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

MULTICORE IMPLEMENTATION OF THE AES ALGORITHM IN THE MEASUREMENT SYSTEM

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

[1] Warsaw University of Life Sciences, Faculty of Applied Informatics and Mathematics, Warsaw, Poland, piotr_bilski@sggw.pl, pbilski@elka.pw.edu.pl
[2] Warsaw University of Technology, Institute of Radioelectronics, Warsaw, Poland, W.Winiecki@ire.pw.edu.pl

Abstract – The paper presents the implementation of the Advanced Encryption Algorithm (AES) in the measurement system, where the virtual instruments equipped with the multi-core processors are used. The encryption algorithm is first presented, then its modification to take advantage over the multi-core processor is described. Implementation of the latter in the virtual instrument working under the Real-Time (RT) mode is presented. Finally, test vectors are used to verify the validity of the modified algorithm and comparison between the traditional algorithm and the version using the parallel computations is performed.

Keywords: virtual instrumentation, cryptography, Real-Time systems

1. INTRODUCTION

The influence of the computer technologies on the modern measurement systems is steadily increasing. Multiple solutions not only allow connecting the traditional devices to the computers, but require the computer network or processors to perform calculations on the acquired data. As the distributed measurement systems are a common implementation, the problem becomes the security of the data transferred between the nodes of the system. The threats related to the traditional computer network are well identified and complete software and hardware solutions were implemented. Security of the distributed measurement systems is still not clearly defined, but numerous incidents indicate that in the near future all the data in such a system will have to be secured from the unauthorized access. The most popular method of the data security is the implementation of the cryptographic algorithms. In the measurement systems involving the software (for example, virtual instruments - VI), implementation of the algorithm in the application run by the processor is the correct approach.

The measurement system can be supplemented with the encryption schemes in two ways, related to its structure. Generally, there are two types of the nodes. The first one are measurement nodes, where the data acquisition (DAQ) operations are performed. They are usually sensory networks or general purpose computers equipped with the DAQ hardware, etc. The nature of the measurements depends on the character of the analyzed phenomenon: temperature, pressure, voltage, current and similar. When the measured quantities are converted into the form acceptable by the computer system, they must be sent to the second element of the system, i.e. data processing server. It is the central node of the system, responsible for storing the gathered information and performing the data processing. The scheme of the system is presented in Fig. 1.

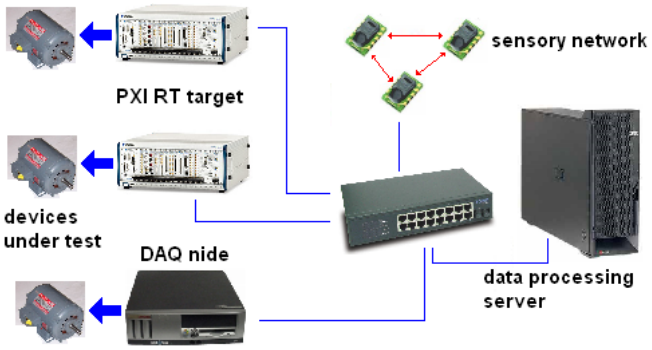

Fig. 1. Architecture of the distributed computer measurement system.

Introduction of the encryption schemes into the system is asymmetric. When the DAQ nodes gather measured data, they must be encrypted before sending to the server. Unfortunately, a very limited computing power is available here, such as single core processors of lower frequencies. The cryptographic algorithms are usually computationally expensive, and may be difficult to implement in the small microcontrollers, such as Intel 8051. On the other side there is the high performance computing machine working as a server. Although it has greater processing abilities, its responsibility is the simultaneous decryption of the measurement data sent from multiple nodes and encryption of the configuration data sent to them.

It is often important to acquire the data from the sensors and process them within the predefined time limits. To ensure such determinism, the RT system is needed. The latter can easily be implemented on the server (personal computer), and operate the measurement application designed in on of the integrated programming environments, such as National Instruments LabVIEW, or Agilent VEE. One of the newest achievements in the measurement systems is the ability to use the multi-core processors in the RT mode. This gives the opportunity to increase the speed

ISBN 978-963-88410-0-1

of computations and overcome the problem of the RT data processing.

The paper presents the implementation of the AES algorithm in the RT measurement server. The former was designed using the LabVIEW environment and run on the four-core Intel processor. Organization of the paper is as follows. In section 2 the AES algorithm is presented. Section 3 contains discussion of the modification of the original algorithm to take advantage of the multiple cores. In section 4 implementation of the algorithm in LabVIEW environment is described. Section 5 contains the verification of the algorithm and measurements of its efficiency. In section 6 there are conclusions and future prospects.

2. THE AES ALGORITHM DESCRIPTION

The AES algorithm is the most popular symmetric block cipher, approved by the National Institute of Standards and Technology in 2001 [1]. The purpose of the algorithm is to replace the older and less reliable algorithms, such as Data Encryption Standard (DES). The algorithm operates on the 128-bit (16-byte) data blocks (called plain text, in the measurement system the latter can be represented by the acquired samples, arrays etc.) and uses 128-, 192-, or 256-bit key to obtain 128-bit cipher. Application of the algorithm consists of encryption (transformation of the plain text into the cipher) and decryption (the opposite transformation). In both operations, the same key is used. The hardware and software requirements are relatively small, making the AES the most popular symmetric standard [2]. Therefore it can be successfully used in the simple measurement nodes of the distributed system, where sensors and DAQ hardware gather the desired quantities, then create blocks, encrypt them and send to the server. The implementation of the algorithm is based on the substitution-permutation network.

2.1. AES encryption scheme

The pseudocode for the AES encryption is presented in Fig. 1. It transforms the input data (*IN* – measurements) into the cipher (*OUT*) using the expanded key *W*.

```
AEScipher(IN(4·Nb), OUT(4·Nb), W[Nb·(Nr+1)])
begin
    state[4,Nb] = IN
    AddRoundKey(state, W[0, Nb-1])
    for i = 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, W[i·Nb, (i+1)·Nb-1])
    end
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, W[Nr·Nb, (Nr+1)·Nb-1])
    OUT = state
end
```

Fig. 2. Pseudocode of the AES encryption [1].

where *Nb* is the number of the processed bytes (here four), *Nr* is the number of the algorithm's iterations (depending on the length of the key, equal to 10, 12, or 14 respectively), *IN* is the 128-bits long plain text, *OUT* is the output vector (cipher, 128-bits long), *W* is the expanded key (the sequence of $Nb\cdot(Nr+1)$ bytes generated from the initial *Nk* bytes of the key *K*) and *state* is the state matrix (consisting of 4 rows and 32 bits in each row). Operations performed during the encryption include:

- Assignment of the *IN* vector to the *state* matrix and the *state* matrix to the OUT vector. The input bytes vector is transformed into the array, according to (1):

$$state[row, column] = IN[row + 4 \cdot column] \quad (1)$$

  The output bytes vector is obtained from the state according to (2):

$$OUT[row + 4 \cdot column] = state[row, column] \quad (2)$$

- AddRoundKey – the xor operation performed on every column of the *state* array, using the particular bytes of the expanded key (3):

$$state[*, column] = state[*, column] \oplus W[i \cdot Nb + column] \quad (3)$$

- SubBytes – operation of exchanging the elements of the *state* matrix into the new values, using the structure called s-box. The latter is the array 16x16, containing the substitution values. The substitution is performed according to (4):

$$b_i = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \quad (4)$$

  where c_i is the bit of the byte $\{01100011\}$.
- ShiftRows – operation of rotating right the rows of the *state* array at *i* positions, where *i*=(0,1,2,3). This means that the first row is not changed, and the fourth row has the form: $\{a_{3,3}\, a_{3,0}\, a_{3,1}\, a_{3,2}\}$.
- MixColumns – in this operation every column of the state array is transformed using the equation (5):

$$\begin{bmatrix} state_{0,column} \\ state_{1,column} \\ state_{2,column} \\ state_{3,column} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} state_{0,column} \\ state_{1,column} \\ state_{2,column} \\ state_{3,column} \end{bmatrix} \quad (5)$$

2.2. AES decryption scheme

The decryption algorithm is similar to the encryption, but contains the inverse array operations, which are also in different order than in the encryption scheme. It will be used by the data processing server do decrypt the incoming measurement data and by the DAQ nodes to decrypt the instructions from the control nodes or the server. The generic decryption procedure [1] is presented in Fig. 3.

The additional operations are as follows:

- InvShiftRows – operation of rotating left the rows of the *state* array at *i* positions, where *i*=(0,1,2,3). Again, the first row is not changed, and the fourth row has the form: $\{a_{3,1}\, a_{3,2}\, a_{3,3}\, a_{3,0}\}$.

- InvSubBytes – the inverse operation of exchanging the elements of the *state* matrix into the new values, using the inverted s-box.
- InvMixColumns – in this operation every column of the state array is transformed using the equation (6):

$$\begin{bmatrix} state_{0,column} \\ state_{1,column} \\ state_{2,column} \\ state_{3,column} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \cdot \begin{bmatrix} state_{0,column} \\ state_{1,column} \\ state_{2,column} \\ state_{3,column} \end{bmatrix} \quad (6)$$

```
AESinvcipher(IN(4·Nb), OUT(4·Nb), W[Nb·(Nr+1)])
begin
    state[4,Nb] = IN
    AddRoundKey(state, W[Nr·Nb, (Nr+1)·Nb-1])
    for i = Nr-1 to 1
        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state, W[i·Nb, (i+1)·Nb-1])
        InvMixColumns(state)
    end
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, W[0, Nb-1])
    OUT = state
end
```

Fig. 3. Pseudocode of the AES decryption [1].

2.2. Expanded key generation scheme

The last element of the AES scheme is the expansion of the key *W*. It takes the initial cipher key *K* and generates from it *Nb·(Nr+1)* bytes, further used in the algorithms presented in Fig. 2 and 3.

```
AESkeyexpansion(K(4·Nk), W[Nb·(Nr+1)], Nk)
begin
    i = 0
    while (i < Nk)
        W[i] = [K[4·i] K[4·i+1] K[4·i+2] K[4·i+3]]
        i = i + 1
    end

    i = Nk
    while (i < Nb·(Nr+1))
        temp = W[i-1]
        if (mod(i, Nk)==0)
          temp = xor(SubWord(RotWord(temp)),Rcon[i/Nk])
        elseif (Nk > 6 & mod(i, Nk) == 4)
            temp = SubWord(temp)
        end
        W[i] = xor(W[i-Nk], temp)
        i = I + 1
    end
end
```

Fig. 4. Pseudocode of the key expansion [1].

The algorithm is performed before the encryption or decryption takes place. Its main operations are:

- SubWord – the operation similar to SubBytes, as it transforms the input four bytes into the output bytes using the s-box.
- RotWord – the operation of a cyclic permutation, changing the position of the bytes in the sequence, according to the following schema: $\{a_0\, a_1\, a_2\, a_3\} \rightarrow \{a_1\, a_2\, a_3\, a_0\}$.
- Rcon – a constant word array, that has the following form: $\{x^{i-1}\ \{00\}\ \{00\}\ \{00\}\}$, where x^{i-1} is the power of *{02}* in the field $GF(2^8)$

The detailed information about all the algorithms can be found in [1].

3. THE AES ALGORITHM MODIFICATIONS

The algorithms presented in section 2 can be used in any node of the distributed system. However, to take advantage of the multiprocessing units present in the nodes responsible for the data processing, their modifications must be proposed. According to the parallel computations paradigms [3], the independent parts of the algorithms must be identified and then prepared to work in separate threads. One of two approaches should be applied here. Both are based on the observation that the algorithms from section 2 perform independent matrix row or column transformations, which can be performed concurrently. Section 3.1 presents the overall approach to the simultaneous block encryption and decryption assuming that multiple processor cores are available (so each core processes its own block). In section 3.2 additional parallelism of the algorithms operations are presented. The efficiency of the latter solution is limited to the size of the matrices and works best with four cores.

3.1. Concurrent block encryption and decryption

The first modification of the original algorithm does not require any changes in the algorithm itself. The procedure involves dividing the input, plaintext bytes into blocks that can be encrypted and decrypted independently. This way multiple blocks can be processed simultaneously. The procedure is multiplied and every copy processes every *i*-th block, according to (6):

$$AES_j = f(IN_j) \qquad j = \mathrm{mod}(i,4) \quad (6)$$

where $j = \{1,\ldots,n\}$, n is the number of the copies of the AES procedure (performing n simultaneous blocks processing) and $i=\{1,\ldots,m\}$, m is the number of the plain text blocks. The first operation here is separating the plain text or cipher blocks into independent streams and then applying the AES encryption or decryption procedures.

The expected increase of the computational efficiency depends on the number of the applied copies of the procedure. Implementation of the modification in the programming language requires multi-threading mechanism. The example of the encryption using the multiple copies of the encrypting procedure is in Fig. 5.

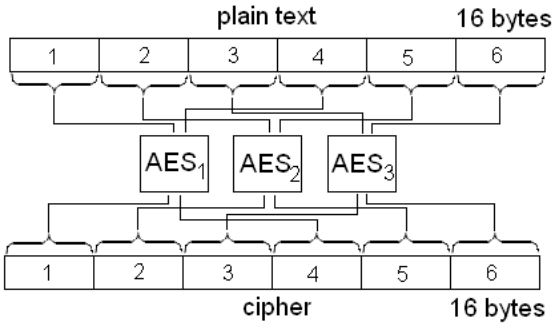

Fig. 5. Exemplary encryption using concurrent AES procedures for n=3 and m=6.

This modification is especially useful in the data processing server, which obtains multiple data from many DAQ nodes. Also, large amount of data sent from one node (for example, long waveforms – of thousands of samples) must be divided into blocks.

3.2. Modification inside the algorithm

The modification presented in section 3.1 is simple and requires little toil from the designer of the measurement system. However, a more efficient approach may be the modification made inside the algorithm to make every subprocedure inside the AES schedule to be able to run on the independent processor cores. This might also be necessary when the plain text or cipher blocks are processed in relation to the values obtained in the previous iterations. To modify the original algorithm, the analysis of the scheme presented in Fig. 2, 3 and 4 was conducted. The general solution is presented on the exemplary encryption scheme. The decryption and key expansion algorithms are performed the same way. Note that the parallelism of the whole scheme requires firstly row, then column operations.

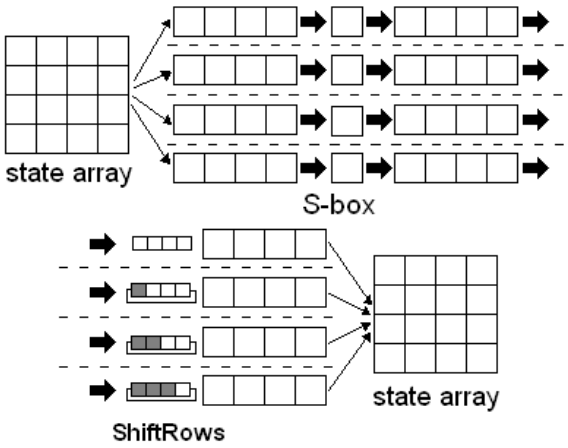

Fig. 6. Concurrent SubBytes and ShiftRows operations.

Firstly, the steps of the algorithms that are in a sequence can not be modified to be run simultaneously, as their results depend on the results obtained from the preceding operations. Therefore, the steps inside the "for" loop (SubBytes, ShiftRows, MixColumns and AddRoundKey) must be processed sequentially, and no concurrent execution is possible. However, the analysis of the operations inside of these steps shows that the matrix operations can be broken into independent parts.

The SubBytes and ShiftRows are the operations that transform the individual rows in the *state* array – each row can be processed separately. Moreover, as these operations are put one after another in the algorithm, there is no need to treat them as the separate subprocedures. They can be used inside one program function. The result of the optimization is presented in Fig. 6.

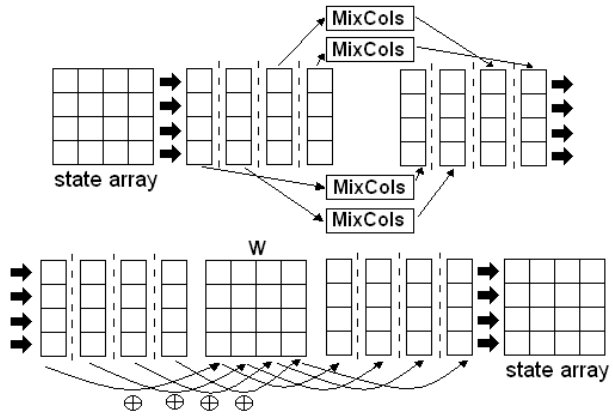

Fig. 7. Concurrent MixColumns and AddRoundKey operations.

Similarly, MixColumns and AddRoundKey can be decomposed into the simultaneously processed parts, but this time the latter operate on columns. The solution is presented in Fig. 7.

4. IMPLEMENTATION OF THE ALGORITHM

The proposed modifications were implemented in the exemplary measurement system, where the presented operations could be tested. It consisted of two nodes – DAQ and processing, which justifies application of the encryption and decryption schemes. The algorithm was implemented in the LabVIEW programming environment. To run the parallel tasks under the RT system, RT module was used. Because the cryptographic algorithms for LabVIEW are currently available thanks to the Crypto-G library [4], the AES implementation from the latter was exploited as the reference application. To ensure the parallel computations, the algorithm implementation was modified to run under the RT module. The abilities of the latter of running simultaneously computation-demanding tasks were verified during the previous research [5]. In the designed software the operations must be divided into separate threads (to reflect the concurrent data processing – see section 3). Two approaches to implement the latter were tried. The first one uses the timed sequences [6], i.e. programming structures allowing determination of the order of the particular functions execution and assign the processor cores to these tasks. In every subprocedure presented in section 3, the *state* array was decomposed into the independent parts, and then inserted into the timed sequence, assigned to the particular core. Finally, the independent rows or columns were put together, restoring the array. Note that using the timed

structures (designed especially for the RT operating system (RTOS)) makes the AES procedure runnable only under the latter. The example of the algorithm implementation using the timed sequence is in Fig. 8.

The second approach is based on the ability of the RTOS to automatically assign the particular tasks to the processor's cores. This time the array operations are simply divided into the separate data streams, without any explicit core assignment. The example of the code without the timed structures is presented in Fig. 9. The main advantage of such implementation is the ability to run the functions both under the general purpose operating system (GPOS) and RTOS.

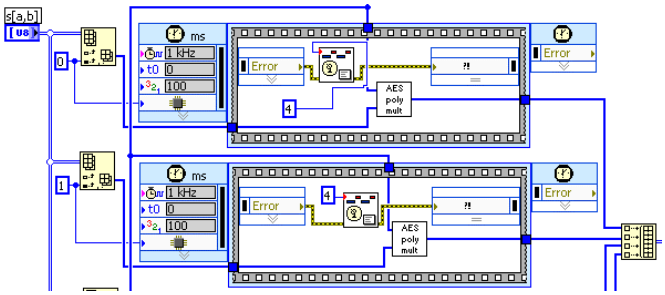

Fig. 8. Fragment of the MixColumns procedure using the timed sequence.

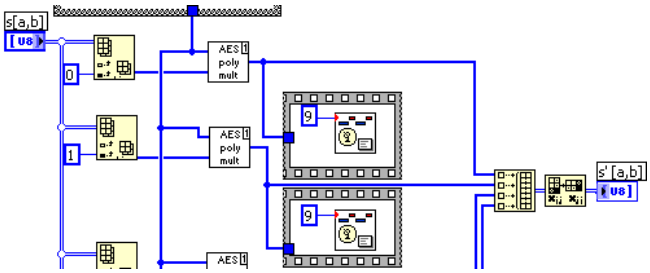

Fig. 9. Fragment of the MixColumns procedure without the timed structures.

During tests both implementations were verified with respect to the correctness of the encryption and decryption, and the speed of computations. To obtain the latter the measurement mechanisms of the RT module were exploited, i.e. timestamp measurements with the microsecond accuracy. The first one is used to take the time of the encryption and decryption computations. The second is used to observe, how the processor's cores are used to perform the measurement of application's computations.

To test the algorithm the following laboratory configuration was assembled (similar to [7]):

- RT target – a computer run under the RTOS and executing the AES algorithm implemented in the LabVIEW environment. It is equipped with the Intel Core2Quad processor, 2 GB of RAM, 160 GB hard disk drive with FAT32 partition and the network card accepted by the system.
- control node – a computer run under Windows XP operating system, controlling the RT node. It is used to design the software part of the virtual instrument and deploy it on the RT node. It is equipped with the Athlon XP 2800+ processor, 2 GB of RAM and standard network card. The version of the LabVIEW environment used to design and implement the algorithm was 8.5 (the first one supporting the multi-core programming techniques under the RTOS).
- Networking infrastructure – used to ensure the communication between the control node and RT target. Both computers must be equipped with the network cards , which are also connected to the switch.

The laboratory test stand is presented in Fig. 10.

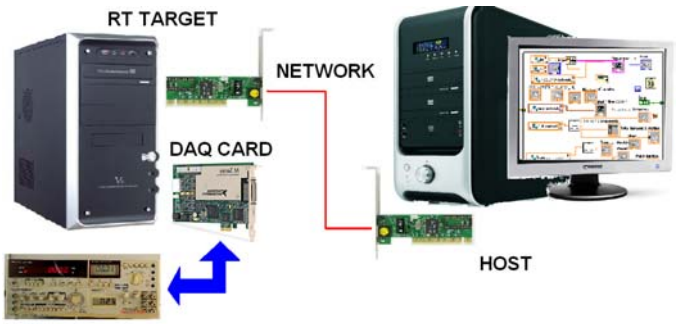

Fig. 10. Laboratory test stand.

5. TESTS AND MEASUREMENTS

All versions of the algorithm were verified using the standard test vectors, including 16-byte plain texts and 16- 24- and 32-byte long keys. The tests included both encryption and decryption to compare the times of both operations. The example of the time analysis of processing one 16-byte word using all types of the keys is presented in Table 1. The algorithm version with the implicit core assignment - ICA (see Fig. 8) is the fastest one, the version with explicit core assignment (ECA) requires too much overhead for the timed structures to be effective against the single block. However, for larger data structures it may be much faster. The encryption and decryption operations have similar time of execution, because both consist of identical number of analogous matrix operations.

Table 1. Speed of different versions of the AES encryption and decryption for the single plain text block and different key lengths.

Key length	Basic	ICA	ECA
AES-128	75,35 ms	45,78 ms	118,74 ms
AES-192	96,74 ms	51,59 ms	150,36 ms
AES-256	118,65 ms	57,35 ms	181,55 ms

The reference algorithm from the Crypto-G library worked correctly under GPOS, but did not work under RTOS. On the other hand, the algorithm designed for the RTOS using the time sequences did not run well under the GPOS. The version of the algorithm designed without the RT programming elements worked under both OS.

Another experiment consisted in the block encryption and decryption of the whole waveform gathered in the node (see section 3.1). This time the encryptions mode must be used. In the simplest case, ECB, each identical sample would be encrypted the same way, producing the same

cipher, as each block is encrypted separately here. If periodical signals are measured, the ECB mode compromises the cipher without the knowledge of the keys. Therefore the more sophisticated mode, CBC is used. Here the cipher from the previous iteration is mixed with the next plaintext block (using the XOR operation) and then put to the encryption procedure. The sequence of blocks is the sequence of measured values of the waveform. Each sample has a double representation, i.e. requires 32 bits. AES works with input sequences of four 32-bit long words, therefore sequences of four words must be put as the input of the AES scheme. The method was tested on the RT target, which received waveforms of different length. For encryption all three key lengths were used. Before the data is processed, the conversion between the numerical values and their binary representation must be performed. The samples vector was also decomposed into subvectors, each of which was encrypted by the separate core. The reciprocal operation is required to be performed on the other side of the transmission line, when the data must be decrypted. Results of the encryption/decryption scheme for the exemplary 1024-samples long waveform are in Table 2.

Table 2. Speed of different versions of the algorithm for the 1024 samples waveform and different key lengths.

Key length	Basic	ICA	ECA
AES-128	9,83 s	4,74 s	16,21 s
AES-192	14,84 s	8,03 s	21,99 s
AES-256	19,98 s	11,21 s	28,88 s

The relation between the particular versions of the AES scheme is similar to the one from Table 1. Again, the implicit core assignment is the most effective. To successfully transmit all samples between the nodes in the distributed system, the overall time will be required:

$$t_{ovr} = t_{conv} + t_{enc} + t_{net} + t_{dec} + t_{conv} \qquad (7)$$

where t_{ovr} is the overall time of sending the data, t_{conv} is the time of converting the numbers to the byte arrays (1024 total conversions) and dividing the samples vector into four concurrent vectors, t_{enc} is the time of the whole vector encryption, t_{net} is the time required for successful network transmission, t_{dec} is the decryption time of the whole samples vector. The largest amount of time is required for encryption and decryption. In the "Basic" implementation (where no multithreading for the matrix operations is involved), each operation lasts for about 50 ms. The optimal implementation requires about 30 ms. Note that the times of the overall waveform encryption are 64 times the single block encryption/decryption time, because the 1024 samples vector is decomposed into four 256-element subvectors. Each four subsequent elements of the particular subvector are the input of the AES scheme (to form 128-bit sequence).

The influence of the waveform length on the encryption and decryption efficiency is presented in Fig. 11. The most efficient is the ICA version of the algorithm.

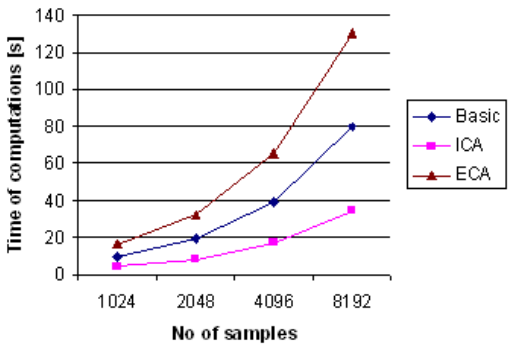

Fig. 11. Time of the AES scheme execution for different lengths of samples vectors.

6. CONCLUSIONS

The implemented AES algorithm works well under the RTOS and can be used in both the measurement server, gathering the encrypted data from the nodes of the system, and DAQ node equipped with the multi-core processor (such as PXI-8110 [8]). The improved implementation of the algorithm may be useful in the future reliable measurement systems, where not only the efficiency and determinism will be important, but also safety of transmitted data. The time analysis of the presented algorithm shows that the RT version is more efficient than the straightforward version from the Crypto-G library. This implies that the similar approaches to implement additional cryptographic algorithms should be made in the future. The power of the hardware responsible for the algorithm realization strongly affects the efficiency of encryption/decryption scheme. Therefore the designer must assess computational requirements of his system and prepare the computers capable of fulfilling them. Also, the aim of the upcoming research may be the full cryptographic library suited for the RT systems.

REFERENCES

[1] "Announcing the Advanced Encryption Standard (AES)," Available: www.nist.gov

[2] T. Good, M. Benaissa, "AES on FPGA: from the fastest to the smallest," Proceedings CHES, Edinburgh, UK. Aug. 29 -Sept. 1, 2005, pp. 427-440.

[3] H. Kasahara, S. Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," IEEE Trans. Comput., Vol.c-33, No.11,pp. 1023-1029,Nov.1984.

[4] "Vartor Crypto-G library," available at http://www.vartortech.com/cryptog.html

[5] W. Winiecki and P. Bilski, "Multi-Core Programming Approach in the Real-Time Virtual Instrumentation", Proceedings IEEE I2MTC, Victoria, British Columbia, Canada, 12-15 May, 2008, pp. 1031-1036.

[6] "Multicore Programming with LabVIEW Technical Resource Guide," available at: ftp://ftp.ni.com/evaluation/labview/ekit/multicore_programming_resource_guide.pdf

[7] P. Bilski, W. Winiecki, "Distributed Real-Time Measurement System Using Time-Triggered Network Approach," International Journal of Computing, 2008, Vol. 7, pp. 22-29.

[8] "National Instruments Announces PXI-8110 3U Quad-Core Embedded Controller for PXI System," available at: http://embeddedsystemnews.com