2-8]. Except from the arithmetic operations algorithms, it is interesting approaches to the architecture of the software libraries [9-18] with field operations, which allows decreasing overheads on fields operations in general.

Cryptographic transformations	Encryption, decryption	Digital signature generation, verification	Key exchange
Arithmetic in finite field	Multiplication		
	Exponentiation	Addition	Substruction	Squaring	Inversion
CPU commands	mov, mul, shr, shl, add, sub ...		

Fig. 1. Operation hierarchy of elliptic curve cryptosystem

The analysis of publications [2-7], allowed to identify the most efficient multiplication algorithms Comba [2, 3] and Karatsuba [3, 8, 10]. However, the Comba algorithm shows better results in performance tests (benchmark) of software implementations on modern platforms [3-9]. In [8] described the Karatsuba-Comba multiplication (KCM) algorithm for the RISC

processors. The KCM algorithm is an interesting symbiosis of Comba and Karatsuba algorithms, where Karatsuba algorithm is specially used only for machine word multiplication.

Therefore, the main goal of this paper is approaches suggestion for the effectiveness increasing of software implementation of finite field number multiplication (squaring) via well-known Comba algorithm [2, 3, 8]. Such researches were caused by the necessity the effectiveness confirmation of software implementations of known algorithms for continuous development of modern 32-bit and 64-bit platforms. It is significant, last ten years has seen much development in the direction of the multi-core processors and multiprocessor systems [8, 9].

2. Multiplication algorithm-prototype description and its modification

The Comba algorithm [2] based on a main loops p. 2, p. 3 and nested loops p. 2.1, p.3.1. At the low level of hierarchy, in loops p. 2.1 and p. 3.1 computes 64-bit integer product $(uv)^{(64)}$ which splits on two 32-bit integer $u^{(32)}$ and $v^{(32)}$.

The sum accumulation occurs in 32-bit temporary variables r_0, r_1 and r_2, on each iterations p. 2.1.2, p. 2.1.3.

The final result assignment and temporary variables r_0, r_1 and r_2 changing, occurs on each iteration on p. 2.2.

Algorithm. Comba's integer multiplication

Input: integers $a, b \in GF(p)$, $w = 32$, $n = \log_{2^w} a$ Output: $c = a \cdot b$

1. $r_0^{(32)} \leftarrow 0$, $r_1^{(32)} \leftarrow 0$, $r_2^{(32)} \leftarrow 0$.
2. For $k \leftarrow 0$, $k < 2n-1$, $k{+}{+}$ do
2.1. For $i \leftarrow 0$, $i < n$, $i{+}{+}$ do
2.1.1. For $j \leftarrow 0$, $j < n$, $j{+}{+}$
2.1.1.1. If $(i + j == k)$
2.1.1.1.1. $(uv)^{(64)} \leftarrow a_i^{(32)} \cdot b_j^{(32)}$.
2.1.1.1.2. $r_0^{(32)} \leftarrow r_0^{(32)} + v^{(32)}$,
$r_1^{(32)} \leftarrow r_1^{(32)} + u^{(32)} + carry$,
$carry \leftarrow 0$.
2.1.1.1.3. $r_2^{(32)} \leftarrow r_2^{(32)} + carry$, $carry \leftarrow 0$.
2.2. $c_k^{(32)} \leftarrow r_0^{(32)}$, $r_0^{(32)} \leftarrow r_1^{(32)}$, $r_1^{(32)} \leftarrow r_2^{(32)}$, $r_2^{(32)} \leftarrow 0$.
3. $c_{2n-1}^{(32)} \leftarrow r_0^{(32)}$.
4. Return (c).

Consider the main drawbacks of Comba's algorithm:

- In the internal loops p. 2.1 and p. 3.1 there is a sum accumulation with carry in 32-bit temporary variables r_0, r_1 and r_2, p. 2.1.2, p. 2.1.3 and p. 3.1.2, p. 3.1.3:

2.1.2. $r_0^{(32)} \leftarrow r_0^{(32)} + v^{(32)}$, $r_1^{(32)} \leftarrow r_1^{(32)} + u^{(32)} + carry$, $carry \leftarrow 0$.

2.1.3. $r_2^{(32)} \leftarrow r_2^{(32)} + carry$, $carry \leftarrow 0$.

In this case there are 3 additions of 32-bit integer (two of them with the carry), 3 assignments 32-bit variables r_0, r_1 and r_2. The sum accumulation with carry takes place at each iteration of loop p. 2.1.

– In internal loops p. 2.1 and p. 3.1, for the sum accumulation, for 32-bit variables r_0, r_1 and r_2 the transfers are considered, using the assembler code for the implementation of addition operation with carry. That in turn doesn't allow to pair and parallelize [11], as a result we observe an ineffective processor resource using.

– Loops p. 2 and p. 3 cannot be effectively parallelized due to high internal linkage code (carry consideration).

– Using modern processors support of 64-bit operations is not considered in Comba.

It is easy to obtain a computational complexity for the Comba's algorithm:

$$I_{mul}^{Comba} = 4I_{assign}^{32} + \left(\tfrac{n+1}{2}n + \tfrac{1+n-1}{2}(n-1)\right)\times$$
$$\times\left(1I_{mul}^{32} + 3I_{add}^{32} + 6I_{assign}^{32}\right) + 4(2n-1)I_{assign}^{32} =$$
$$= 4I_{assign}^{32} + n^2\left(1I_{mul}^{32} + 3I_{add}^{32} + 6I_{assign}^{32}\right) + 4(2n-1)I_{assign}^{32}$$

where I_{assign}^{32} - an assignment operation of 32-bit integers, I_{add}^{32} - an addition operation of 32-bit integers, I_{mul}^{32} - a multiplication operation of 32-bit integer. Fig. 2 illustrates the drawbacks of algorithm for $n = 3$ and its impact on computational complexity of algorithm.

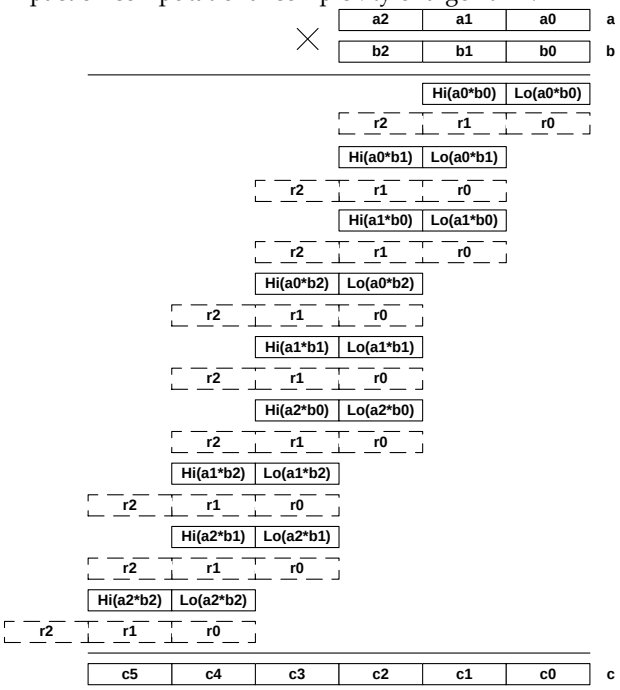

Fig. 2. Graphic interpretation of Comba algorithm

In upper part of figure there are two big numbers a and b represented by three 32-bit integers $a = (a_2, a_1, a_0)$ and $b = (b_2, b_1, b_0)$, where a_i and b_i have a machine word bit size. Algorithm iterations are presented under the solidus. It should be noted that algorithm Comba implements long multiplication technique, known from school, with small difference: the multiplier part a_i $i = \overline{1, n}$ multiply on all parts of other

multiplier b_j $j=\overline{1,n}$, in case of fulfillment the condition $(i+j==k)$ (in columns).

Such approach does not lead to rows addition (intermediate results of multiplication) as in long multiplication, but to columns addition. That allows to find a part of resulting product c_i (under the lower line). As shown in the Fig. 2, each multiplication is accompanied by the sum accumulation with a carry.

The computational complexity for $n=3$, will be:

$$I_{mul}^{Comba}=9\left(1I_{mul}^{32}+3I_{add}^{32}+6I_{assign}^{32}\right)+$$
$$+4I_{assign}^{32}+20I_{assign}^{32}=78I_{assign}^{32}+9I_{mul}^{32}+27I_{add}^{32}.$$

Now we begin considering the approaches proposed by the authors and dedicated to elimination of these drawbacks:

− The modern 32-bit processors effectively implement the addition operations of 32-bit and 64-bit integers, using 64-bit or 32-bit commands. That allows to implement a carry accumulation by addition of 32-bit variables in 64-bit variable-accumulator, that save the carry accounting and correction requirements after the addition with variables r_0, r_1 and r_2. Accumulated carry will be considered in the final iterations of the loops in p.2 and p.3.

− Modern processors have multi-core architecture; that allows them to execute several instruction flows at the same time. This property brings to parallel execute of iterations in loop p.2 and p.3 by the OpenMP library [11-13].

Following notations are to be introduced: through $t^{(64)}$ will symbolized 64-bit variables, and through $t^{(32)}$ - 32-bit variables; operation $\mathrm{hi}_{(32)}\left(t^{(64)}\right)$ extracts 32 the most significant bits in 64-bit variable, and operation $\mathrm{low}_{(32)}\left(t^{(64)}\right)$ extracts 32 the least significant bits in 64-bit variable.

Algorithm. Modified Comba's integer multiplication

Input: integers $a,b\in GF(p)$, $w=32$, $n=\log_{2^w}a$, $nk=2n-1$.

Output: $c=a\cdot b$

1. $r_0^{(64)}\leftarrow 0$, $r_1^{(64)}\leftarrow 0$, $r_2^{(64)}\leftarrow 0$.

2. For $k\leftarrow 0$, $k<n$, $k++$ do

2.1. For $i\leftarrow 0$, $j\leftarrow k$, $i\le k$, $i++$, $j--$ do

2.1.1. $(uv)^{(64)}\leftarrow a_i^{(32)}\cdot b_j^{32}$.

2.1.2. $r_0^{(64)}\leftarrow r_0^{(64)}+v^{(32)}$, $r_1^{(64)}\leftarrow r_1^{(64)}+u^{(32)}$.

2.2. $r_1^{(64)}\leftarrow r_1^{(64)}+\mathrm{hi}_{(32)}\left(r_0^{(64)}\right)$, $r_2^{(64)}\leftarrow r_2^{(64)}+\mathrm{hi}_{(32)}\left(r_1^{(64)}\right)$.

2.3. $c_k^{(32)}\leftarrow \mathrm{low}_{(32)}\left(r_0^{(32)}\right)$, $r_0^{(64)}\leftarrow \mathrm{low}_{(32)}\left(r_1^{(32)}\right)$,

$r_1^{(64)}\leftarrow \mathrm{low}_{(32)}\left(r_2^{(32)}\right)$, $r_2^{(64)}\leftarrow 0$.

3. For $k\leftarrow n$, $l\leftarrow 1, k<nk$, $k++$, $l++$ do

3.1. For $i\leftarrow l$, $j\leftarrow k-l$, $i<n$, $i++$, $j--$ do

3.1.1. $(uv)^{(64)}\leftarrow a_i^{(32)}\cdot b_j^{32}$.

3.1.2. $r_0^{(64)}\leftarrow r_0^{(64)}+v^{(32)}$, $r_1^{(64)}\leftarrow r_1^{(64)}+u^{(32)}$.

3.2. $r_1^{(64)}\leftarrow r_1^{(64)}+\mathrm{hi}_{(32)}\left(r_0^{(64)}\right)$, $r_2^{(64)}\leftarrow r_2^{(64)}+\mathrm{hi}_{(32)}\left(r_1^{(64)}\right)$.

3.3. $c_k^{(32)}\leftarrow \mathrm{low}_{(32)}\left(r_0^{(32)}\right)$, $r_0^{(64)}\leftarrow \mathrm{low}_{(32)}\left(r_1^{(32)}\right)$,

$r_1^{(64)}\leftarrow \mathrm{low}_{(32)}\left(r_2^{(32)}\right)$, $r_2^{(64)}\leftarrow 0$.

4. $c_{nk}^{(32)}\leftarrow \mathrm{low}_{(32)}\left(r_0^{(32)}\right)$.

5. Return (c).

It is not difficult to get a computational complexity of modified Comba algorithm:

$$I_{mul}^{Mod.Comba}=4I_{assign}^{64}+\left(\tfrac{n+1}{2}n+\tfrac{1+n-1}{2}(n-1)\right)*$$
$$*\left(1I_{mul}^{32}+2I_{add}^{64|32}+2I_{assign}^{64}\right)+$$
$$+(2n-1)\left(2I_{add}^{64|32}+1I_{assign}^{64}+1I_{assign}^{32}\right)=$$
$$=4I_{assign}^{64}+n^2\left(1I_{mul}^{32}+2I_{add}^{64|32}+2I_{assign}^{64}\right)+$$
$$+(2n-1)\left(2I_{add}^{64|32}+1I_{assign}^{64}+1I_{assign}^{32}\right)$$

where I_{assign}^{32} - an assignment operation of 32-bit integers, I_{assign}^{64} - an assignment operations of 64-bit integers, I_{add}^{32} - an addition operation of 32-bit integers, $I_{add}^{64|32}$ - an addition operation of 32-bit and 64-bit integers, I_{mul}^{32} - a multiplication of 32-bit integers.

Fig. 3, 4 illustrate the algorithm 2 for $n=3$; computational complexity for this case will be:

$$I_{mul}^{Mod.Comba}=27I_{assign}^{64}+9I_{mul}^{32}+28I_{add}^{64|32}+5I_{assign}^{32}.$$

3. Comparison with other algorithms

For the objective comparison of given results, the authors have made the review of well-known software math libraries [14-24] for public key cryptography. According to the review results the software library GMP was selected as an etalon [14]. The Karatsuba multiplication algorithm for the integer multiplication [2] is used in GMP. The comparison of software implementations will be done by the comparing the average time execution of software implementation of Comba, modified Comba algorithms and implemented in GMP library for one million iterations.

The proposed modified algorithm Comba (MC) and its prototype – algorithm Comba have implemented in C++, compiled with Intel C++ Compiler XE 13 in Release configuration with 32-bit and 64-bit machine word size with Maximize Speed parameter and SSE2 instruction support. The etalon library GMP v4.1.2 compiled with Microsoft Visual Studio 2010 and inserted to the test application. The test application compiled with Intel C++ Compiler XE 13 in Release configuration with Maximize Speed parameter and SSE2 instruction support. Also tested the implementation obtained using Visual Studio 2010. The results are not significantly different from those obtained by Intel C++ Compiler (± 1%), suggesting that the independence of the results on the compiler.

In testing have used desktop platform with Intel Core i5-3570 (6M Cache, 3.40 GHz) processor and Microsoft Windows 7 Ultimate x64 SP1 operating system. To measure the performance of algorithms software implementation is offered for integers ranging

in size from 128 to 8192 bit, which are recommended to be used in cryptographic application for the different security levels [25]. In table 1 there are shown the performance measurement results for software implementations of Comba, MC algorithms and GMP for one million multiplications for specified arrays with 32-bit words length. Table 2 shows the ratio of GMP to MC values and Comba to MC values for identifying the benefits of the proposed algorithm.

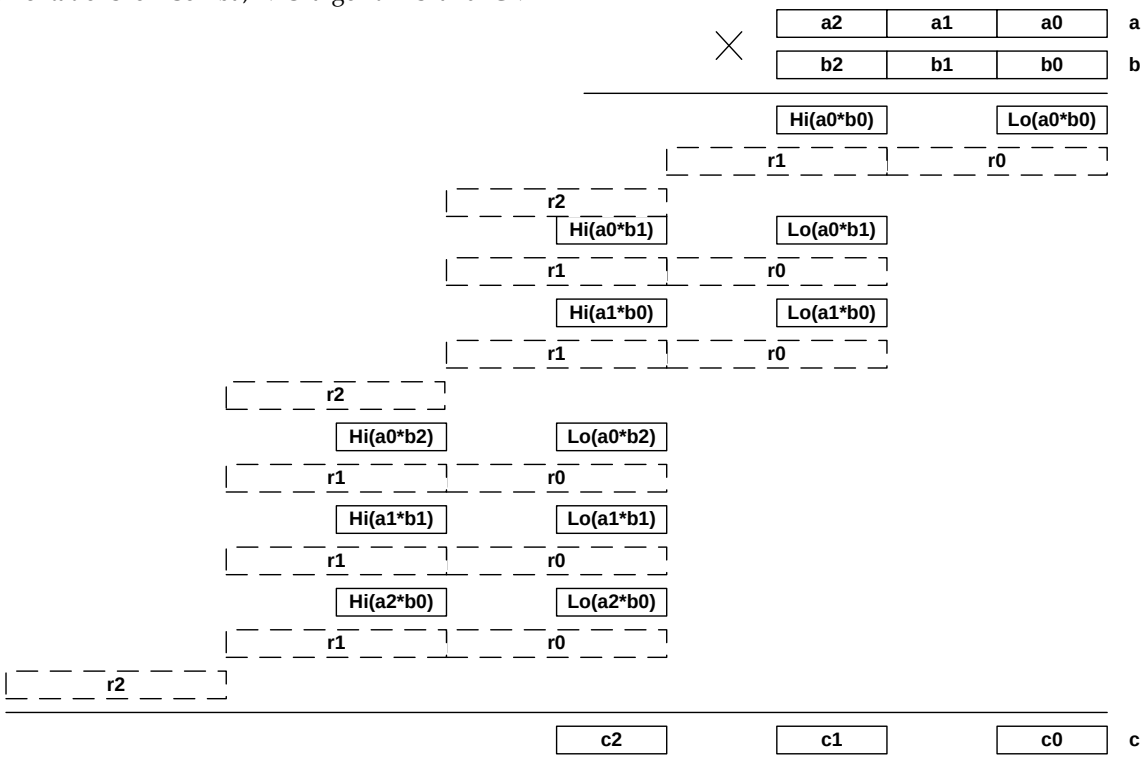

Fig. 3. Graphic interpretation of loop 2 in Modified Comba algorithm

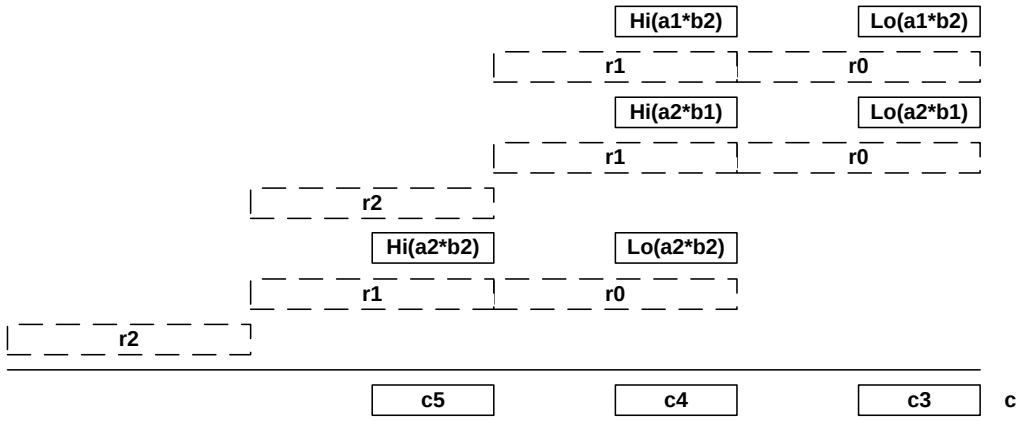

Fig. 4. Graphic interpretation of loop 3 in Modified Comba algorithm

Table 1

Test results of integer multiplication for 32 bit machine word size

SIZE, bit	Comba, ms	MC, ms	GMP, ms
128	171	78	93
256	1 451	203	312
512	11 045	671	1 186
1024	78 624	2 371	3 697
2048	478 219	8 985	11 794
3072	1 394 128	19 968	21 871
4096	3 031 194	35 069	37 050
6144	9 289 505	77 985	67 751
8192	20 840 046	137 295	117 265

Table 2

Comparison of results for integer multiplication algorithms with 32 bit machine words

SIZE, bit	Comba/MC	GMP/MC
128	2,192	1,192
256	7,148	1,537
512	16,461	1,768
1024	33,161	1,559
2048	53,224	1,313
3072	69,818	1,095
4096	86,435	1,056
6144	119,119	0,869
8192	151,790	0,854

The MC algorithm is better on cryptographically interested integers from 128 to 4096 bits, in comparison to GMP because it has lower computational complexity and experimental estimates [2, 3, 14]. On 6144-8192 bit integers GMP library with Karatsuba algorithm is better, because it has a significant lower computational complexity and experimental estimates [2, 3, 14]. Classical implementation of algorithm Comba is the slowest, that confirmed by the theoretical estimate. MC is better more than 150 times on 8192 bit integers in comparison with original Comba algorithm.

In table 3 there are shown the performance measurement results for software implementations of Comba and MC algorithms for one million multiplications for specified arrays with 64-bit words length.

Table 3

Test results of integer multiplication for 64 bit machine word size

SIZE, bit	Comba, ms	MC, ms
128	16	15
256	125	47
512	858	140
1024	6833	546
2048	52884	2137
3072	182833	4851
4096	402762	8564
6144	1239984	18970
8192	2766945	33431

Table 2 shows comparison of 32-bit and 64-bit implementation of MC and Comba for identifying the benefits of the proposed algorithm.

Table 4

Comparison of results for integer multiplication algorithms

SIZE, bit	Comba32/ Comba64	MC32/ MC64	Comba64/ MC64
128	10,688	5,200	1,067
256	11,608	4,319	2,660
512	12,873	4,793	6,129
1024	11,507	4,342	12,515
2048	9,043	4,204	24,747
3072	7,625	4,116	37,690
4096	7,526	4,095	47,030
6144	7,492	4,111	65,366
8192	7,532	4,107	82,766

The 64-bit implementation of MC algorithm has benefits about 4.5 times in comparison to the 32-bit MC implementation. The 64-bit implementation of original Comba algorithm has benefits about 9.5 times in comparison to the 32-bit MC implementation. As we can see, 64-bit MC implementation is better more than 80 times (on 8192 bit integer) in comparison to the original Comba algorithm with 64-bit machine words.

4. Conclusions

The research resulted in the following conclusions:

1. Proposed in paper delayed carry approach, allows to increase performance of software implementation of Comba integer multiplication and squaring algorithms in 150 times (32-bit implementation with 8196 integers) and surpass the performance of the popular math library GMP v4.1.2 in 1,5 times.

2. Modified multiplication Comba algorithm is preferred than Karatsuba algorithm [2] which used in GMP library, because implementation of modified Comba algorithm is faster than Karastuba [2] implementation in GMP for the modern hardware platform (32 & 64-bit).

3. Proposed in paper delayed carry approach can applied to classical box multiplication, but can not to other multiplication algorithms like Karatsuba, FFT, Fürer's, Schönhage–Strassen and etc.

4. Delayed carry mechanism allows to apply different parallelization techniques to modified Comba algorithm, for example OpenMP [28], Intel Threading Blocks [30], OpenCL [29].

Recently, the microprocessors development increases the number of instruction processing flows. Thus, it should be said about the necessity of suitable algorithms development for efficient parallelization.

Nvidia company, already proposes GPU with more than 1256 cores and suitable CUDA Toolkit [27] which allows to create a valid multithread applications. This area is already under close monitoring, what is demonstrated in work [9]. A further area of research will be focused on effective parallelization algorithms for arithmetic operations with integers.

References

[1] Diffie W., Hellman M. E., "New directions in cryptography," IEEE Transactions on Information Theory, vol. IT-22, pp. 644–654, 1976.

[2] Comba P. G. Exponentiation cryptosystems on the IBM PC // IBM Systems Journal. –Vol. 29(4). - 1990. - pp. 526–538.

[3] Brown M., Hankerson D., Lopez J., Menezes A. Software implementation of the NIST elliptic curves over prime fields // Research Report CORR 2000–55. Department of Combinatorics and Optimization, University of Waterloo. –Canada: Waterloo, Ontario, 2000. –21p.

[4] Hong S-M., Oh S-Y., Yoon H. New Modular Multiplication algorithms for fast modular exponeniation // Advances in Cryptology-Proceedings of Eurocrypt '96. –Springer-Verlag. - 1996. – pp.166-177.

[5] Avanzi R. M. Aspects of hyperelliptic curves over large prime fields in software implementations // Cryptology ePrint Archive. – Report 2003/253. – 2003. – 23p. Available from: http://eprint.iacr.org

[6] Paar C. Implementation options for finite filed arithmetic for elliptic curve cryptosystems // Worchester Polytechnic Institute. –ECC'99. – 1999. – 31p. Available from: http://www.ece.wpi.edu/research/crypto.html

[7] Gaubatz G. Versatile Montgomery multiplier architectures. Master thesis: electrical and computer engineering. – 2002. – Worcester polytechnic institute. – 101p.

[8] Johann Großschadl, Roberto M. Avanzi, Erkay Sava, Stefan Tillich. Energy-Efficient Software Implementation of Long Integer Modular

Arithmetic // Advances in Cryptology-Prociding in CHES'2005. – Springer-Verlag. - 2005. - LNCS 3659. - pp. 75-90.

[9] Giorgi P. Izard T, Tisserand A. Comparison of Modular Arithmetic Algorithms on GPUs // ParCo'09: International Conference on Parallel Computing, France. – 2009. Available from: http://hal-lirmm.ccsd.cnrs.fr/lirmm-00424288/fr/

[10] Weimerskirch A., Paar C. Generalizations of the Karatsuba Algorithm for Efficient Implementations. // Cryptology ePrint Archive. –Report 2006/224. – 2006. –17p. Available from: http://eprint.iacr.org

[11] Intel® 64 and IA-32 Architectures Optimization Reference Manual. Order Number: 248966-025. Available from: http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf

[12] The OpenMP API Specification for Parallel Programming. Available from: http://openmp.org/wp/openmp-specifications/

[13] OpenMP in Visual C++. Available at: http:// http://msdn.microsoft.com/en-us/library/tt15eb9t.aspx

[14] The GNU Multiply Precision Library (GMP). Available from: http://gmplib.org

[15] LiDIA. Available from: https://www.cdc.informatik.tu-darmstadt.de/en/cdc

[16] Multiprecision Unsigned Number Template Library (MUNTL). Available from: http://mktmk.narod.ru/eng/muntl/muntl.htm

[17] TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. Available from: http://discovery.csc.ncsu.edu/software/TinyECC

[18] Galois Field Arithmetic Library. Available from: http://www.partow.net/projects/galois/

[19] MPFQ: Fast Finite Fields Library. Available from: http://mpfq.gforge.inria.fr/

[20] BBNUM. Available from: http://www.iw-net.org/index.php?title=Bbnum_library

[21] FLINT: Fast Library for Number Theory. Available from: http://www.flintlib.org

[22] Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL). Available from: http://indigo.ie/~mscott

[23] LibTom Projects: LibTomMath, TomsFastMath. Available from: http://libtom.org

[24] Abusharekh A., Gaj K. Comparative Analysis of Software Libraries for Public Key Cryptography // Software Performance Enhancement for Encryption and Decryption, SPEED'2007. June 11-12, 2007.

[25] Giorgi P., Imbert L., Izard T. Multipartite Modular Multiplication. Preprint. Available from: http://hal.archives-ouvertes.fr/lirmm-00618437/fr/

[26] National Institute of Standards and Technology, Recommended Elliptic Curves for Federal Government Use, Appendix to FIPS 186-2, 2000. –43p.

[27] NVIDIA. NVIDIA CUDA Programming Guide 2.0. Available from: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf

[28] OpenCL - The open standard for parallel programming of heterogeneous systems. Available from: http://www.khronos.org/opencl

[29] Intel Threading Blocks. Available at: http://software.intel.com/en-us/articles/intel-tbb

UDC 004.051/056 (045)

Ковтун В.Ю, Охріменко., А.О. Алгоритм множення цілих чисел з використанням відкладеного переносу для криптосистем з відкритим ключем

Анотація. *Автори пропонують підхід до підвищення продуктивності програмної реалізації алгоритму множення цілих чисел для 32-бітних і 64-бітних платформ. Цей підхід ґрунтується на механізмі відкладеного переносу зі старшого біта при накопиченні суми. Ця стратегія дозволяє уникнути необхідності врахування переносу зі старшого розряду на кожній ітерації циклу накопичення суми. Механізм відкладеного переносу дозволяє зменшити загальну кількість операцій суми і ефективно застосовувати сучасні технології розпаралелювання.*

Ключові слова: *множення цілих чисел, програмна реалізація, криптографічні перетворення, криптосистема, розпаралелювання, відкладений перенос.*

Ковтун В.Ю., Охрименко А.А. Алгоритм умножения целых чисел с использованием отложенного переноса для криптосистем с открытым ключом

Аннотация. *Авторы предлагают подход к повышению производительности программной реализации алгоритма умножения целых чисел для 32-битных и 64-битных платформ. Этот подход основывается на механизме отложенного переноса из старшего бита при накоплении суммы. Эта стратегия позволяет избежать необходимости учета переноса из старшего разряда на каждой итерации цикла накопления суммы. Механизм отложенного переноса позволяет уменьшить общее количество операций суммирования и эффективно применять современные технологии распараллеливания.*

Ключевые слова: *умножение целых чисел, программная реализация, криптографические преобразования, криптосистема, распараллеливание, отложенный перенос.*

Отримано *30 січня 2013 року*, затверджено редколегією *15 березня 2013 року*