www.czasopisma.pan.pl PAN www.journals.pan.pl
INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2011, VOL. 57, NO. 3, PP. 257–262
Manuscript received June 19, 2011; revised September 2011. DOI: 10.2478/v10177-011-0034-7

Implementation and Analysis of Elliptic Curves-Based Cryptographic Algorithms in the Integrated Programming Environment

Robert Lukaszewski, Michal Sobieszek, and Piotr Bilski

***Abstract*—The paper presents the implementation of the Elliptic Curves Cryptography (ECC) algorithms to ensure security in the distributed measurement system. The algorithms were deployed in the LabWindows/CVI environment and are a part of its cryptographic library. Their functionality is identical with the OpenSSL package. The effectiveness of implemented algorithms is presented with the comparison of the ECDSA against the DSA systems. The paper is concluded with future prospects of the implemented library.**

***Keywords*—Cryptography, distributed measurement systems, integrated programming environments.**

I. Introduction

CURRENTLY one of the most pressing issues in the distributed measurement and control systems (DMCS) is security of the data transmitted between the nodes. Large distances between operational units enable the intruder to penetrate the system's infrastructure. The problem becomes more important as the wireless technologies (Bluetooth, ZigBee or WiFi) gain popularity. Although some standards have cryptographic protocols embedded (ZigBee or WiFi), there are multiple specialized protocols without the required security. With the increasing terrorist threat it is important to ensure safety in systems crucial for the society [1]. Design of the measurement system software is based on the specialized integrated programming environments, such as LabVIEW, VEE or LabWindows/CVI. They include multiple libraries with functions for communication, signal processing and mathematical operations. Unfortunately, cryptographic algorithms are still absent, justifying their implementation in user-defined modules. Similar attempts have been made [2] with promising results. Elliptic curves cryptography (ECC) is a set of methods for encrypting and decrypting data based on algebraic operations in the finite field of integer numbers. It is currently popular and widely exploited because of its small memory requirements and computational resources compared to other methods [3]. These algorithms can be implemented in small devices such as embedded systems, smart cards and sensor grids. Their advantage is high efficiency. The latter is important, as multiple applications of DMCS require working in the real-time mode. This imposes keeping the time limitations, which is possible only for operations fast enough. The paper presents the implementation of the ECC library in the LabWindows/CVI environment. It is a popular tool for designing DMCS using the C language. As it lacks the proper cryptographic library, it was a good target to implement it. Two methods of assimilating the algorithms into LabWindows/CVI were selected. The first one consists in obtaining the ready-made dynamically linked library - DLL (such as in OpenSSL) called from the C code in the programming environment. The second option is to adapt the library to the form accepted by the environment. The paper presents implementation and verification of both solutions in the LabWindows/CVI. In section II fundamentals of cryptography required to implement the project are introduced. Section III contains description of the ECC library for LabWindows/CVI. In section IV efficiency tests of the library are presented. Finally, section V contains conclusions and future prospects.

R. Lukaszewski, M. Sobieszek, and P. Bilski are with the Institute of Radioelectronics, Warsaw University of Technology, Warsaw, Poland (e-mails: r.lukaszewski@ire.pw.edu.pl, m.j.sobieszek@elka.pw.edu.pl, p.bilski@ire.pw.edu.pl).

P. Bilski is also with the Department of Applied Informatics, Warsaw University of Life Sciences, Warsaw, Poland (e-mail: piotr_bilski@sggw.pl).

II. Cryptography in DMCS

The multitude of devices used in DMCS requires uniform methods of data transmission. Most systems use computer network with WiFi or Ethernet standard. In the TCP/IP protocol stack, used in local and wide area networks (LAN and WAN, respectively), there are means of security. Unfortunately, industrial networks often have minimal functionality because of the high efficiency and determinism requirements. Here, cryptographic algorithms should be individually implemented. The domain has a long history and is based on numbers theory, algebra, computer algorithms and the probability calculus. Although there are multiple purposes of cryptography, in DMCS it is used to ensure confidentiality of the transmitted data. It is especially important in large facilities, such as nuclear plants or water pumping stations, where parameters are measured by sensors in remote locations and sent to processing or control nodes (such as industrial or personal computers). To avoid the intrusion, the plain data (measurements or control commands) are encrypted using keys to obtain ciphers and sent to the destination node (see Fig. 1). There, the cipher is decrypted to get the plain data again. Two main aspects of the security assurance are vital here: method (function) of exchanging the plain data into the cipher and the key used during this process. In the symmetric cryptography for both encryption and decryption the same key is used. It must then be kept secret from intruders. The best known algorithms are DES (Data Encryption Standard), 3DES (Triple DES) and

AES (Advanced Encryption Standard) [2], [4]. Although the symmetric cryptography is fast [5] and secure (it is resilient to the attacks using quantum algorithms), the main problem is the key distribution, difficult in DMCS. Therefore in the Internet the asymmetric cryptography is a standard, where the key is divided into two parts (public and private), one of which is available for everyone in the environment. The main idea is that the knowledge of the public key does not allow the intruder to act as a legal party. If the public key is for encryption, the intruder can use it to encrypt his data, but he will not be able to decrypt anything encrypted by the private key. The most popular asymmetric systems are Rabin's cipher or RSA (Rivest, Shamir, Adelman). Such systems are also used for creating digital signatures: DSA (Digital Signature Algorithm) and to distribute secret keys [4], [6]. The main disadvantage of such cryptography is its potential security problem. Quantum computation (Shor's algorithm) is able to compromise even the strongest RSA system [7].

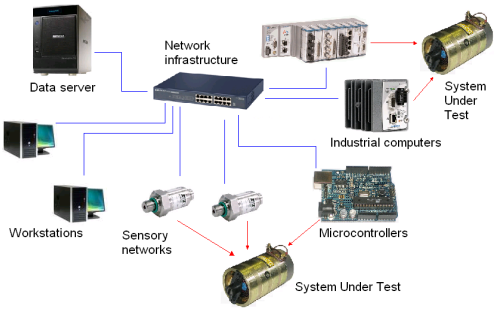

Fig. 1. Structure of the contemporary DMCS.

The alternative for the traditional algorithms are elliptic curves-based cryptographic (ECC) systems. It is a relatively new approach (proposed in 1985 by Miller and Koblitz) to obtain the fast and reliable public key cryptography. The elliptic curve E over the field F is the following curve (1):

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \qquad (1)$$

Its solutions are rational points P=(x,y) with the additional "point of singularity" O. The points (x,y) form a group G, with O being its identity element. If F is finite, the cardinality of a group is also finite [8]. In practice, a_1, a_2, a_3 are equal to zero, so (1) is simplified to (2).

$$Y^2 = X^3 + aX + b \qquad (2)$$

where a,b $\in F$. The elliptic curves were successfully used in solving multiple problems (proof of the Great Fermat Theorem) [9]. The ECC systems are useful in cryptography because of the difficulties in finding the discrete logarithm in the group (x,y), i.e. for given g, y $\in G$, where g is the base of the group G, finding the integer value x such that $g^x = y$. Usually the group is a cyclic group (ring of integers) modulo n. The problem (called Elliptic Curve Discrete Logarithm Problem - ECDLP) has multiple solutions (for instance, Pohling-Hellman, Pollard or Shanks methods), they have the exponential computational complexity. This enables using shorter keys in ECC than in traditional cryptography, maintaining the same security level (see Table 1) [10]. It is important in embedded systems and other small devices with limited memory and processor's speed [3]. The ECC is approved by NIST [10] for the commercial usage.

TABLE I
NIST GUIDELINES FOR PUBLIC KEY SIZES FOR AES

ECC Key Size (bits)	RSA Key Size (bits)	Ratio	AES Key Size (bits)
163	1024	1:6	–
256	3072	1:12	128
384	7680	1:20	192
512	15360	1:30	256

The application of elliptic curves to cryptography can be explained using the ECDH system. Here both nodes in DMCS willing to communicate (N_1 and N_2) must define the parameters: the rational point P of the elliptic curve E. Both are public constants. Then the procedure is as follows:

1) Both nodes randomly generate two integers, k_1 and k_2, which are private keys of N_1 and N_2, respectively.
2) Nodes compute parts of the public key: $k_1 \cdot P$ and $k_2 \cdot P$ and exchange them through the communication medium.
3) Both nodes are able to compute $k_1 \cdot k_2 \cdot P$ obtaining the same secret information, which cannot be acquired by the intruder without knowing both k_1 and k_2. In particular, he cannot calculate it from $k_1 \cdot P$ and $k_2 \cdot P$, although P is known.

Currently the minimum acceptable level of security requires the 128-bit key (in AES). Similar ECC systems (ECDH, ECDSA and SHA-256) require 256-bit keys. Analogous security in traditional asymmetric algorithms (DH, DSA, RSA) requires 2048-bit keys. The 256-bit key AES and 384-bit ECC are required to encrypt and decrypt top secret data [11].

III. ECC IMPLEMENTATION IN LABWINDOWS/CVI

The adaptation of ECC algorithms in LabWindows/CVI environment used the OpenSSL project, which source code is freely available. Its aim is to provide universal security tool for the Internet. It implements Secure Sockets Layer (SSL) and Transport Layer Security (TLS), openssl command tools and cryptographic library, i.e. libcrypto . Therefore it is a good source of the ready-made algorithms, requiring adjustment to the programming environment format. Also, OpenSSL is quickly developing; the presented work used version 0.8.9j, although currently version 1.0.0d is available. The implementation exploited the code of the libcrypto library in two ways (see section 1). Firstly, it was compiled to the DLL file, which could be called from the LabWindows/CVI code. To do that, the Microsoft Visual Studio 2008 was used. Secondly, the code was modified to the native format of the environment (further called CVL). This way it could be used as any other part of the environment code. In this task, LabWindows/CVI 9.0 was used. The first task, i.e. creating the DLL file, is easy. It requires only executing the configuration script, creating the makefile for the Visual Studio C/C++ compiler. To build the library, the nmake command was applied. On the other hand, compilation of the library under LabWIndows/CVI was difficult. The built-in C compiler

is simple and does not provide multiple features offered by, for instance, Microsoft Visual C/C++ or Borland C. It does not support the assembler code, used by OpenSSL creators to increase efficiency of some algorithms. Although there is the method of assimilating the code ignoring unacceptable fragments, the speed of the resulting algorithm is slower than in the original version. To adapt the external library to the form acceptable by the specialized environment, the designer must know the structure of the software project created for the needs of DMCS. This imposes the knowledge of all file types used in the environment and relations between them. In LabWindows/CVI the configuration of the designed program is stored in the project file (see Fig. 2). It contains general information about program properties and the list of internal files. These are: user interface (i.e. the front panel of the designed instrument), code files (standard c files) and library (header) files (containing definitions of structures and functions used inside the program). The external files (DLLs) can also be accessed here, through the environment interface. The functionality of library files is facilitated by their function panels.

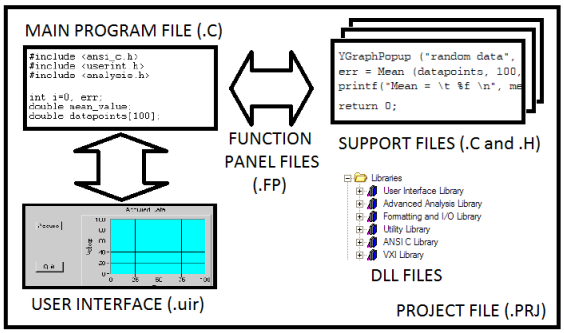

Fig. 2. Structure of the project in LabWindows/CVI environment.

The process of creating the native library consisted in two phases. The first one was to change the OpenSSL code to the form accepted by the integrated environment. The second step was to compile the modified code and transform it into the standard LabWindows/CVI form. The procedure requires the general knowledge about the process of compiling and loading the project into the memory of the operating system [12]. Particular stages are in Fig. 3.

Modification of the libcrypto code to be usable and compilable under LabWindows/CVI was also difficult, as it does not use makefiles. Therefore the file prepared for MS Visual Studio was useless. Rules for compilation are in the project file (with .prj extension). To use the OpenSSL source code, its c files were included in the project. To correctly build the library here, the preprocessor directives had to be prepared. They make the code platform-independent and all compilers know what to do with particular parts of the program. The majority of directives refers to header files, because they are available only for the specific operating system. In various platforms the same operations are performed by different versions of the code. In particular, the macrodefinitions to modify included indication of the 32-bit operating system (OPENSSL_SYSNAME_WIN32 and MK1MF_PLATFORM_VC_WIN32). Also the definition

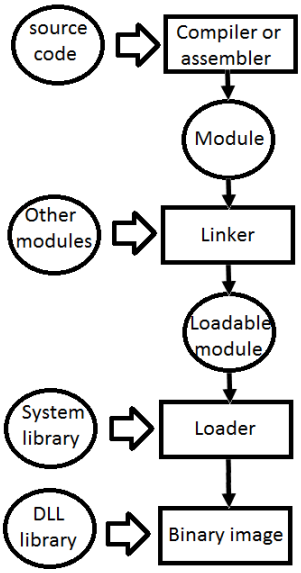

Fig. 3. Processing of the user program.

OPENSSL_IMPLEMENTS_strncasecmp had to be created to add the declaration of the strncasecmp function. It compares (case-insensitive) n characters of two strings. In LabWindows/CVI there are multiple header files named identically as in MS Windows. Some functions and structures required to import the OpenSSL code are in the lowlvlio.h file. Other had to be defined from scratch or rewritten under the same name. In the presented project, the following changes in the original cryptographic code had to be performed:

- Implementation of the structure from the times.h file, responsible for storing the duration of the operation measured by the times() function. In LabWindows/CVI variables of the clock_t type and clock() function were used instead.
- The Microsoft-native tchar type was replaced by ANSI char type.
- The fileno() function, returning the open file descriptor had to be defined from the beginning.
- The mode of opening the file. According to ANSI, for changing the mode of the open file into binary, the setmode() function is used. Because in LabWindows/CVI there are no such function, all files were opened in the binary mode by default.

After making the code compilable and linkable, the project was translated into the libcrypto.lib file. Finally, the LabWindows/CVI function panel files (with .fp extension) had to be created. With them, using both types of libraries (DLL and native) is identical. After importing the library to the project functions can be used as any other, existing in the environment. The functionality of the library is available through the tree (Fig. 4). Selection of the particular function using the left mouse button opens the function panel (Fig. 5) allowing for inserting values of input variables. Under the panel the code is generated. It is copied to the program file.

The functionality of the libcrypto library includes symmetric encryption and decryption systems (such as Blowfish, AES,

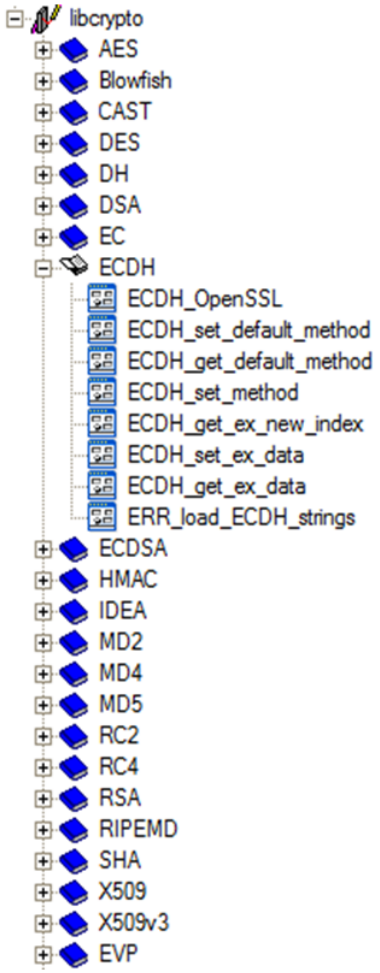

Fig. 4. Cryptographic library for LabWindows/CVI after import.

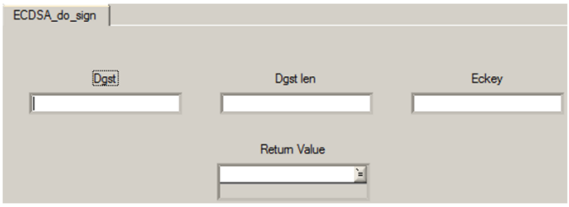

Fig. 5. The ECDSA_do_sign function panel in LabWindows/CVI.

RC5), public key cryptography (DSA, RSA, DH), digital signature systems (MD4, MD5, SHA-256, etc.). Support for X.509 certivicates and random number generator are also included. All algorithms are available through the LabWindows/CVI standard design interface and can be used as any other libraries.

IV. Tests of the ECC Algorithms Efficiency

Experiments with both versions of the library were performed on the personal computer equipped with Intel Pentium 4 processor (2.8 GHz) and 1 GB of RAM. The operating system was Windows XP. During experiments the system was in the "clean boot" mode, where only basic services are run. This allowed to minimize the influence of other applications to the tested software. In the general purpose operating system (GPOS) interaction with the user (such as moving the mouse cursor) disturbs the execution and scheduling of working processes. Experiments consisted in measuring mean and maximum durations of creating digital signatures using the ECDSA method with their verification. The second group of experiments covered establishing a key pair using the EDSH algorithm. Tests were performed for both libraries: CVL and DLL. The following section contains results for combinations of algorithms and libraries with necessary comments. The standard method of time measurement in C is the GetTickCount function. It is not appropriate for measuring durations of presented operations because of small resolution (10 ms) [13]. Digital signatures require more accurate measurements of the QueryPerformanceCounter and QueryPerformanceFrequency functions (available in MS Windows). The first one stores the actual value of the high resolution counter. To obtain the time measurement, this value is divided by the result of the second function. This way the time measurement is taken in the number of counter increments per second. The timer frequency is constant in a single operating system execution [14].

A. ECDSA Examinations

This algorithm is analogous to DSA. The experiments shown large differences between creating the signature and its verification (see Fig. 6). Durations of both operations increase with increasing the key length.

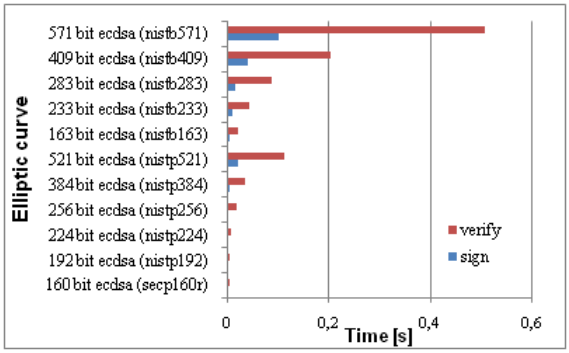

Fig. 6. Mean times of creating and verifying signature using ECDSA method from CVL.

The requirement of the real-time mode in the DMCS makes the mean durations a not adequate efficiency measure. More important are maximum times of the algorithm execution, which give the knowledge of the method behavior in the worst case. Both durations are proportional to the key length. The signature verification is longer than its creation (see Fig. 7). In GPOS the exact duration of the operation can never be determined accurately. The measured time depends on the operating system's state and configuration of other processes in memory [15].

Similar results were obtained for the DLL version of the library. The relation between the signature generation and verification is the same as for CVL. The DLL-based solution works faster - three to seven times, depending on the key length (see Fig. 8 and 9). This is because the DLL

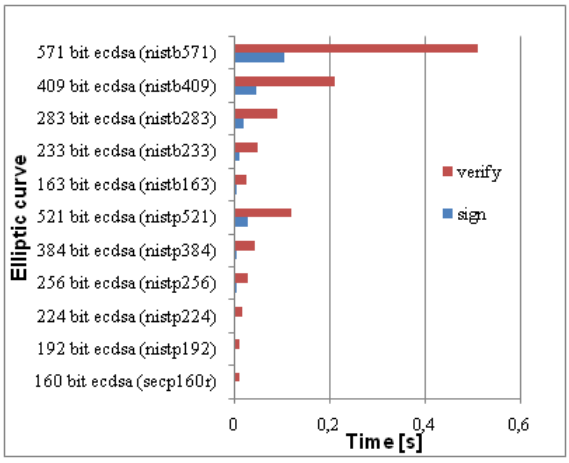

Fig. 7. Maximum times of creating and verifying signature using ECDSA method from CVL.

code was optimized by the Visual Studio compiler, while the LabWindows/CVI compiler required modifications in the code, degrading its effectiveness (see section 3). This is also confirmed by previous research [16].

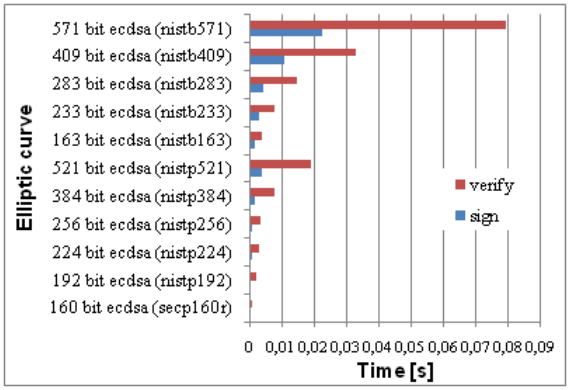

Fig. 8. Mean times of creating and verifying signature using ECDSA method from DLL.

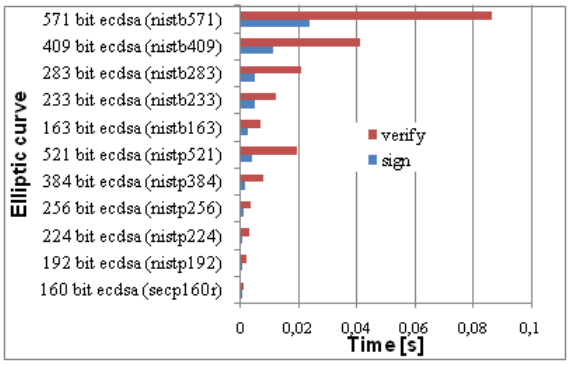

Fig. 9. Mean times of creating and verifying signature using ECDSA method from DLL.

B. ECDH Examinations

Experiments regarding the second algorithm had similar structure. Results obtained for DLL are, as before, much faster than their counterparts for CVL. The phenomenon of increasing the algorithm time with increasing key length is also visible here (see Fig. 10 and 11).

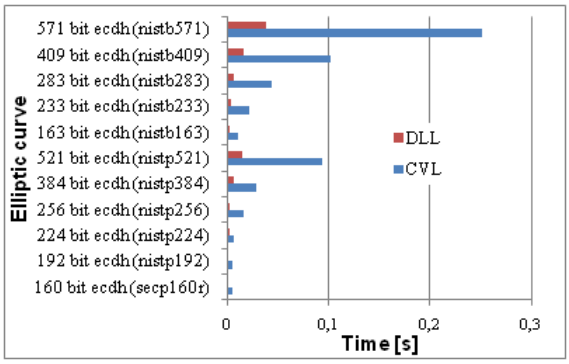

Fig. 10. Mean times of creating and verifying signature using ECDH method from DLL and CVL.

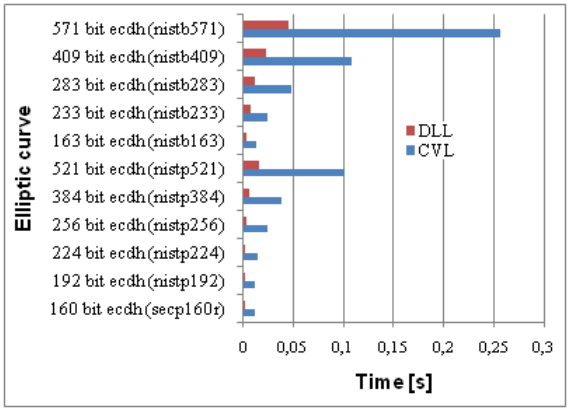

Fig. 11. Mean times of creating and verifying signature using ECDSH method from DLL and CVL.

C. Comparison Between DSA and ECDSA

The presented results were used to compare mean durations of generating and verifying digital signatures for DSA and ECDSA systems in both versions of libraries. The comparison between algorithms must be based on configurations giving the same security level, which is obtained using different key lengths. For example, the ECDSA with 192-bit key assures similar security as DSA with 1024-bit key, which is currently minimal, considered safe. The 224-bit ECDSA key reflects the 2048-bit DSA key, and so on. Again DLL-based algorithms work faster than the ones from CVL. The shortest durations were obtained using ECDSA algorithm from DLL (see Fig. 12).

Comparison between mean times of digital signatures verification (see Fig. 13) gives similar results as during their generation. The CVL-based DSA is the slowest solution, while the DLL-based ECDSA - the fastest.

The ECDSA system requires much shorter keys (even 9 times) to assure the same level of security as DSA. Storing them requires larger memory and processor time. Therefore ECDSA is more attractive for both general purpose (such as Internet) and specialized (industrial networks) systems.

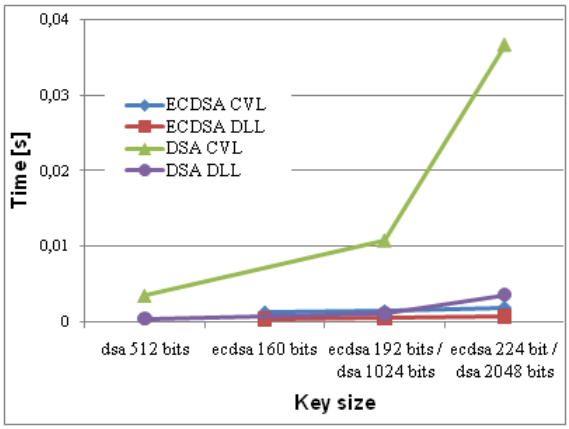

Fig. 12. Mean duration of generating the digital signature - comparison of algorithms and libraries.

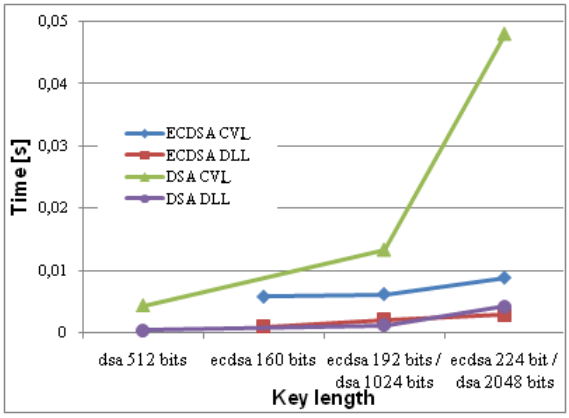

Fig. 13. Mean duration of verifying the digital signature - comparison of algorithms and libraries.

V. Conclusions

The paper presented the implementation of the ECC algorithms in the integrated programming environment on the example of LabWindows/CVI. The library from the OpenSSL package was assimilated to the form acceptable by this environment using two approaches (DLL and native). Experiments were conducted to show effectiveness of both solutions for the particular algorithms. The comparison between the traditional digital signature algorithms and their ECC-based counterparts was performed. Functions and algorithms originating from the DLL library are always the fastest ones. The LabWindows/CVI-native version is not optimized because of multiple drawbacks, for example elimination of the assembler code, which is faster than the one written in C. Also, ECDSA system is faster than DSA, requiring shorter keys to get the same security level. This confirms usefulness of the ECC algorithms, which probably will become the predominant standard in the nearest future. Although the DLL-based system is more efficient and flexible, the native library gives open-source advantages. The designer has the control over all function applied in the code and is able to modify it from the programming environment level. The DLL requires additional mechanisms to ensure software integrity (for instance, applying infrastructure certificates to sign parts of the code). Increasing role of security in DMCS makes implementation of cryptographic algorithms and procedures in specialized software development environments needed and useful. The designer should be able to apply fundamental security measures to the software targeted for industrial computers and embedded systems. Therefore works similar to [2] should be continued in the future. Adjusting existing library to the programming environment's needs is simpler and faster than creating a new one from scratch. As most cryptographic algotihms is already implemented, such an approach decreases the duration of the design procedure. Therefore the approach presented in the paper should be applied whenever there is the need to add the functionality to the specialized environment.

References

[1] W. Winiecki, T. Adamski, P. Bobinski, and R. Lukaszewski, "Bezpieczenstwo rozproszonych systemow pomiarowo-sterujacych (rsps)," *Przeglad Elektrotechniczny*, vol. LXXXIV, pp. 220–227, May 2008, in Polish.

[2] P. Bilski and W. Winiecki, "Multi-core implementation of the symmetric cryptography algorithms in the measurement system," *Measurement*, no. 43, pp. 1049–1060, 2010.

[3] The basics of ecc. [Online]. Available: http://www.certicom.com/index.php/the-basics-of-ecc

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

[5] T. Good and M. Benaissa, "Aes on fpga: from the fastest to the smallest," in *Proc. Cryptographic Hardware and Embedded Systems (CHES'2005)*, Edinburgh, United Kingdom, Aug 2005, pp. 427–440.

[6] B. Schneier, *Applied Cryptography*, 2nd ed. New York: John Wiley & Sons, 1996.

[7] D. Mermin, "Breaking rsa encryption with a quantum computer: Shor's factoring algorithm," *Lecture notes on Quantum computation*, pp. 481–681, 2006.

[8] N. Koblitz, *Algebraic Aspects of Cryptography*. Springer-Verlang Gmbh, 1999.

[9] D. Henderson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer-Verlang Gmbh, 2004.

[10] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, *Recommendation for Key Management Part 1: General (Revised)*, National Institute of Standards and Technology (NIST) Recommendation for Cryptographic Key Management Standard 800-57, March 2007. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf

[11] N. S. Agency. Nsa suite b cryptography. [Online]. Available: http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

[12] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. John Wiley & Sons, 2003.

[13] Microsoft. How to use queryperformancecounter to time code. [Online]. Available: http://support.microsoft.com/kb/172338

[14] ——. Queryperformancefrequency function. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms644905(VS.85).aspx

[15] P. Bilski and W. Winiecki, "Time optimization of soft real-time virtual instrument design," *IEEE Transactions on Instrumentation and Measurement*, vol. 54, no. 4, pp. 1412–1416, Aug 205.

[16] R. Lukaszewski and M. Sobieszek, "Biblioteka kryptograficzna w labwindows," in *Metrologia dzis i jutro*, Gdansk, Poland, 2009, pp. 113–120, in Polish.