XIX IMEKO World Congress
Fundamental and Applied Metrology
September 6−11, 2009, Lisbon, Portugal

LARGE NUMBER LIBRARY – THE NEW LABVIEW TOOL FOR SECURE MEASUREMENT SYSTEMS

Piotr Bobiński [1], *Wiesław Winiecki* [2]

[1] Warsaw University of Technology, Warsaw, Poland, P.Bobinski@ire.pw.edu.pl
[2] Warsaw University of Technology, Warsaw, Poland, W.Winiecki@ire.pw.edu.pl

Abstract − The paper presents the proposal of a new mathematical tool for LabVIEW environment – the Large Number Library. After the short introduction describing the security issue in the Distributed Measurement-Control Systems (DMCS), the most popular asymmetric encryption algorithms (public-key encryption algorithms) are described in order to demonstrate the need for developing such a library. The LN library was implemented in two variants: the native LabVIEW code (G language) and using external software modules (DLLs). The algorithms implemented in both alternatives have been developed basing on the Number Theory Library (NTL) – Victor Shoup's open source library. The results of accuracy and efficiency tests for several functions implemented in both variants are shown. The developed LN library will allow for the creation of the advanced cryptographic libraries dedicated to the LabVIEW environment, enabling the development of secure communication channels in DMCS and information security of DMCS networks.

Keywords: Large Numbers, LabVIEW, public-key encryption systems.

1. INTRODUCTION

The modern applied metrology is integrally linked with other fast-growing domains, such as computer technology, data processing and telecommunications. Adaptation of the information systems' techniques for the needs of measurement systems created a new interdisciplinary field dealing with Distributed Measurement-Control Systems (DMCS). Elements of DMCS (nodes) are distributed territorially, connected via wired or wireless network and able to exchange information between each other. Currently the research in the area of DMCS is focused on the applicability issues and adapting of new information and communication technologies for such systems [8].

Nowadays, a very important issue in DMCS technology ensures the safety of communication. In many cases, the success of often costly experiments or missions and also the biological and economical security depends on the proper functioning of such systems. Due to the still growing integration with telecommunications and general public computer networks, the security of distributed measurement systems has been dramatically reduced. In many DMCS the information security of the network becomes one of the major development problems. Information security issues are also very important (because of the specific openness of such systems, and ease of attack) in the wireless and mobile DMCS systems. Since the nodes of DMCS can be both so-called Measuring Servers, usually based on PCs with huge processor power, as well as mobile wireless sensors powered from battery, the existing disparity of calculation power makes another important issue for the development of common methods, ensuring the safety of DMCS. Therefore there is an urgent need to develop proper methods and tools to ensure the safety and security of these systems [1, 8].

The software plays nowadays a huge role in measurement systems and very often determines their quality. The growing processors' computing power and memory capacity allows for the development of more complex software. An important issue becomes the creation of new methods and software tools for designing distributed measurement systems, and in particular low-cost and easy-to-use libraries and tools for designing software that provides secure exchange of information independently of used information and communication infrastructure.

Existing software design tools dedicated for DMCS, integrated software environments such as LabVIEW, LabWindows/CVI, HPVEE, enable simple and flexible development process of applications, but among others do not include libraries for secure data exchange. The security problem was only slightly considered which has resulted in the introduction of certain access control mechanisms to certain parts of an application (front panels and their components) based on login and password identification system. But there is no use of cryptographic methods, and the information between nodes is sent explicitly, mostly as a plain text.

Therefore, it seems necessary to develop a complete library of functions, programs and tools tailored to specific programming environments, which would give the application or system developer the opportunity to design and simulate secure and safe distributed measurement system in an easy and intuitive way. These additives should help to ensure safe transmission of data in any communication infrastructure and the creation of mechanisms for authentication and integrity of both measurement and control data.

In the previous work, the authors have analyzed the LabVIEW environment capabilities for efficient implementation of cryptographic algorithms [1]. The next

phase of the work, described in this paper, is to develop new mathematical tool for LabVIEW environment - a Large Number library (also known as Big Integer or arbitrary length integer library). This library allows for the computation on numbers with arbitrary (within the limits of available memory) number of decimal digits, far exceeding the typical representation in computer systems (32 or 64 bit). Large numbers are widely used in many popular cryptographic algorithms, including RSA, Rabin or ElGamal public-key encryption systems, used for both, data encryption and the generation of secure digital signatures [2, 3]. The LN library in addition to basic arithmetic operation includes operation modulo N in the suitable rings or finite bodies, functions for calculating the opposite element in such algebras and primality test algorithms.

This paper presents a proposal of the implementation of Large Number library for LabVIEW environment. The main aim of this paper is to show different ways of implementing cryptographic algorithms in the LabVIEW environment, and to give tools that will be helpful for further work, that is for the implementation of specific algorithms. The paper also provides basic theoretical knowledge about public-key encryption systems, which are claimed to be useful in creating safe and secure DMCS. Such a short tutorial should be helpful for better understanding of the topic. The paper is organized as follows. In the section 2 we present the most popular asymmetric encryption algorithms (public-key encryption algorithms) and show the need for developing library which can deal with large numbers. Next, we shortly brief the Number Theory Library, developed by Victor Shoup [7,10], which will be used as a reference software (in section 3) and the Crypto-G library for LabVIEW [9], which is not sufficient for practical use, due to the lack of asymmetric encryption systems (in section 4). In section 5 we present the implementation issues of our LN library in two variants: first in the native LabVIEW graphical language, second based on the external modules implemented in C++. Then, in section 6, we show the results of accuracy and efficiency tests for both alternatives. In the last section we conclude our work and give the recommendations for further work – implementation of cryptographic tools library for LabVIEW environment.

2. PUBLIC-KEY ENCRYPTION SYSTEMS

Public key cryptography is a method of encrypting messages using a nonsecret (public) key. The term public key cryptography also includes various others cryptographic methods using a public key, such as authentication, digital signature schemes, and key agreement [2, 3].

Preliminary analysis of the distributed measurement system for introducing elements of cryptography concludes that one of the main problems may be the disproportion in power consumption between different types of system nodes, such as servers and mobile measurement sensors. Therefore in this section we focus first on the asymmetric cryptographic systems, also known as public key encryption systems [1, 8].

In public-key encryption systems, each entity A has a public key e and a corresponding private key d. In secure systems, the task of computing d given e is computationally infeasible. The public key defines an encryption transformation E_e, while the private key defines the associated decryption transformation D_d. Any entity B wishing to send a message m to A obtains an authentic copy of A's public key e, uses the encryption transformation to obtain the ciphertext $c = E_e(m)$, and transmits c to A. To decrypt c, A applies the decryption transformation to obtain the original message $m = \underline{D_d}(c)$ [4].

The public key need not be kept secret, and, in fact, may be widely available – only its authenticity is required to guarantee that A is indeed the only party who knows the corresponding private key. A primary advantage of such systems is that providing authentic public keys is generally easier than distributing secret keys securely, as required in symmetric key systems.

Public-key encryption schemes are typically substantially slower than symmetric-key encryption algorithms such as DES. For this reason, public-key encryption is most commonly used in practice for the transport of keys subsequently used for bulk data encryption by symmetric algorithms and other applications including data integrity and authentication, and for encrypting small data items such as credit card numbers and PINs. Public-key decryption may also provide authentication guarantees in entity authentication and key establishment protocols. The main advantage of public key encryption systems used for distributed measurement systems is asymmetric computational power requirement for both sides: encryption and decryption, which can fit the DMCS architecture.

A public key encryption scheme is comprised of three algorithms: a key generation algorithm, an encryption algorithm and a decryption algorithm [2, 3]. In the next three subsections, the most popular asymmetric encryption algorithms will be presented with particular emphasis on arithmetic operations that need to be done for arbitrary length integers.

2.1 RSA public-key encryption

The RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adleman, is the most widely used public-key cryptosystem. It may be used to provide both secrecy and digital signatures and its security is based on the intractability of the integer factorization problem. This section briefly describes the RSA encryption scheme, notes on its security and some implementation issues can be found in the literature [3, 4].

Key generation for RSA public-key encryption
Each entity A should do the following:

- Generate two large random primes p and q, each roughly the same size.
- Compute $n = pq$ and $k = (p - 1)(q - 1)$.
- Select a random integer e, $1 < e < k$, $\gcd(e, k) = 1$.
- Compute the unique integer d, $1 < d < k$, such that
- $ed \equiv 1 \pmod{k}$.
- A's public key is (n, e); A's private key is d.

RSA public-key encryption algorithm

Encryption. *B* should do the following:

- Obtain *A*'s authentic public key (n, e).
- Represent the message as an integer m in $[0, n-1]$.
- Compute $c = me \bmod n$.
- Send the ciphertext c to *A*.

Decryption. To recover plaintext m from c, *A* should do the following:

- Use the private key d to recover $m = cd \bmod n$

Given the latest progress in algorithms for factoring integers, a 512-bit modulus n provides only marginal security from concerted attack. For long-term security, 1024-bit or larger modulus should be used.

In order to improve the efficiency of encryption, it is desirable to select a small encryption exponent e such as $e = 3$. A group of entities may all have the same encryption exponent e, however, each entity in the group must have its own distinct modulus. Thus a small encryption exponent such as $e = 3$ should not be used if the same message, or even the same message with known variations, is sent to many entities. Alternatively, to prevent against such an attack, a pseudorandomly generated bitstring of appropriate length should be appended to the plaintext message prior to encryption; the pseudorandom bitstring should be independently generated for each encryption (so called "salting the message") [4].

As was the case with the encryption exponent e, it may seem desirable to select a small decryption exponent d in order to improve the efficiency of decryption.

2.2 Rabin public-key encryption

The Rabin public-key encryption scheme was the first example of a provably secure public-key encryption scheme – the problem faced by a passive adversary of recovering plaintext from some given ciphertext is computationally equivalent to factoring [4, 6].

Key generation for Rabin public-key encryption

Each entity *A* should do the following:

- Generate two large random primes p and q, each roughly the same size.
- Compute $n = pq$.
- *A*'s public key is n; *A*'s private key is (p, q).

Rabin public-key encryption algorithm

Encryption. *B* should do the following:

- Obtain *A*'s authentic public key n.
- Represent the message as an integer m in $[0, n-1]$.
- Compute $c = m^2 \bmod n$.
- Send the ciphertext c to *A*.

Decryption. To recover plaintext m from c, *A* should do the following:

- Find the four square roots (from m_1 to m_4) of c mod n.
- The message sent was either m_1, m_2, m_3, or m_4. *A* somehow decides which of these is m.

Note: there exists simple algorithm for finding square roots of $c \bmod n = pq$ when $p \equiv q \equiv 3 \pmod 4$

Rabin encryption is an extremely fast operation as it only involves a single modular squaring. By comparison, RSA encryption with $e = 3$ takes one modular multiplication and one modular squaring. Rabin decryption is slower than encryption, but comparable in speed to RSA decryption.

2.3 ElGamal public-key encryption

The ElGamal public-key encryption scheme can be viewed as Diffie-Hellman key agreement in key transfer mode. Its security is based on the intractability of the discrete logarithm problem and the Diffie-Hellman problem [4, 6]. The basic ElGamal is shown below, the generalized ElGamal encryption schemes can be found in [4].

Key generation for ElGamal public-key encryption

Each entity *A* should do the following:

- Generate a large random prime p and a generator k of the multiplicative group Z_p of the integers modulo p.
- Select a random integer a, $1 \le a \le p-2$, and compute $k^a \bmod p$.
- *A*'s public key is (p, k, k^a); *A*'s private key is a.

ElGamal public-key encryption algorithm

Encryption. *B* should do the following:

- Obtain *A*'s authentic public key (p, k, k^a).
- Represent the message as an integer m in $[0, p-1]$.
- Select a random integer i, $1 \le i \le p-2$.
- Compute $j = k^i \bmod p$ and $l = m \cdot (k^a)^i \bmod p$.
- Send the ciphertext $c = (j, l)$ to A.

Decryption. To recover plaintext m from c, *A* should do the following:

- Use the private key a to compute $j^{p-1-a} \bmod p$.
- Recover m by computing $(j^{-a}) \cdot l \bmod p$.

3. NTL LIBRARY

Number Theory Library is a high-performance, portable C++ library providing data structures and algorithms for arbitrary length integers; for vectors, matrices, and polynomials over the integers and over finite fields; and for arbitrary precision floating point arithmetic [10].

NTL provides high quality implementations of state-of-the-art algorithms for:

- arbitrary length integer arithmetic and arbitrary precision floating point arithmetic;
- polynomial arithmetic over the integers and finite fields including basic arithmetic, polynomial factorization, irreducibility testing, computation of minimal polynomials, traces, norms, and more;
- lattice basis reduction, including very robust and fast implementations of Schnorr-Euchner, block Korkin-Zolotarev reduction, and the new Schnorr-Horner pruning heuristic for block Korkin-Zolotarev;
- basic linear algebra over the integers, finite fields, and arbitrary precision floating point numbers.

NTL provides a clean and consistent interface to a large variety of classes representing mathematical objects. It provides a good environment for easily and quickly implementing new number-theoretic algorithms, without sacrificing the performance.

NTL is written and maintained by Victor Shoup with some contributions made by others. NTL is free software,

and may be used according to the terms of the GNU General Public License.

3.1 Large Numbers' representation in NTL

The class ZZ is used to represent signed arbitrary length integers. Routines are provided for all of the basic arithmetic operations, as well as for some more advanced operations such as primality testing. Space is automatically managed by the constructors and destructors. This module also provides routines for generating small primes, and fast routines for performing modular arithmetic on single-precision numbers.

One can compute with ZZs much as with the regular data types, in that most of the standard arithmetic and assignment operators can be used in a direct and natural way. The C++ compiler and the NTL library routines automatically take care of all the bookkeeping involved with memory management and temporary objects.

For every function in NTL, there is a procedural version that stores its result in its first argument. The reason for using the procedural variant is efficiency: using an operator usually causes a temporary ZZ object to be created and destroyed, whereas the procedural version will not create any temporaries. Where performance is critical, the procedural version is to be preferred [10].

4. LABVIEW AND CRYPTO-G LIBRARY

It is well known that the LabVIEW environment has built in the huge number of libraries and programming tools. However, the lack of elements related to security of information systems, prevents the possibility of creating secure, cryptographic systems. In addition, well equipped mathematical library, have some limitations that prevent the efficient implementation of specific cryptographic algorithms. The basic problem is limited to 32 bits, from version 8.0 increased to 64 bits, integer numbers precision. This limitation prevents, among others, for the immediate implementation of secure encryption algorithms with public key, in which the huge primes are used (for example primes with one hundred digits).

After the analysis of the on-market availability of existing cryptographic solutions for LabVIEW environment, the Crypto-G library was found [9]. This library, provided by the VARTOR Technology Solutions as a shareware, is treated as unauthorized by National Instrument set of tools for LabVIEW environment (LabVIEW Toolkit). Crypto-G is advertised as the most comprehensive cryptographic library for LabVIEW and contains over 50 functions (Virtual Instruments) including the following functions:

- Symmetric Encryption
  - Advance Encryption Algorithm (AES)
  - Data Encryption Algorithm (DES)
  - SKIPJACK, TEA, BLOWFISH
- Hashing
  - Secure Hash Algorithm (SHA-1)
  - Message Digest 2 and 5 (MD2, MD5)
- Message Authentication
  - Keyed-Hashed Message Authentication (HMAC)
  - Data Authentication Code (DAC)
  - Random Number Generators (Based on SHA-1)
- Several Miscellaneous VIs
  - Large Numbers library (Beta)
  - Key Exchange Algorithm (KEA) (Beta)

Nevertheless, a set of encryption algorithms that is available in the Crypto-G library is rather small. The encryption algorithms are limited to a few systems with a private key, the public key systems are not implemented at all, a shortcut functions are limited to three, in the version without the key. The lack of public key encryption algorithms is probably the result of issues discussed earlier, namely the limited numbers' precision. Although the library contains a Large Number sub-palette, it is only the pre-release version (so called beta version) which is incomplete, inefficient and contains many errors.

As it was said in the introduction in the previous work, the authors have analyzed the LabVIEW environment capabilities for efficient implementation of cryptographic algorithms [1]. On the basis of the conclusions of that analysis and due to the issues presented above, the authors decided to develop a new tool for LabVIEW environment - a Large Number Library. This library allows for the computation on numbers with arbitrary (within the limits of available memory) number of decimal digits and would allow one to build asymmetric cryptographic systems for both, data encryption and the generation of secure digital signatures.

5. LN LIBRARY FOR LABVIEW ENVIRONMENT

5.1 Large Numbers' representation

The integers of arbitrary length can be represented in many ways. First of all they are not negative numbers in range from 0 to LN_{max} limited by the maximum number of digits (e.g. 256 or more). To make implementation the most elegant and efficient, the large number is represented as an array of bytes, which are present in LabVIEW as unsigned, 8-bit integers. Every byte acts as a digit in the 256-based system. The bytes are placed in the little-endian order, that means the first array element (index 0) represents weight 256^0, the next (index 1) 256^1, and the last (index k) 256^k.

5.2 Used algorithms

The theory and implementation notes for both, integer and modular multiple-precision arithmetic algorithms can be found in [4, 7] and also in the source files of NTL library [10]. For example, the multiple-precision addition algorithm is shown below [4].

Multiple-precision addition

INPUT: positive integers x and y, each of $n + 1$ base b digits.

OUTPUT: the sum $x + y = (w_{n+1}w_n \cdots w_1w_0)_b$ in radix b representation.

1. $c \leftarrow 0$ (c is the carry digit).
2. For i from 0 to n do the following:
   2.1. $w_i \leftarrow (x_i + y_i + c) \bmod b$.
   2.2. If $(x_i + y_i + c) < b$ then $c \leftarrow 0$; otherwise $c \leftarrow 1$.
3. $w_{n+1} \leftarrow c$.
4. Return$((w_{n+1}w_n \cdots w_1w_0))$.

5.3 List of implemented functions

All libraries' elements can be divided into several categories. All categories and functions are listed in Table 1.

Table 1. List of functions implemented in LabVIEW LN Library.

Arithmetic functions	
LN_Add	adds two LN arguments
LN_Subtract	subtracts two LN arguments (returns underflow if the result is negative)
LN_Multiply	multiplies two LN arguments
LN_Divide	divides two LN arguments (returns quotient and reminder)
LN_Square	raises one LN argument to the square
LN_RightShift	shifts right one LN argument (division by 2)
LN_LeftShift	shifts left one LN argument (multiplication by 2)
Comparison functions	
LN_Equal_0	tests if argument is equal 0
LN_Equal	tests if two arguments are equal
LN_Greater	tests if one argument is grater (or optionally equal) than other
Modular arithmetic functions	
LN_Modulus	calculates the integer reminder of two LN arguments
LN_NegateModulo	calculates the negative of one LN argument modulo second LN argument
LN_AddModulo	calculates the sum of two LN arguments modulo third LN argument
LN_SubtractModulo	calculates the difference of two LN arguments modulo third LN argument
LN_MultiplyModulo	calculates the product of two LN arguments modulo third LN argument
LN_SquareModulo	calculates the one LN argument raised to the square modulo second LN argument
LN_PowerModulo	calculates the one LN argument raised to the second LN argument modulo third LN argument
LN_InverseModulo	calculates the inverse (if exists) of one LN argument modulo second LN argument or the GCD of two LN arguments (otherwise)
Random number generation and primality test	
LN_PRNG	generates a set of pseudo-random large numbers
LN_PrimalityTest	tests if LN argument is a prime number
Utility functions	
LN_LN2String	converts LN argument into decimal string
LN_String2LN	converts decimal string into LN number

5.4 Implementation details

During the development, all previously mentioned functions were implemented in two variants and the accuracy and effectiveness of both alternatives was tested. The results should give the recommendations for further work, namely for the implementation of asymmetric cryptographic systems such as RSA, Rabin or ElGamal systems. The two ways of implementation are the results of previously made analysis, and are as follows:

- implementation in the native LabVIEW graphical language G (see fragment of the block diagram of LN_Add function in Fig. 1);
- implementation using the external software modules written in C++ (with sources from NTL library) and compiled into a DLL (see fragment of code of Add function below).

```
// fragment of Add function
pc = c;
carry = 0;
do {
   long t = (*(++a)) + (*(++b)) + carry;
   carry = t >> NTL_NBITS;
   *(++pc) = t & NTL_RADIXM;
   i--;
} while (i);
// end of code fragment
```

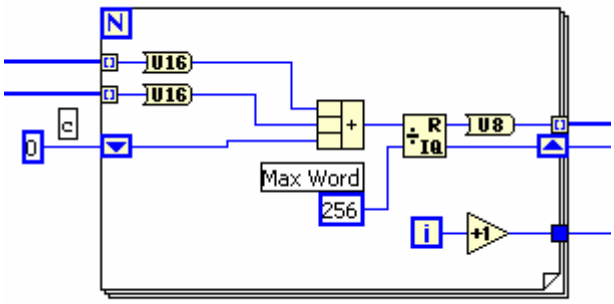

Fig. 1. Fragment of the block diagram of LN_Add function.

To ensure compliance with established types of large numbers representation in the LabVIEW environment and because of the limitations of possible data types that can be transmitted through the Call Library Function Node, in all functions exported from a DLL library, the conversion between objects of class ZZ and byte arrays was made.

6. TESTS RESULTS

In order to examine the accuracy and effectiveness of the developed functions, in both implementation variants, the proper test applications were built. The test applications were designed in a way that allows not only functional validation of implemented operations but also could measure the execution time for every function. Due to the specificity of the tested library, two test applications were developed: one for arithmetic operations (including reduction modulo N) and the other one for the modular arithmetic operations. In each of them the tested functions were run in the loop, for randomly generated input data.

All the library functions and tests applications were developed in the LabVIEW environment in version 8.5, external modules were written in C++ and compiled to a DLL in a Microsoft Visual C++ 2005 Express Edition environment. The execution times were obtained by measuring the timestamps within the code. The timestamp

measurements were conducted for executables built with Application Builder tool.

Experimental results for basic integer and modular arithmetic functions are shown in Table 2 and in Table 3, respectively. Tables contain execution times in seconds, for 1 million iterations, for input numbers contained of 10 and 100 digits (T_{10} and T_{100} respectively). For modular arithmetic the modulus contained two times more digits than the arguments (quite typical situation for public key algorithms). The results of our functions (bold font) are compared to the DLL version (the prefix "extern") and the original Crypto-G version (the prefix "crypto").

Table 2. Test results of integer arithmetic functions.

Function name	T_{10} [s]	T_{100} [s]
LN_Add	5,3	7,5
extern_LN_Add	4,1	5,5
crypto_LN_Add	7,8	10,5
LN_Subtract	4,3	7,4
extern_LN_ Subtract	3,5	5,4
crypto_LN_ Subtract	30,9	37,4
LN_Multiply	5,8	143,3
extern_LN_Multiply	4,1	10,4
crypto_LN_Multiply	9,1	144,8

Table 3. Test results of modular arithmetic functions.

Function name	T_{10} [s]	T_{100} [s]
LN_AddModulo	14,2	20,1
extern_LN_AddModulo	5,3	9,1
crypto_LN_AddModulo	417,9	322,4
LN_SubtractModulo	9,0	15,3
extern_LN_ SubtractModulo	10,4	13,4
crypto_LN_ SubtractModulo	7088,3*)	not tested
LN_MultiplyModulo	18,9	153,1
extern_LN_MultiplyModulo	5,3	12,9
crypto_LN_MultiplyModulo	1577,1	1398,3

*) enormous execution time and also incorrect results

The analysis of the presented results leads to the following conclusions.

- Our Large Number library is in general much more efficient than the beta version of Crypto-G solution (in particular for the modular functions).
- Addition and subtraction functions written in G code are in general only a little bit slower than the external versions for both, the integer and modular version.
- Our multiplication functions written in G code are much slower than the external versions, especially for input data with great number of digits. This is probably due to yet not optimized memory operations and used classical algorithms (in the future we plan to implement more efficient algorithms, like Karatsuba multiplication [4]).
- Our modular multiplication functions for typical large numbers (about one hundred decimal digits) are much slower then DLL version but still almost one order of magnitude faster than the Crypto-G version.

7. CONCLUSIONS

Paper concerns the field of Distributed Measurement-Control Systems and in particular communication security issue in such systems. The huge role of the software in DMCS is shown, and the need to develop some cryptographic tools for such systems is presented. These tools would give the DMCS' developers the opportunity to design secure systems in an easy and intuitive way. The paper provides basic theoretical knowledge about public-key encryption systems which are claimed to be useful to create safe and secure DMCS. The main goal of the paper is to present the new mathematical tool for LabVIEW, the Large Numbers library, which is necessary for further implementation of specific, asymmetric algorithms such as RSA or Rabin encryption systems. The LN library was implemented in two variants: using only native LabVIEW code (G language) and using external software modules (DLLs). Implemented functions were tested and the tests' results lead to the following conclusions.

There is a possibility to implement the Large Number library in pure G code. The efficiency of such a solution could be quite similar to the version using external software modules when the fast algorithms are used and some code optimization steps are performed. The Large Number library written in pure G code can be used not only in DMCS' PC-based modules but also in FPGA-based hardware solutions which can be programmed directly from the LabVIEW environment (using the LabVIEW FPGA Module).

REFERENCES

[1] P. Bobiński, W. Winiecki, "LabVIEW Capabilities Analysis for Cryptographic Algorithms Implementation" (in Polish "Analiza możliwości wykorzystania środowiska LabVIEW do implementacji algorytmów kryptograficznych", *Przegląd Elektrotechniczny*, vol. LXXXIV, no. 5, (2008), pp. 228-231.

[2] Henk C., A. van Tilborg (Ed.), *Encyclopedia of Cryptography and Security*, Springer, 2005

[3] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer Verlag, New York, 1994.

[4] Menezes, P. Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press Inc., 1997.

[6] Schneier B, *Applied Cryptography Second Edition: protocols, algorithms, and source code in C*, John Wiley & Sons, 1996

[7] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, Cambridge University Press, Cambridge 2005.

[8] W. Winiecki, T. Adamski, P. Bobiński, R. Łukaszewski, "Security of Distributed Measurement and Control Systems" (in Polish "Bezpieczeństwo rozproszonych systemów pomiarowo-sterujących (RSPS)", *Przegląd Elektrotechniczny*, vol. LXXXIV, no. 5, (2008), pp. 220-227.

[9] Crypto-G: cryptographic library for LabVIEW, http://www.vartortech.com/cryptog.html

[10] NTL: A Library for doing Number Theory, http://www.shoup.net/ntl/