PERFORMANCE INCREASING APPROACHES FOR BINARY FILD INVERSION

Vladislav Kovtun
Chair of Information Security
National Aviation University
Kiev, Ukraine
vladislav.kovtun@nrjetix.com

Maria Bulakh
Chair of Information Security
National Aviation University
Kiev, Ukraine
bulakh.masha@gmail.com

Authors propose several approaches for increasing performance of multiplicative inversion algorithm in binary fields based on Extended Euclidean Algorithm (EEA). First approach is based on Extended Euclidean Algorithm specificity: either invariant polynomial v remains intact or swaps with invariant polynomial u. It makes it possible to avoid necessity of polynomial v degree computing. The second approach is based on searching the "next matching index" when calculating the degree of the polynomial, since degree polynomial invariant u at least decreases by 1, then it is possible to use current value while further calculation the degree of the polynomial.

When based on the second approach, only significant terms are used in computation during modification of invariants, taking into account current degree of polynomial pairs (u,v) and (b,c) as authors proposed. These approaches can increase performance of software implementation of inversion for 32-bit platforms by 15-20%.

Keywords: multiplicative inversion, Extended Euclidean Algorithm, binary field, polynomial.

Introduction. The wide usage of information and telecommunication systems in modern society requires protection of information that circulates, is created, modified, stored and destroyed in these systems. For this purpose, information security subsystem is created in each such system. The core of information security subsystem is cryptographic subsystem. Among such cryptosystems, public key cryptosystems (directional encryption, secret sharing and digital signature) takes a special place.

Among the widely used public key cryptosystems, can be emphasized: cryptosystems on elliptic (EC) and hyperelliptic curves (HEC), in the fields and rings. While implementation of these cryptosystems operations over field $\mathbf{GF}(p)$ and $\mathbf{GF}(2^m)$ elements are widely used. It is known that fields $\mathbf{GF}(p)$ and $\mathbf{GF}(2^m)$ are used in EC cryptosystems (ECC) in the international standards IEEE P1363-2000 [1], ISO/IEC 15946-2 [2] and in national standards DSTU 4145-2002 [3], ISO/IEC 15946-2 [2].

According to [1-4], ECC can be represented as a hierarchy of operations with a particular significance of operations in field, see Fig. 1.

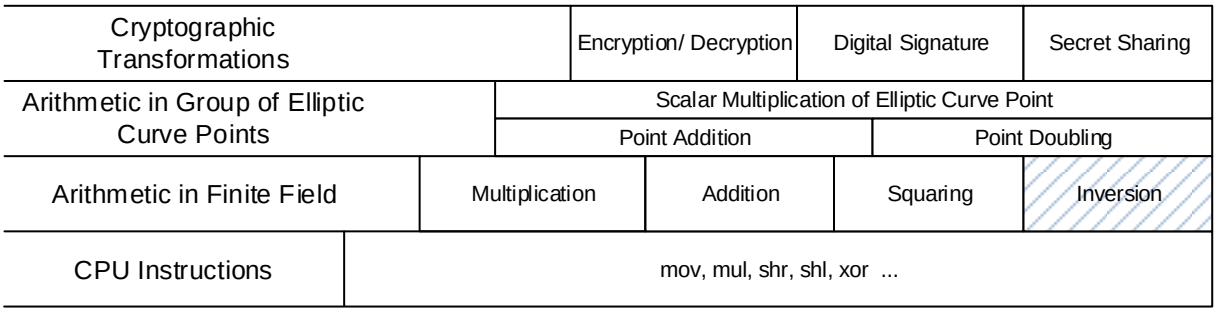

Fig. 1. Hierarchy of operations in ECC

It is known [6-11] that the performance of symmetric cryptosystems substantially exceeds public key cryptosystems' performance, which makes it actual to increase public key cryptosystems' performance research. Among the areas for further improvements, the authors emphasized [6-11]:

- Increase performance of group operations (EC points, the divisor in the Jacobian HEC, etc.).
- Increase performance of operations over data structures used for group elements representation.
- Increase performance of field operations.
- Increase performance of operations over data structures used for field elements representation.
- Optimization of operations over data structures for modern superscalar CPU [5].

At present, there are many publications in this field [6-11], that examines the algorithms of operations over polynomials, even in the context of libraries' software architecture [6-11]. It can significantly reduce the overhead cost of operations over polynomials implementation in general.

There can be distinguished several operations over $\mathbf{GF}(2^m)$ field elements [4]: addition, multiplication, squaring, multiplicative inversion and exponentiation. Researches show [4, 7] that execution time of inversion operation affects on ECC performance, in general. In order to reduce the inversion operation affect on performance [4] it is proposed to use projective representation of EC points. Similar arguments are applicable for divisors of HEC Jacobian. However, it is not possible to exclude inversion from operation over polynomials in public key cryptosystems implementation.

Consequently, authors pay attention on performance increasing of inversion operation in binary field $\mathbf{GF}(2^m)$.

Description of algorithm-prototype of the Extended Euclidean Algorithm and its modification. Algorithm of multiplicative inversion allows inverse of non-zero field element $a \in GF(2^m)$ computing, using Extended Euclidean Algorithm (EEA) for polynomials [4]. The algorithm [4] based on invariants $ba + df = u$ and $ca + ef = v$ for some d and e which are implicitly computed. At each iteration, if $\deg(u) \geq \deg(v)$, then a partial distribution of u via v is performed by subtracting $x^j v$ from u, where $j = \deg(u) - \deg(v)$. Hence, the degree of u is remained constant or decreased at least by 1, or on average by 2. Adding $x^j c$ to b preserves the invariant. The algorithm terminates when $\deg(u) = 0$. In this case $u = 1$ and $ba + df = 1$; hence $b = a^{-1} \bmod f(x)$.

Algorithm 1. Extended Euclidean Algorithm for multiplicative inverse in $\mathbf{GF}(2^m)$.

Input: $a \in \mathbf{GF}(2^m)$, $a \neq 0$.

Output: $a^{-1} \bmod f(x)$.

1. $b \leftarrow 1$, $c \leftarrow 0$, $u \leftarrow a$, $v \leftarrow f$.

2. While $\deg(u) \neq 0$ do

2.1. $j \leftarrow \deg(u) - \deg(v)$.

2.2. if $j < 0$ then: $u \leftrightarrow v$, $b \leftrightarrow c$, $j \leftarrow -j$.

2.3. $u \leftarrow u + x^j v$, $b \leftarrow b + x^j c$.

3. Return (b).

Analysis of Algorithm 1, allows selecting several aspects for further improvement of the EEA:

At the steps 2.2 and 2.3, polynomial u is modified, while polynomial v contains the previous value of the polynomial u. This allows refusing from polynomial v degree computing in step 2.1. As for $\deg(v)$ initial calculating at step 2.1, the degree is known as constant $\deg(v) = \deg(f) = m$.

At step 2.1, degree of the polynomial u is calculated. According to general logic of algorithm, the degree of the polynomial u is permanently decreasing, at least by 1. This allows dismissing the $\deg(u)$ degree computing on each iteration in a general case, and only extends current.

At step 2.3, polynomial v shift and addition with u are performed, but it should be noted, that degree of polynomials v and u is permanently decreasing while step 2 loop execution while the degree of polynomials b and c is growing. This allows to shift and addition not for all elements of field elements' representing array, but for significant only - which are certainly non-zero.

Let's estimate complexity of Algorithm 1, on average case. Lets polynomial a degree $\deg(a) = k$ and Hamming weight of polynomial a $h = \text{weight}(a) = k/2$. Then count of loop iterations will be k, and number of truthful statements $j \neq 0$ will be $2k/3$. In rest $k/3$ cases, the shift is not executed. The complexity of Algorithm 1 on average case is:

$$I_{avr}(A_1) = k(2I_{\deg} + 2I_{add} + 2I_{shl}) + 2k/3(2I_{swp}). \quad (1)$$

where $I_{\deg}$ - complexity of algorithm for computing degree of polynomial; I_{add} - complexity of algorithm for adding two polynomials, I_{shl} - complexity of algorithm for shifting to an arbitrary number of bits (may to exceed the length of machine word); I_{swp} - complexity of two polynomials exchange algorithm (further, I_{swp} will be neglected, because pointers will be used).

In simplified form, formulae (1) can be presented:

$$I_{avr}(A_1) = k(2I_{\deg} + 2I_{add} + 2I_{shl}) \quad (2)$$

It is easy to see from (2) that a reducing complexity of algorithm in general, can be achieved by reducing the number of operations and $I_{\deg}$, I_{add} and I_{shl} complexity.

Algorithm for degree computing of an arbitrary polynomial a requires detailed study. Polynomial basis is used for representation of $\mathbf{GF}(2^m)$ field elements.

Element $b \in \mathbf{GF}(2^m)$ in polynomial basis is represented as binary vector $b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1 x^1 + x_0$, that can be represented via array of machine words with binary length w as $a_{n-1}^{(w)} 2^{(n-1)w} + a_{n-2}^{(w)} 2^{(n-2)w} + \cdots + a_1^{(w)} 2^w + a_0^{(w)}$, where $n = \lceil \frac{m}{w} \rceil$ is number of machine words required for polynomial of binary length m representation; b_i - binary coefficients, $a_j^{(w)}$ - computer words of binary length w.

The main idea of algorithm for polynomial degree computing is to find the index of the most significant array element $a_j^{(w)}$ (MSE), and find the index of most significant bit (MSB) in already found array elements $a_j^{(w)}$. Search of MSE element $a_j^{(w)}$ is sequential search in array starting at the end, as long as desired is found. There are many well-known algorithms [12] to determine the index of MSB in array element (machine word) that require sequential search.

In order to increase performance computing MSB in machine word, authors proposed to use a variety of well-known "tricks" [12], based on bit operations over machine words. Developments proposed by authors are presented in Algorithm 2.

Carry it out a more detailed consideration. At step 2, MSE (containing at least one non-zero bit) is searched in cycle. Search is performed from the end of array (highest machine word) to its beginning.

At step 3, MSE and its index are fixed. Element index will be used in further polynomial degree calculation.

Algorithm 2. Algorithm for polynomial degree calculation in $\mathbf{GF}(2^m)$.

Input: $a \in \mathbf{GF}(2^m)$; $n = \lceil \frac{m}{w} \rceil$, where n - number of machine words, that occupies a polynomial; w - width of machine word, usually $w = 32$.

Output: $\deg(a)$.

1. $i \leftarrow n-1$.

2. While $(a_i^{(32)} \neq 0) \&\& (i > 0)$

2.1. $i \leftarrow i-1$.

3. $t \leftarrow a_i^{(32)}$.

4. $t \leftarrow t \,|\, (t >> 1)$, $t \leftarrow t \,|\, (t >> 2)$, $t \leftarrow t \,|\, (t >> 4)$, $t \leftarrow t \,|\, (t >> 8)$, $t \leftarrow t \,|\, (t >> 16)$.

5. $t \leftarrow t - ((t >> 1) \,\&\, 0\text{x}55555555)$, $t \leftarrow (t \,\&\, 0\text{x}33333333) + ((t >> 2) \,\&\, 0\text{x}33333333)$,
$t \leftarrow ((t + (t >> 4)) \,\&\, 0\text{xf0f0f0f}) \cdot 0\text{x}1010101) >> 24$.

6. Return $((i << 5) + t - 1)$.

At step 4, "trick" [12] are applied for creating a "mask", which allows filling all on-bits least significant bits from the most significant. Hereafter, counting a number of on-bits in a machine word is required, to determine the index of MSB.

In step 5, it is applied "trick" [12], for counting of all on-bits in machine word. Given number is greater than MSB on 1.

Polynomial degree is calculated at step 6, where the total number of bits in each machine word, index of MSE and index of MSB in MSE are taken into account.

Previously described results of Algorithm 1 analysis allows to make following modifications:

- At each iteration of loop in step 2, polynomial v is constant, or it can be swapped with u, so it is not necessary to calculate polynomial v degree for each loop iteration: degree of polynomial v is either constant or equals polynomial u degree. Thus, it could be saved one operation of computing polynomial degree, at each iteration in step 2 loop. Number of loop iterations at step 2 is comparable to degree of polynomial $a \in \mathbf{GF}(2^m)$, which should be inverted.
- At step 2.1, polynomial u degree is computed. At step 2.3, its degree is decreased at least by 1. It means that it is not necessary to compute polynomial degree at each loop iteration, but only adjust: polynomial u degree reduces at least by 1. That makes it possible to use "the next fit" approach to fix the position of current array element. This approach allows to start search not from the ending array element, but from array element found at previous loop iteration. The number of checks while searching MSE, at step 2 cycle is decreased in 2 times.

At step 2.3, polynomial v shift and polynomial u addition operations are performed. Degree of v and u decreases continuously throughout the step 2 loop, and degree of polynomials b and c is growing. Here it could be used "next fit" approach, to operate only with the significant elements of array. This allows shifting and adding not all elements of array, which represents field elements, but only significant. Note, number of additions and shifts of array elements at step 2 loop decreased almost in 2 times.

Applies results of analysis conducted by authors, In Algorithm 1, and algorithm itself to compute the degree of polynomial Algorithm 2, Modified EEA (MEEA) Algorithm for multiplicative inversion in the field $\mathbf{GF}(2^m)$ was proposed. It is presented as Algorithm 3.

Algorithm 3. Modified Extended Euclidean Algorithm for multiplicative inverse in $\mathbf{GF}(2^m)$.

Input: $a \in \mathbf{GF}(2^m)$, $a \neq 0$, $n = \left\lceil \frac{m}{w} \right\rceil$, where n is number of machine words that occupies a polynomial; w is width of machine word, usually $w = 32$.

Output: $a^{-1} \bmod f(x)$.

1. $b \leftarrow 1$, $c \leftarrow 0$, $u \leftarrow a$, $v \leftarrow f$.

2. $i \leftarrow n-1$.

3. While $\left(u_i^{(32)} \neq 0\right) \&\& (i > 0)$

3.1. $i \leftarrow i-1$.

4. $t \leftarrow u_i^{(32)}$.

5. $t \leftarrow t \mid (t >> 1)$, $t \leftarrow t \mid (t >> 2)$, $t \leftarrow t \mid (t >> 4)$, $t \leftarrow t \mid (t >> 8)$, $t \leftarrow t \mid (t >> 16)$.

6. $t \leftarrow t - ((t >> 1) \& 0x55555555)$, $t \leftarrow (t \& 0x33333333) + ((t >> 2) \& 0x33333333)$,
$t \leftarrow ((t + (t >> 4) \& 0xf0f0f0f) \cdot 0x1010101) >> 24$.

7. $\deg U = ((i << 5) + t - 1)$.

8. $\deg V = m$.

9. While $(\deg U > 0)$ do

9.1. if $(\deg U < \deg V)$ then $k \leftarrow \deg V - \deg U$, $u \leftrightarrow v$. $b \leftrightarrow c$.

9.2. else $k \leftarrow \deg U - \deg V$.

9.3. if $(k > 0)$ then $u = u + x^k \cdot v$, $b = b + x^k \cdot c$.

9.4. else $u = u + v$, $b = b + c$.

9.5. if $(\deg U < \deg V)$ then $\deg V = \deg U$.

9.6. While $(u_i^{(32)} \neq 0) \& \& (i > 0)$

9.6.1. $i \leftarrow i - 1$.

9.7. $t \leftarrow u_i^{(32)}$.

9.8. $t \leftarrow t \mid (t >> 1)$, $t \leftarrow t \mid (t >> 2)$, $t \leftarrow t \mid (t >> 4)$, $t \leftarrow t \mid (t >> 8)$, $t \leftarrow t \mid (t >> 16)$.

9.9. $t \leftarrow t - ((t >> 1) \& 0x55555555)$, $t \leftarrow (t \& 0x33333333) + ((t >> 2) \& 0x33333333)$,
$t \leftarrow ((t + (t >> 4) \& 0xf0f0f0f) \cdot 0x1010101) >> 24$.

9.10. $\deg U = ((i << 5) + t - 1)$.

10. Return (b).

At step 1, initialization of polynomials b, c, u and v, that will be modified during algorithm work, is performed. Then at step 2, an array MSE $u_i^{(32)}$ index i of ending, non-zero of polynomial u machine word is determined. An index i search starts from MSE.

Desired array element $u_i^{(32)}$ uses at steps 4-6 for calculation of MSB index t in word $u_i^{(32)}$, and at step 7, for polynomial u degree calculation. The initial degree of polynomial v is already known and equals m. The "trick" [12] is used for MSB selecting. All LSB sets to 1. For on-bits counting in word another "trick" [12] is used.

The loop at step 9 is basal for whole algorithm and it is performed while polynomial u degree not equals 0. The average count of iterations is $\deg(a) = k$. At step 9.1 of loop, it degree $\deg U$ and $\deg V$ checks of the polynomials u and v, respectively are performed. In case $(\deg U < \deg V)$, it is necessary to swap contents of polynomials u, v and b, c respectively. At the average, number of events $(\deg U < \deg V)$ is $k/3$. The diminution $(\deg U - \deg V)$ is used for polynomials v and c shifts. At steps 9.3 and 9.4, MSB of polynomial u is cleared and the corresponding bits of polynomial b are set. Note, steps 9.3 and 9.4 operate only with significant array elements, i.e.

elements that contain bits that are lower then $\deg(b)$ and $\deg(v)$. The polynomial b degree is increasing, the degree of u is decreasing, where $\deg(b)+\deg(u)=m$.

Polynomial v degree calculation is performed at step 9.5, and degree of v changes if condition $(\deg U<\deg V)$ of exchange u and v polynomials is true. Next, at steps 9.6 and 9.7 MSE $u_i^{(32)}$ index i of u is determined. Desired array element $u_i^{(32)}$ is used on steps 9.8-9.9 for calculation of MSB index t in word $u_i^{(32)}$ and at step 9.10 while the degree $\deg U$ of polynomial u calculation.

Upon completion of loop, the algorithm returns polynomial $b=a^{-1} \bmod f(x)$.

Lets estimate complexity of Algorithm 3, on average. Lets degree $\deg(a)=k$ of polynomial a and Hamming weight $h=\text{weight}(a)=k/2$. Then count of loop iterations will be k, and number of truth statements $j\neq 0$ will be $2k/3$. Complexity of Algorithm 3, on average case is:

$$I_{avr}(A_3)=k\left(I_{\deg}+\tfrac{1}{2}(2I_{add}+2I_{shl})\right)+2k/3(2I_{swp}), \tag{3}$$

Further, complexity I_{swp} can be neglected. In simplified form, formulae (3) can be presented:

$$I_{avr}(A_3)=k(I_{\deg}+I_{add}+I_{shl}). \tag{4}$$

Software implementation notes. It is necessary to implement in software well-known EEA and proposed MEEA for confirmation of theoretical estimations of MEEA efficiency in comparison with well-known algorithms. Software implementations of MEEA and well-known EEA should be tested on polynomials of different length [13] and average time estimations (timings) should be compared.

In software implementation, authors use polynomial presentation of $\mathbf{GF}(2^m)$ elements. Polynomial represents as array of fixed length for given m.

While writing program, tailored the superscalar architecture of 32-bit CPU and the possibilities of modern compilers on branch prediction, parallel commands execution, loop unrolling, etc., in accordance with [5].

Software implementation was performed on high-level language C++ in Microsoft Visual Studio, in Release configuration with a compiler Microsoft Visual C++ 2010 (/O3, with SSE2 support) and Intel C++ Compiler XE2013 (/O3, with SSE4.2 support) on 32-bit platforms. For convenience, acronym of MCC and ICC, will be used respectively.

Comparison of well-known and proposed algorithms. In accordance of convenience mentioned in previous section, authors in software implemented both Algorithms 1 and 3 for estimation efficiency of proposed approaches for modification of multiplicative inversion algorithm in $\mathbf{GF}(2^m)$ field. In software implementation averaged on runtime of 1 million inversion operations, and timings presented in Table 1. Experiments have taken into account that the degree of inverted polynomial, can affect on number of main loop iterations, so polynomials of degree near to maximum were used for fields that are used in cryptosystems DSTU 4145-2002 [3] and FIPS-186- 3 [13].

Experiments were performed for various degrees of binary fields extension [3, 13], on wide used mobile CPU Intel Core i3 M350 and desktop CPU 3th generation Intel Core i5-3570 and 4th generation Intel Core i5-4670 running on Windows 7 SP 1 x86-64.

Software implementations timing of inversion operation for different fields are presented in Table 1.

Timing for software implementations of inversion operation for different algorithms Table 1

m	**Time, us**											
	Intel Core i3-350M				**Intel Core i5-3570**				**Intel Core i5-4670**			
	ICC XE2013		**MCC2010**		**ICC XE2013**		**MCC2010**		**ICC XE2013**		**MCC2010**	
	Inv	**Inv***	**Inv**	**Inv***	**Inv**	**Inv***	**Inv**	**Inv***	**Inv**	**Inv***	**Inv**	**Inv***
89	6,71	5,44	6,27	5,10	2,53	1,95	2,76	1,84	2,53	1,95	2,37	1,84
163	16,08	11,34	14,38	11,96	6,85	4,05	6,33	4,65	6,85	3,95	6,13	4,05
191	18,17	14,52	17,38	14,71	7,73	5,46	7,85	5,48	7,73	5,26	7,65	5,48
233	27,13	22,12	24,57	19,39	11,80	7,04	11,59	7,65	11,80	7,02	11,02	6,90
257	31,56	25,18	27,93	24,54	13,21	7,99	13,33	8,56	12,17	7,81	12,33	7,60
307	37,72	30,45	34,24	26,11	17,98	11,11	17,64	12,41	17,68	9,58	17,54	11,41
367	53,18	42,17	46,05	33,81	23,35	14,78	21,84	16,63	22,35	13,18	21,64	14,63
409	61,31	40,18	54,48	47,76	26,41	16,97	26,81	18,51	25,97	14,93	26,41	17,51
431	67,75	54,24	59,10	44,03	28,99	17,86	29,29	19,25	28,27	17,00	28,99	18,25
571	103,44	64,86	94,42	70,26	46,46	25,47	44,87	26,98	43,39	24,64	44,67	26,83

* - modified algorithm with proposed optimization approaches.

Experimental results show that the degree of polynomial does not affect on time using EEA, while MEEA shows a linear decrease of time with degree decreasing. Experimental results for the field elements $\mathbf{GF}\left(2^m\right)$ with less work degrees are given.

CPU Intel Core i3-350M, MEEA shows gain in 18-21% (ICC) and 16-18% (MCC). At the same time, implementation of MCC is better on 2.2% then ICC. CPU Intel Core i5-3570, MEEA shows gain in 19-22% (ICC) and 17-22% (MCC). At the same time, implementation of ICC is better on 4.9% than MCC. CPU Intel Core i5-4670, implementation of MEEA shows gain in 18-25% (ICC) and 14-22% (MCC). At the same time, implementation of ICC is better on 14.7% than MCC.

The resulting timings are fully consistent with theoretical estimations of computational complexity in (2) and (4). Software implementation of proposed MEEA shows better performance by 20-50% (for mobile CPU Intel Core i3-350M with a lower clock frequency and desktop CPU Intel Pentium 4 530J with a twice-lower clock speed) than in [7], for corresponding fields.

MEEA software appliance for digital signature generation and verification in accordance with DSTU 4145-2002 [3] (with "left-to-right" scalar multiplication algorithm) [4]:

- in affine points representation this has made it possible to increase performance by 16-20%, with field size rising;
- in Lopez-Dahab projective points representation [4] this has made it possible to increase performance by 2-4%, with field size rising.

These comparisons are applicable to results obtained with the Intel C++ Compiler XE2013 and Visual C++ 2010 for architecture x86.

Conclusion. On given results of research, there are following conclusions:

1. Optimization techniques proposed of authors for multiplicative inversion algorithm in $\mathbf{GF}\left(2^m\right)$ field, have reduced computational complexity MEEA in two times. It is confirmed by the experimental estimations.

2. MEEA software implementation has higher performance by 15-20% than algorithm-prototype, on average.

3. Applying of modified inversion algorithm for digital signature generation and verification based on DSTU 4145-2002, has increased performance by 16-20% in affine point representation and by 2-4% in Lopez-Dahab projective representation, respectively.

4. Proposed implementation of MEEA inversion algorithm have not adjusted for multi-threaded execution. It does not allow realizing full potential of modern multicore CPU.

5. Modern CPU developments is focused on increasing of number of commands execution threads. This requires development of valuable algorithms for efficient software implementation on perspective CPU. At this rate, NVIDIA company already offers GPU with more than 512 cores, and user-friendly framework CUDA [14-18] for creation multithreading applications. Further direction of researches will be focused on study and effective parallelizing of arithmetic algorithms in $\mathbf{GF}\left(2^m\right)$ field.

Literature

1. IEEE Std 1363-2000: Standard Specifications For Public Key Cryptography. Institute of Electrical and Electronics Engineers, 2000. –228 p. UR: http://grouper.ieee.org/groups/1363/.
2. ISO/IEC 15946-2:2002. Information technology - Security techniques - Cryptographic techniques based on elliptic curves -Part 2: Digital signatures. –36 p.
3. DSTU 4145-2002, Information Technologies. Cryptographic Information Protection. Digital Signature based on Elliptic Curves. Creation and Verification, Dec. 28, 2002, Derzhspozhivstandart Ukrainy, Kyiv (2003) . –39 p.
4. D. Hankerson, J. Hernandez, A. Menezes. Software implementation of Elliptic Curve Cryptography over binary fields. Proceedings of Workshop on Cryptographic Hardware and Embedded System. –LNCS 1965. –2000. –pp. 1-24.
5. Intel® 64 and IA-32 Architectures Optimization Reference Manual. Order Number: 248966-025. –660 p. URL: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html
6. Giorgi P. Izard T, Tisserand A. Comparison of Modular Arithmetic Algorithms on GPUs // ParCo'09: International Conference on Parallel Computing, France. –2009. –8 p. URL: http://hal-lirmm.ccsd.cnrs.fr/lirmm-00424288/fr/
7. Zhijie Jerry Shi and Hai Yan. Software Implementations of Elliptic Curve Cryptography // International Journal of Network Security. –Vol.7. –No.2. –2008. –pp. 57–166.
8. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. URL: http://discovery.csc.ncsu.edu/software/TinyECC/
9. Galois Field Arithmetic Library. URL: http://www.partow.net/projects/galois/
10. MPFQ: Fast Finite Fields Library. URL: http://mpfq.gforge.inria.fr/
11. LibTom Projects: LibTomPoly. URL: http://libtom.org
12. Sean Eron Anderson. Bit Twiddling Hacks. URL: http://graphics.stanford.edu/~seander/bithacks.html
13. National Institute of Standards and Technology, Recommended Elliptic Curves for Federal Government Use, Appendix to FIPS 186-2, 2000. –43 p.

14. NVIDIA. NVIDIA CUDA Programming Guide 5.0. –2013. –214 p. URL: http://docs.nvidia.com/cuda/cuda-c-programming-guide/
15. M. Bluhm, S. Gueron. Fast Software Implementation of Binary Elliptic Curve Cryptography // Cryptology ePrint Archive. –Report 2013/741. –2013. –19 p. URL: http://eprint.iacr.org/2013/741.pdf
16. E. M. Mahé, J.-M. Chauvet. Fast GPGPU-Based Elliptic Curve Scalar Multiplication // Cryptology ePrint Archive. –Report 2014/198. –2014. –9 p. URL: http://eprint.iacr.org/2014/198.pdf
17. Aaron E. Cohen, Keshab K. Parhi. GPU accelerated elliptic curve cryptography in GF(2^m) // 53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2010. –pp. 57–60.
18. Tony Cheneau, Aymen Boudguiga, Maryline Laurent. Significantly Improved Performances Of The Cryptographically Generated Addresses Thanks To ECC And GPGPU // Computers and Security Journal (CoSe), Elsevier. –Vol. 29. –2010. –pp. 419-431.