THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 14, Number 2/2013, pp. 177–183

PERFORMANCE EVALUATION OF AES ALGORITHM UNDER LINUX OPERATING SYSTEM

Boris DAMJANOVIĆ, Dejan SIMIĆ

University of Belgrade, Faculty of Organizational Sciences, 11000 Belgrade, Serbia,
E-mail: damjanovic@koledzprijedor.org, dsimic@fon.bg.ac.rs

Today, eleven years after being standardized, Advanced Encryption Standard (AES) is de facto a world standard for data encryption. After being published in the FIPS-197 document for the first time, continuous efforts have been made towards the algorithm improvement. In this paper we have compared the software performances of the AES algorithm implementations into cryptographic packages Oracle/Sun, Bouncy Castle, FlexiProvider and Cryptix, which all use Java Cryptography Extension (JCE), as well as their adaptability to three different versions of the virtual machines - Java SE 5.0u22 for Linux x86, Java SE 6u31 for Linux x86 and Java SE 7u2 for Linux x86. As a tool of the results presentation and interpretation we have used regression analysis. The end results within the chosen environment indicate that the Oracle/Sun implementation the AES algorithm is superior under the versions 5 and 6 JDK, but the Bouncy Castle implementation takes over the lead in the case of the version 7 JDK. Surprisingly, the AES implementation of the aging Cryptix package achieved very good results in our tests with regards to its speed.

Key words: cryptography, algorithms, AES, performance.

1. INTRODUCTION

Nowadays, eleven years after being standardized, Advanced Encryption Standard (AES) [14, 5, 4, 13] has become de facto a world standard for data encryption. After its five-year standardization process had been completed, AES was published in the USA by the US NIST (National Institute of Standards and Technology) in the FIPS-197 document in November 2001. The evaluation process started in January 1997 by announcing a public competition for the best algorithm. The first convention dedicated to this algorithm introduced fifteen competitors. After the second and the third conference held in April and August 1999 respectively, only five candidates were still to compete: MARS, RC6, Rijndael, Serpent and Twofish. In October 2000 the algorithm that eventually won the competition was Rijndael. From that time on, continuous efforts that were launched in an attempt to improve the algorithm performances have been progressing in two different directions. Concerning the software related attempts, the algorithm optimization has been based upon Dr. Gladman's [9] ideas as well as those based upon Bertony's [1] work. On the other side, hardware acceleration reached its peak at the moment when Intel and AMD processors were introduced supporting AES New Instructions (AES NI) instruction set. Additionally, VIA x86, AMD Geode, and Marvell Kirkwood processors use driver based accelerations.

When opting for a certain AES algorithm implementation, it is possible to select different criteria such as: performance, length of key, legislation, ethical and regulatory compliance requirements. However, in this paper we particularly focus on the performance criterion in different Java virtual machines environment under Linux operating system. We will make a comparison between the software performances of the cryptographic packages as of Oracle/Sun, Bouncy Castle, FlexiProvider and Cryptix that all use Java Cryptography Extension (JCE) without AES NI set of the instructions, and their adaptability to three different versions of virtual machines – Java SE 5.0u22 for Linux x86, Java SE 6u31 for Linux 86 and Java SE 7u2 for Linux x86. Regression analysis [12] will be used for the presentation of the results and their interpretation.

2. CRYPTOGRAPHIC PACKAGES INCLUDED IN TESTS

Java cryptography architecture [10, 15] is a framework providing access to cryptographic functions of the Java platform. It includes APIs for the large number of the cryptographic services, including MD algorithms, digital signature algorithms, symmetric and asymmetric cryptography, (pseudo-)random number generators, etc. The very essence of the JCA architecture stems from the idea of the existence of the security providers. In practice, a provider is a collection of algorithm classes headed up by a java.security.Provider object [11].

Since the 1.1 version, each JDK comes with the default provider name SUN. But, the former USA export legislation used to control the cryptographic methods published worldwide.

So, the SUN provider contained only cryptographic mechanisms which could not encrypt data directly. That is why a separate API was created, providing all applications with ability for data encryption and decryption. Java Cryptography Extension (JCE) was developed as a special optional package, which was available in a form of an extension in JDK versions 1.2 and 1.3. During the JDK version 1.4 development the US regulations allowed the JCE to be bundled as a part of JDK. Nowadays, to use the keys with a longer length we have to download and install Java Cryptography Extension Unlimited Strength Jurisdiction Policy Files from the Oracle's site.

JCE provides us with the implementation of encryption algorithms, key generation and key agreement as well as for Message Authentication Code (MAC) algorithms. We could say that JCE extends JCA in a way that reveals the number of different algorithm implementations. Although it was just an optional package at the beginning, JCE becomes fully integrated in the Java 2SDK 1.4 version. Despite the fact that Oracle acquired SUN in 2010, Java continues to be delivered with pre-installed and registered provider named SunJCE. Oracle's SunJCE provider implements AES with a number of modes of operation and several padding schemes.

The Cryptix cryptographic package was designed in 1995, in the age when the USA banned the export of cryptography. In those days, Cryptix was the first public cryptographic library. After the appearance of various cryptographic libraries that became publicly available, there was no need for further development of this project that eventually ended in 2005. Cryptix has the infrastructure that fits in with JCE framework.

Bouncy Castle is an open source Java and C# cryptographic package which supports over 30 symmetric and asymmetric cryptographic engines. As well as Cryptix, Bouncy Castle package can be plugged in as a provider to JCE framework. The last stable Java version (1.47) was published on 23.3.2012.

FlexiProvider is another open source product that can be plugged into any application that was built upon the JCE framework. FlexiProvider was developed by the Technische Universitat Darmstadt, Germany, and the last stable version (1.7p3) was published in November 2011.

3. TESTING METHODOLOGY

As a testing platform, we used Asus notebook with Intel (R) Core (TM) i5 450M at 2.40GHz, (without AES-NI instruction set) with 4GB RAM and WD 3200BEV external USB hard disk and with Linux Fedora 16 operating system.

Four above mentioned cryptographic packages were tested with tree versions of JDK:

- Java SE Development Kit 7u2 for Linux × 86,
- Java SE Development Kit 6u31 for Linux × 86,
- Java SE Development Kit 5u22 for Linux × 86 using default VM model.

This was a reason we created 4 applications to be used to test the implementation of AES algorithm of mentioned cryptographic packages. We will hereinafter refer to these applications as ORA for Oracle/Sun implementation, BC for Bouncy Castle implementation, Cryptix for the same implementation and Flexi for FlexiProvider implementation of AES algorithm. Source code was implemented as micro-benchmarks in the way that ignored the influence of the class initialization and hard disc. As it is in [2, 3, 7, 8] we combined 2 methodologies for Java application performance measuring. At first, in each application we measured the time of encryption for different file lengths using single VM call for each particular file. In this way, each file used in the test was processed for 5 times using every single application. After that, we put the same

source code in a loop and executed it for six times in one particular VM call. Than we disregarded the first result, that according to [2, 3] is considered to be the time required for profiling/compiling. At the end, we combined two methodologies in the way that we calculated arithmetical mean of the results achieved in the tests series mentioned above.

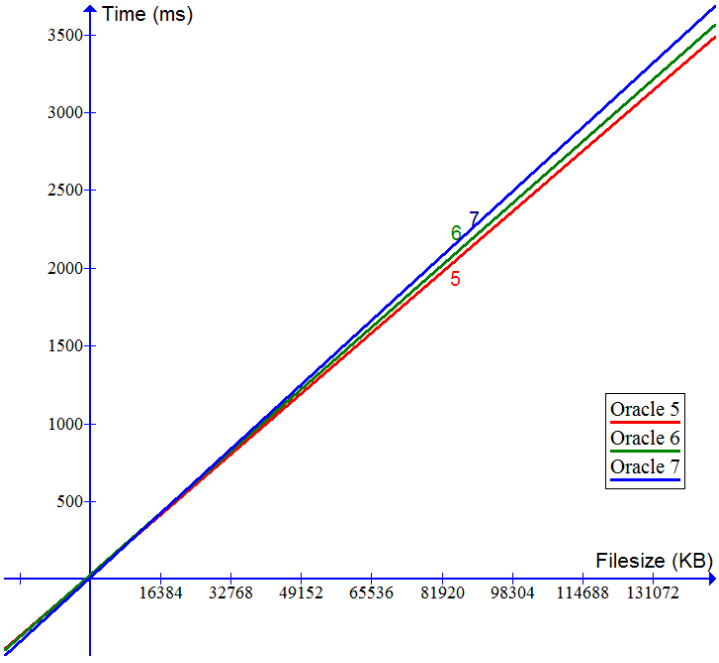
For the presentation of the results, we used the regression analysis, corresponding tables, and graphs. The regression analysis is a method used to describe the relationship between two or more variables, i.e. examines the impact of one or more variables on another variable. In the example given, we observed influence of the independent variable X (file size) on the dependent variable Y (execution time). Linear regression was used for modelling data points, with file size as the independent variable and resulting B_0 and B_1 parameters, as in:

$$\overline{Y} = \overline{B}_0 + \overline{B}_1 x \,. \tag{1}$$

At the end we used estimated standard errors to create the confidence intervals at a 95% confidence level.

4. TESTING RESULTS

Given below, we present the most interesting testing results. At first, we compared only 256 bit encryption results for each manufacturer and examined how they adapted to each tested version of JDK. The upper section of the table shows the calculated arithmetic means of 256 bit encryption while the regression analysis with calculated coefficients B0 and B1, standard error B0Err and B1Err and confidence intervals LowB0, UpB0, LowB1 and UpB1 are indicated at the bottom.

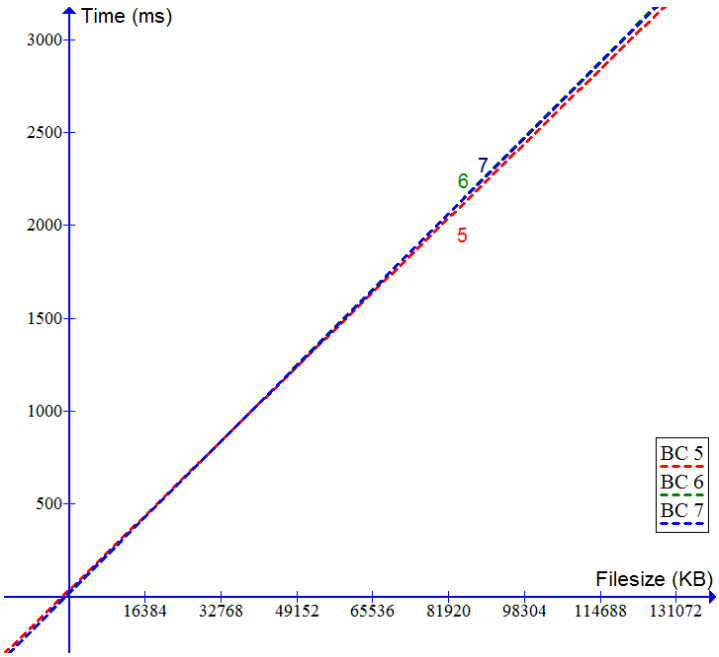


256 bit AES encryption	Oracle JDK5 (ms)	Oracle JDK6 (ms)	Oracle JDK7 (ms)
512 KB	35	35	33
4096 KB	123	127	126
8192 KB	221	231	223
16384 KB	425	425	429
32768 KB	802	829	830
65536 KB	1577	1626	1615
131072 KB	3141	3205	3331
Regression analysis			
B0	26.301	29.226	12.162
B1	0.0238	0.0243	0.0252
B0Err	2.681	2.669	11.770
B1Err	0.000	0.000	0.000
LowB0	19.408	22.364	-18.095
UpB0	33.194	36.087	42.419
LowB1	0.0236	0.0241	0.025
UpB1	0.0239	0.0244	0.025

Fig. 1 – Oracle/SUN implementation – 256 bit encryption results for JDK-5-6-7.

The Oracle/SUN implementation of AES algorithm shows certain differences within different testing environments, i.e. gradual deceleration while moving from older to newer testing environment. According to the results shown in Fig. 1, we can see that in the version 5 JDK implementation of Oracle/SUN produces the lowest coefficient B1, i.e. the smallest slope. The same source code that calls Oracle implementation of the AES algorithm which is compiled and executed under the JDK 5 shows slightly better results than the one that compiles and executes under JDK version 6. Furthermore, it shows better results than the one compiled and executed under the version 7 JDK. Altogether, Oracle implementation of the AES algorithm shows

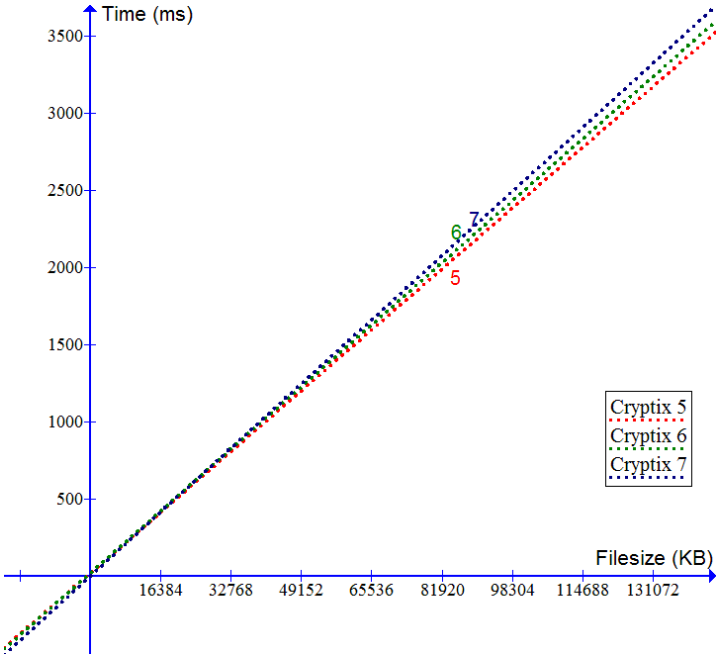
better results using older versions of JDK. In Fig. 2 we show the testing results of the Bouncy Castle implementation of the AES algorithm within different versions of JDK.



256 bit AES encryption	BC JDK5 (ms)	BC JDK6 (ms)	BC JDK7 (ms)
512 KB	45	32	33
4096 KB	135	124	126
8192 KB	240	224	228
16384 KB	433	430	429
32768 KB	837	833	840
65536 KB	1643	1642	1629
131072 KB	3237	3298	3296
Regression analysis			
B0	35.687	17.937	19.826
B1	0.0244	0.02497	0.02490
B0Err	1.893	3.438	19.826
B1Err	0.000	0.000	0.000
LowB0	30.821	9.099	4.567
UpB0	40.553	26.776	35.084
LowB1	0.0244	0.0248	0.025
UpB1	0.0245	0.0251	0.025

Fig. 2 – Bouncy Castle implementation – 256 bit encryption results for JDK-5-6-7.

Bouncy Castle implementation of the AES algorithm provides very consistent results when it comes to 256 bit encryption on different virtual machines. Some slightly faster encryption can be noted only when it comes to version 5 JDK. In Fig. 3 we present the testing results of the Cryptix implementation of the AES algorithm within different versions of JDK.

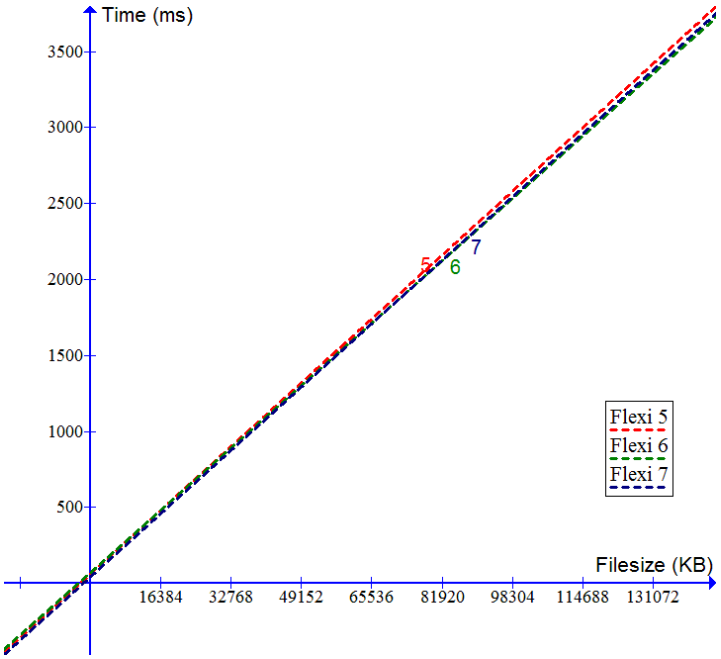


256 bit AES encryption	CR JDK5 (ms)	CR JDK6 (ms)	CR JDK7 (ms)
512 KB	33	34	28
4096 KB	123	124	120
8192 KB	223	231	219
16384 KB	419	424	420
32768 KB	812	830	825
65536 KB	1595	1626	1613
131072 KB	3171	3234	3345
Regression analysis			
B0	24.224	24.826	4.309
B1	0.0240	0.02448	0.02530
B0Err	1.142	1.813	12.960
B1Err	0.000	0.000	0.000
LowB0	21.288	20.164	-29.005
UpB0	27.160	29.487	37.623
LowB1	0.0240	0.0244	0.025
UpB1	0.0241	0.0246	0.026

Fig. 3 – Cryptix implementation – 256 bit encryption results for JDK-5-6-7.

Cryptix implementation of AES algorithm gives results which are very similar to the Oracle/Sun implementation of this algorithm. The same source code that calls Cryptix implementation of the AES algorithm which is compiled and executed under the JDK 5, shows slightly better results than the one

compiled and executed under the version 6 JDK, and it further shows better results than the one compiled and executed under the version 7 JDK. In Fig. 4, the testing results of FlexiProvider implementation of AES algorithm are presented.

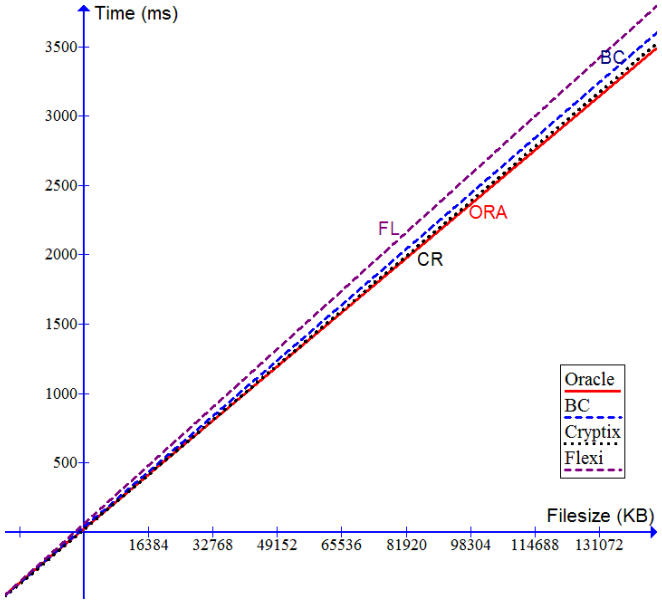


256 bit AES encryption	FL JDK5 (ms)	FL JDK6 (ms)	FL JDK7 (ms)
512 KB	68	91	64
4096 KB	171	165	155
8192 KB	263	282	255
16384 KB	486	477	460
32768 KB	916	885	876
65536 KB	1731	1709	1665
131072 KB	3415	3353	3394
Regression analysis			
B0	62.280	70.011	42.681
B1	0.0256	0.02503	0.02540
B0Err	4.795	3.249	10.918
B1Err	0.000	0.000	0.000
LowB0	49.954	61.660	14.615
UpB0	74.606	78.362	70.747
LowB1	0.0254	0.0249	0.025
UpB1	0.0258	0.0252	0.026

Fig. 4 – FlexiProvider implementation – 256 bit encryption results for JDK-5-6-7.

The FlexiProvider implementation of the AES algorithm shows very close results within all three testing environments and just a bit higher encryption speed in the case of the updated versions of JDK.

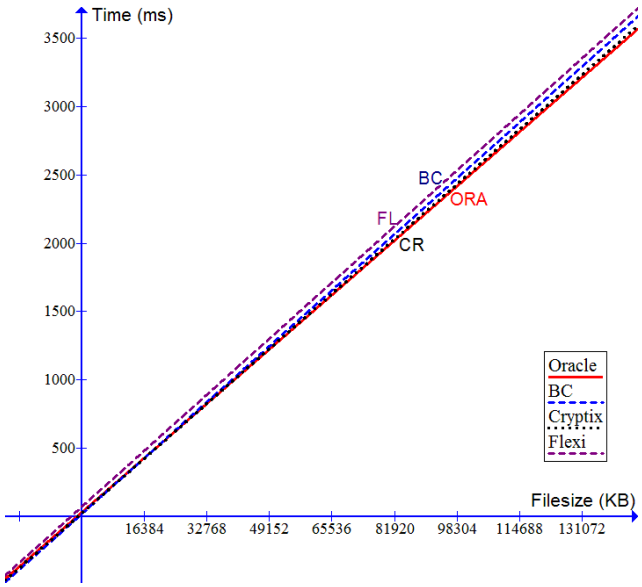
In Fig. 5 we show the 256 bit encryption results of all tested implementations in the JDK version 5.



256 bit AES encryption	Oracle (ms)	BC (ms)	Cryptix (ms)	Flexi (ms)
512 KB	35	45	33	68
4096 KB	123	135	123	171
8192 KB	221	240	223	263
16384 KB	425	433	419	486
32768 KB	802	837	812	916
65536 KB	1577	1643	1595	1731
131072 KB	3141	3237	3171	3415
Regression analysis				
B0	26.301	35.687	24.224	62.280
B1	0.0238	0.0244	0.0240	0.0256
B0Err	2.681	1.893	1.142	4.795
B1Err	0.000	0.000	0.000	0.000
LowB0	19.408	30.821	21.288	49.954
UpB0	33.194	40.553	27.160	74.606
LowB1	0.0236	0.0244	0.0240	0.0254
UpB1	0.0239	0.0245	0.0241	0.0258

Fig. 5 – 256 bit encryption results – JDK 5.

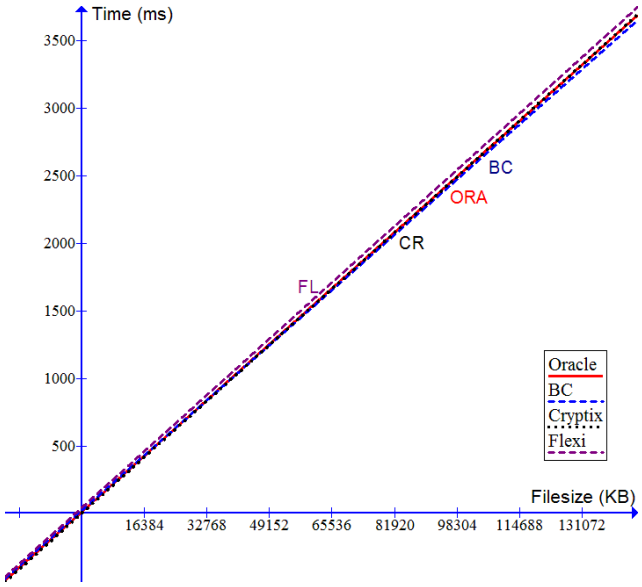
Concerning the tests taken under JDK version 5, the figures and tables show the smallest coefficient B1 (the smallest slope) which implies the highest speed of Oracle implementation of the AES algorithm. To our surprise, almost equally satisfactory results are obtained with the elderly Cryptix implementation of this algorithm. In Fig. 6 the 256 bit encryption results of all tested implementations are shown for the version 6 JDK.



256 bit AES encryption	Oracle (ms)	BC (ms)	Cryptix (ms)	Flexi (ms)
512 KB	35	32	34	91
4096 KB	127	124	124	165
8192 KB	231	224	231	282
16384 KB	425	430	424	477
32768 KB	829	833	830	885
65536 KB	1626	1642	1626	1709
131072 KB	3205	3298	3234	3353
Regression analysis				
B0	29.226	17.937	24.826	70.011
B1	0.0243	0.0250	0.0245	0.0250
B0Err	2.669	3.438	1.813	3.249
B1Err	0.000	0.000	0.000	0.000
LowB0	22.364	9.099	20.164	61.660
UpB0	36.087	26.776	29.487	78.362
LowB1	0.0241	0.0248	0.0244	0.0249
UpB1	0.0244	0.0251	0.0246	0.0252

Fig. 6 – 256 bit encryption results – JDK 6.

Testing of the different implementations in the JDK version 6 shows that the Oracle and Cryptix implementations of the AES algorithm have very similar results, as evidenced by close values of their B1 coefficients. A bit lower results are obtained with the Bouncy Castle implementation of the AES algorithm followed by FlexiProvider implementation of the same algorithm. In Fig. 7 we present the results of the encryption for all tested implementations in version 7 JDK.



256 bit AES encryption	Oracle (ms)	BC (ms)	Cryptix (ms)	Flexi (ms)
512 KB	33	33	28	64
4096 KB	126	126	120	155
8192 KB	223	228	219	255
16384 KB	429	429	420	460
32768 KB	830	840	825	876
65536 KB	1615	1629	1613	1665
131072 KB	3331	3296	3345	3394
Regression analysis				
B0	12.162	19.826	4.309	42.681
B1	0.0252	0.0249	0.0253	0.0254
B0Err	11.770	19.826	12.960	10.918
B1Err	0.000	0.000	0.000	0.000
LowB0	-18.10	4.567	-29.01	14.615
UpB0	42.419	35.084	37.623	70.747
LowB1	0.025	0.025	0.025	0.025
UpB1	0.025	0.025	0.026	0.026

Fig. 7 – 256 bit encryption results – JDK 7.

When it comes to the tests that are performed within the JDK version 7, Bouncy Castle implementation of AES algorithm produces the smallest slope of the regression curve, i.e. the best results. Less favourable results are the ones of Oracle/SUN and elderly Cryptix implementation, while FlexiProvider implementation of the AES algorithm brings about the worst results in our tests.

5. CONCLUSION

In this paper we have presented the test results of different implementations of the AES algorithm which are compiled and executed under different versions of JDK under Linux Fedora operating system. We used the simple linear regression for the calculation of the appropriate coefficients, confidence intervals and

the standard errors. For the presentation of the results we used tables with calculated arithmetic means, and the results of the regression analysis as well as the appropriate graphs with shown regression lines.

Test results can be divided into two groups. The first set of the results shows to which extent each tested implementation of the AES algorithm is adapted to a certain version of the JDK. Each particular implementation was tested using the aforementioned versions of the java virtual machines. As expected, the elderly Cryptix implementation shows somewhat better performance under the older versions of JDK. However, Oracle/Sun implementation of the AES algorithm surprisingly shows better performance under older versions of java virtual machines, while Bouncy Castle and FlexiProvider implementations of the AES algorithm show fairly consistent results for different versions of the JDK.

Cross-comparison between different implementations under versions 5 and 6 of the JDK (Java SE Development Kit 5u22 and Java SE Development Kit 6u31) proves slight, yet expected advantage of the Oracle/Sun implementation over Bouncy Castle and FlexiProvider implementations of the AES algorithm. Cryptix implementation shows amazingly favourable results, almost equal to the results of the Oracle/Sun implementation of the AES algorithm in each conducted test. As for JDK version 7 (Java SE Development Kit7u2), experiments in our testing environment show that Bouncy Castle implementation of the AES algorithm is fairly more adaptable to newer versions of JDK then others.

According to the empirical results it is possible to conclude that we will not blunder when opting for a certain AES algorithm under Linux operating system with regards to its performance. But, if the performance criterion is critical for our selection, we'd better use the Oracle, Bouncy Castle or Cryptix implementations with some older version of Java virtual machines, because they could produce slightly better results.

ACKNOWLEDGMENTS

The work presented here was partially supported by the Serbian Ministry of Science and Technological development (under project of Multimodal Biometry in Identity Management, contract no TR-32013).

REFERENCES

1. BERTONI G., BREVEGLIERI L., FRAGNETO P., MACCHETTI M., MARCHESIN S., *Efficient Software Implementation of AES on 32-Bit Platforms*, CHES 2002, **2523**, pp. 159–171, 2003.
2. BOYER B., *Robust Java benchmarking. Part 1: Issues, Understand the pitfalls of benchmarking Java code*, IBM developerWorks, 2008.
3. BOYER B., *Robust Java benchmarking. Part 1: Statistics and solutions, Introducing a ready-to-run software benchmarking framework*, IBM DeveloperWorks, 2008.
4. CID, C., MURPHY S., ROBSHAW M. *Algebraic Aspects of the Advanced Encryption Standard*, Springer Science-Business Media, LLC, 2006.
5. DAEMEN J., RIJMEN V., *The Design of Rijndael*, Springer-Verlag, Inc., 2002.
6. DAMJANOVIĆ B., SIMIĆ D., *Comparative Implementation Analysis of AES Algorithm*, Journal of Information Technology and Applications. **1**, *2*, pp. 119–126, 2011.
7. GEORGES A., et al., *Java Performance Evaluation through Rigorous Replay Compilation*, OOPSLA '07 Proceedings of the 22nd annual (ACM SIGPLAN) Conference on *Object-oriented Programming Systems and Applications*, **43**, pp. 367–384, 2008.
8. GEORGES A. et al., *Statistically Rigorous Java Performance Evaluation*, OOPSLA '07 Proceedings of the 22nd annual (ACM SIGPLAN) Conference on *Object-oriented Programming Systems and Applications*, **42**, pp. 57–76, 2007.
9. GLADMAN B., *A Specification for Rijndael, the AES Algorithm*, available at: http://gladman.plushost.co.uk/oldsite/cryptography_technology/rijndael/aes.spec.v316.pdf (Accessed: May 2012).
10. HOOK D., *Beginning Cryptography with Java*, Wrox Press, 2005.
11. KNUDSEN J.B., *Java Cryptography*, O'Reilly Media, 1998.
12. LOVRIC M. et al., *Statistical Analysis – Methods and Application*, Faculty of Economics, Banja Luka, BH, 2008.
13. MINIER M., *A Three Rounds Property of the AES*, Advanced Encryption Standard AES, 4th International Conference, Bonn, Germany, 2004, Springer-Verlag, 2005, pp. 16–26.
14. NIST, FIPS 197, *Specification for the Advanced Encryption Standard (AES)*, 2001; available at: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (Accessed: May 2012).
15. WEISS, J., *Java cryptography extensions: practical guide for programmers*, Elsevier, Morgan Kaufmann, 2004.

Received June 8, 2013