ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20150928110055
Original scientific paper

An Application of Partial Homomorphic Encryption in Computer System with Limited Resources

Dejan SAVIĆ, Mladen TRIKOŠ, Mladen VEINOVIĆ, Dejan SIMIĆ

Abstract: Mathematical calculation over ciphertext during homomorphic encryption makes result of the calculation in a form of ciphertext. Obtained result can be decrypted with an appropriate key for further usage. In the era of cloud computing, this feature can be used to solve problems of storing and processing sensitive data in the cloud. Also, use of homomorphic encryption keeps both, data and results of processing, highly secured at the same time. Considering the overwhelming presence of computers with limited resources, the presence of the Internet and the ultimate practical need for use of cloud and its resources, in this paper we explored and presented the characteristics of a practical use of partial homomorphic encryption and its processing of ciphertext in the cloud on computer system with limited resources.

Keywords: cloud computing; homomorphic encryption; limited resources; partial homomorphic encryption

1 INTRODUCTION

Cloud computing is defined as a type of computing that relies on sharing computing resources rather than having local servers or personal devices. It provides users and enterprises with various capabilities to store and process their data in third-party data centres. Therefore, cloud computing has numerous advantages like almost unlimited processor power and storage capacity, relatively easy process of backup and restore of data, automatic software integration, quick deployment to cost efficient, and fine control of needed resource [1-3].

It can be said that cloud computing is everything one enterprise needs for successful IT support. However, there are some risks involved in cloud computing [4, 5]. One of the major risk issues is cloud computing security followed by application readiness and cost.

Cloud computing security is a complex combination of computer security, network security and more broadly information security [6, 7]. Cloud security includes a broad set of technologies, applied policies and many controls deployed to protect data, applications and the associated infrastructure of cloud computing.

Because of such complexity of cloud technology, there are many security risks involved in it.

Cloud computing brings some specific security vulnerability vectors like virtual machine level attacks, cloud provider vulnerabilities, phishing of cloud provider, expanded network attack surface threat, authentication and authorization in a cloud, forensic in a cloud, etc. Also, there is a third party control where security risks originate due to diligence, audit ability, contractual obligations, cloud provider espionage, data lock-in, transitive nature of process etc. When a third party holds the data, there is also a potential lack of control over it. Furthermore, the legal implication of data and applications being held by a third party are complex and sometimes not well understood.

There are numerous techniques and technologies, which can be used to solve the security risks listed above. One of the most trustworthy techniques for cloud data security is encryption. Encryption is a process of encoding messages or information in such a way that only authorized parties can read them, but it comes with one major problem - loss of computation over data. This issue can be avoided by using homomorphic encryption.

The security of ciphertext stored on the cloud directly depends on applied crypto algorithm. Homomorphic cryptosystem allows calculations over its ciphertext which, as a result, have a new ciphertext that can be decrypted with a proper private key. If this property is used in cloud computing, it can solve some of security issues. This is the main reason why academic society is interested in a usage of homomorphic cryptosystems in the cloud [8-11].

Unfortunately, at this moment, there is no homomorphic scheme which can be practically applied. On the other hand, there is cryptographic scheme which can solve some of full homomorphic issues but with certain limitations. They are called partially homomorphic scheme and they can implement some of the computations over ciphertext.

Nowadays, we are surrounded by small computer systems which are connected to the Internet. These small computer systems can be found in our pockets (smartphones), cars (GPS), on our hands (fitness trackers, smart watches), home appliances and etc. Although these small computer systems have limited resources, if we used them as a terminal or tiny client powered by cloud computing, they can be a lot of help to us in everyday activity.

The goal of our work is to investigate if it is possible to use such a tiny client in a partially homomorphic scheme.

In our work, we have explored property of a partially homomorphic scheme (Section 2) and have done practical implementation of it on the computer systems with limited resources (Section 3). We have also applied it to one experimental problem and have come up with usable solutions.

We have found that it is possible to do crypting and decrypting of a reasonable amount of data on a computer system with limited resources and, by the help of cloud, do some calculation over that ciphertext for some management needs in solving practical problems (Section 4).

2 HOMOMORPHIC ENCRYPTION

Homomorphic encryption [12, 13] is a special type of encryption scheme, which allows specific types of

computations to be carried out on a ciphertext. As a result the computations on the ciphertext obtain a ciphertext which is the result of those operations performed on the plaintext.

In the literature, we can find a difference between fully encryption scheme where additively and multiplicatively homomorphic scheme is possible [14, 15] and a partially homomorphic scheme which is either additively or multiplicatively homomorphic, not both, such as Unpadded RSA, ElGamal, Paillier or Goldwasser-Micali [16-19] schemes.

The additive homomorphic property of a homomorphic cryptosystem can be mathematically shown as an Eq. (1).

$$Enc(a) \times Enc(b) = Enc(a + b) \tag{1}$$

where a and b are two plaintext message blocks, and Enc is an encryption function that takes a plaintext message block with an encryption key and returns the ciphertext block.

In the above Eq. (1), + operates on the plaintext is the× operates on the ciphertext. An example of such an encryption scheme is the Paillier crypto system. On the other hand, ElGamal encryption scheme is an example of multiplicative homomorphic which can be shown in Eq. (2).

$$Enc(a) \times Enc(b) = Enc(a * b) \tag{2}$$

where a, b, Enc and × share the same meanings with the formula above, and operation $*$ is multiple operation on the plaintext.

3 PROBLEM DESCRIPTION AND TESTING ENVIRONMENT

Suppose that we have some system, biological or technical, and we do some measurements over it. Sometimes the obtained measurements can be very sensitive information (it can be patient health status, some vital parameters of the system, system reading etc.). Usually, these readings must be safely stored and available for some future usage. The cloud is generally a very reasonable place for storing such data when you have a tiny client. Of course, it is desirable that, besides data storage, you can do some computation on the data.

Probably the safest way to store data is to encrypt them before sending to the cloud. In that case, the data will be safe against a malicious attacker or curious provider and possible intruder. If we choose to use some of the homomorphic encryptions than we can do some computation on a data store on the cloud and get some results in a very safe way.

3.1 System Description

Fig. 1 shows a brief outline of our system. The system has sensors which read some status of an object. Our tiny client reads data from sensors and does some preparation of them by interpreting them in the form appropriate for storing. Using a private key and a crypto algorithm, the tiny client does encryption of those data and sends them over Internet or private network to Cloud provider 1 ($CP1$) as ciphertext. There is an appropriate application on the $CP1$ which stores those data in some database together with some metadata which is in the form of open text.

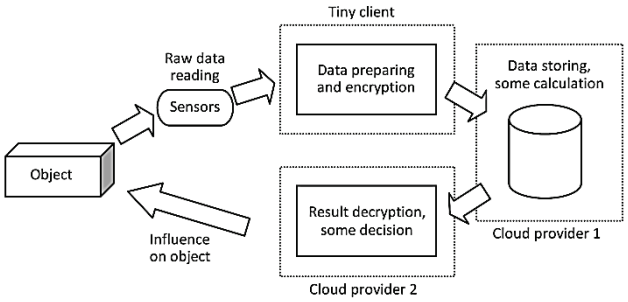

Figure 1 Basic outline of system

Data are stored on the $CP1$ and are safe according to the security of cryptosystem in use.

Very often, there is a need to correct some of a system's parameters according to its state, read data and desirable condition. Therefore, we needed to do some calculation over stored data to get some input parameters for system correction.

We let Cloud provider 2 ($CP2$) do some correction on the system. To do that $CP2$ needs correction data which is supplied by $CP1$. $CP1$ can do some periodically homomorphic calculation on the data and sends the result of correction function as a ciphertext to $CP2$. $CP2$ has a private key and it decrypts the result of correction function. Based on that result, $CP2$ has some influence on the system.

Because of that, *CP2* is in a special relation with *CP1*. These two must be the true opponents and without any trust in each other. This must be reached because *CP2* cannot decrypt result without knowledge of the private key.

Based on this concept, we established an environment for testing. We supposed that our system is a smart house with some sensors (open/close door, window, temperature measurement, movement detection etc.). As an end-user device, we used *Raspberry Pi* [20]. The *Raspberry Pi* is a low cost, credit-card sized computer that has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. Of course, it has a very reasonable price.

We used the *Raspberry Pi* together with Ubuntu Linux and implemented an additive version of a partially homomorphic cryptosystem. We chose it because of nature of our calculation over data, which is in our case addition.

3.2 Paillier Cryptosystem and Implementation of Testing Environment

Paillier cryptosystem is an additive version of a partially homomorphic cryptosystem. If E_{pk} is encryption function with public key pk given by (N, g) and D_{sk} is decryption function with secret key sk given by a trapdoor function λ (that is, the knowledge of the factors of N, N is

the *RSA* modulus of bit length *K* and generator $g \in Z^*_{N^2}$), then for any given *a, b* $\in Z_N$, the Paillier encryption scheme exhibits the following properties [21]:

- Homomorphic Addition Eq. (3):

$$D_{sk}\left(E_{pk}\left(a+b\right)\right) = D_{sk}\left(E_{pk}\left(a\right) * E_{pk}\left(b\right) \bmod N^2\right), \quad (3)$$

- Homomorphic Multiplication Eq. (4):

$$D_{sk}\left(E_{pk}\left(a*b\right)\right) = D_{sk}\left(E_{pk}\left(a\right) * b \bmod N^2\right), \quad (4)$$

- Semantic Security - The encryption scheme is semantically secure [22, 23] which means that in a given set of ciphertexts, an adversary cannot deduce any additional information regarding the corresponding plaintexts.

There are many extensions to the Paillier cryptosystem that have been proposed in the literature [24, 25, 26]. We used *libpaillier* by *SRI International* as a starting point. This library implemented the original Paillier cryptosystem. For mathematical calculation it uses the well-known *GNU Multiple Precision Arithmetic Library* which is well integrated into Linux OS. Paillier library and operating systems on *Raspberry Pi*, *CP1* and *CP2* we used, are under GPL and are publicly accessible.

For test purpose, we developed an application for *Raspberry Pi, CP*1 and *CP*2. On the *Raspberry Pi,* we had an application which periodically read memory state of sensors, processed data and did encryption.

In the case of Paillier cryptosystem, we needed a private key in order to do encryption. We used a smart card as a vault for storing keys.

Also we implemented a virtual file system on the *Raspberry Pi* that communicates with a smart card [27]. The smart card is accessible over smart card reader connected to *Raspberry Pi's* USB port. When the virtual file system is activated, it brings the file with private keys to the operating system.

We secured access to the smart card which brings private keys with the help of a PIN code. So, a user must enter a valid PIN code to activate virtual file system. With this method, private keys are present only in the random access memory and are never stored on *Raspberry Pi*. We also implemented a blocking mechanism to block smart card after three failed attempts of entering wrong PIN number.

As a result of *Raspberry Pi* calculation over sensors data, we got ciphertext. That ciphertext has been sent to *CP1* together with some metadata like *date*, *time* and *house ID* which are not crypt.

In our case, *CP*1 is a laptop computer with more powerful resource than *Raspberry Pi*. We connected *CP*1 with *Raspberry Pi* over a 100Mb local network.

On *CP1,* the ciphertext is stored in *MySQL* database together with metadata. Those records are stored in the cloud and are available to end user for later analysis.

Also, those data use *CP*1 application for some homomorphic calculation. As an example, we implemented a simple state function given as Eq. (5):

$$F(t) = \sum_{i=1}^{n} SS(i,t), \quad (5)$$

where *n* is a number of sensors and *SS* is their states implying that 0 value is a regular state of a sensor and some other value means some "action demand measurement".

This function is implemented in software on *CP*1 and, with some period or by *CP*2 call, it does homomorphic addition over ciphertext and sends the result to *CP*2. In order to do homomorphic addition, *CP*1 must have a public key and that key is stored in the same way as we stored it on *Raspberry Pi*, on the smart card accessible through virtual file system.

*CP*2 is in our case a laptop used by supervision center in order to detect if there is a problem in some house with respect to *houseID*, so some action can be done. Software on *CP*2 does a decryption of state result function and presents it. Also, it can be some automotive system with an influence on our system.

*CP*2 must have a private key to decrypt result which is in the form of ciphertext. We also implemented virtual file system on the *CP*2 software and put a private key on the smart card, so that only *CP*2 operator knows PIN code as it was a case with *CP*1.

However, regardless of securing a private and a public key with PIN code and storing them on a smart card which is only accessible over authentic Java Card applet which is secret, there is still possibility that if *CP*1 and *CP*2 cooperate they can steal keys, so *CP*1 and *CP*2 must be real not trusted opponents. If somehow *CP*1 gets a private key, it can decrypt our data stored on its server, or if *CP*2 gets access to the database hosted on *CP*1 server, our data are in threat.

4 RESULTS

Our main question is what characteristics such a system has. First, we measured performance of *Raspberry Pi* and *CP1* during the key generating stage. We showed statistic values obtained from series of measurements in Fig. 2. Each point in the figure is a mean value of ten measurements and we did that for all measurements shown in this work.

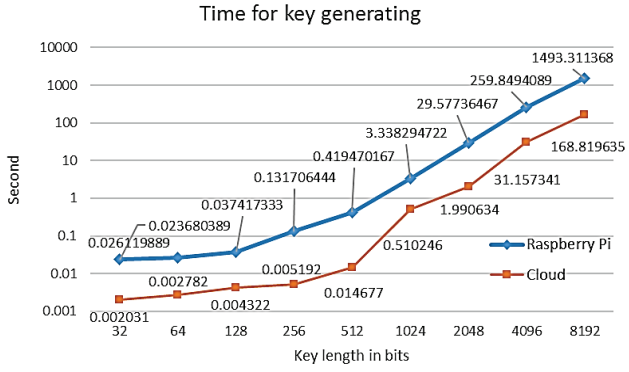

Figure 2 Time for key generating versus key length on Raspberry *Pi* and Cloud

We wanted to know how long it takes to generate a private and a public key for a different length of keys on the booth. We started from the shortest key of 32-bit

length and finished measurements with a key of 8192-bit length. We could see that key generation on the medium laptop we used as *CP*1 is about 10 times faster than on *Raspberry Pi*.

The interesting fact is that for the key length of 4096 bits and more *Raspberry Pi* needs too much time which is not acceptable for a practical application.

Furthermore, we measured how much time is necessary for the crypting operation on the *Raspberry Pi*. We measured that time for different length of keys and for different number of sensors.

Increase in number of sensors results in a linear increase of the time necessary for crypting as it can be seen in Fig. 3.

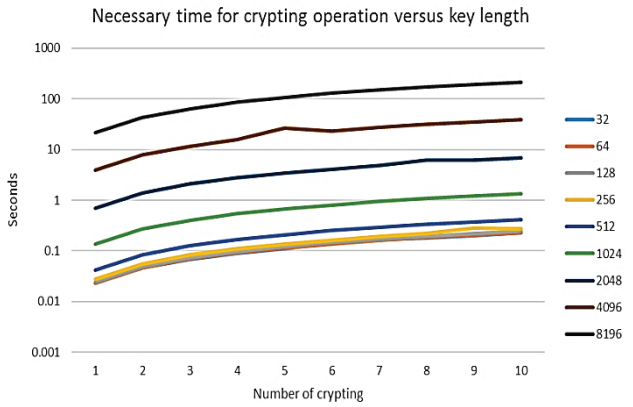

Figure 3 Necessary time for crypting operation versus key length on Raspberry *Pi*

Also, it can be seen that there is a limitation which originates from characteristics obtained from this measurement.

You can see that for the key length of 4096 bits and more *Raspberry Pi* needs much more time for crypting two or more data from sensors than it is acceptable for a practical application.

This limitation implies that it is unpractical to have keys longer than 2048 bits if we need more than three sensors and we use a computer system with limited resources.

In other words, if our supervisor system must have at least one-minute response from the system, we need to have a smaller key length or less number of sensors to do crypting successfully.

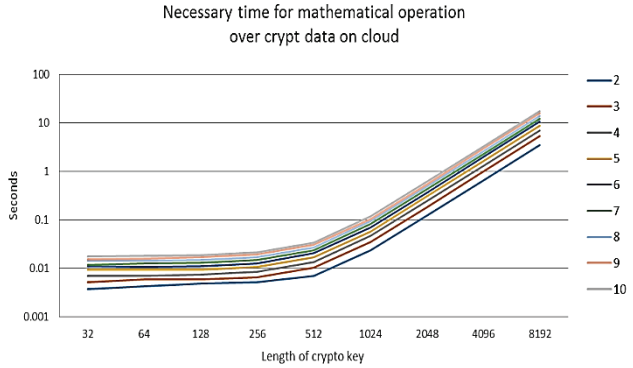

Figure 4 Necessary time for mathematical operation over crypt data on cloud

Fig. 4 shows the necessary time *CP1* needs to do homomorphic calculation over stored data which are in the form of ciphertext in dependence of a number of sensors.

As you can see, there is a certain amount of time needed for homomorphic calculation and that amount directly depends on the number of calculations.

In our case, you can see that for longer keys, especially for keys of 4096 or 8192 bits of length and for 10 sensors calculation can take few seconds. We talked about delay which can be critical if our system needs correction within intervals of few seconds.

According to our measurements, we can say that such a scheme can be applicable to the system which demands response of 0.1 seconds and upper.

As we can see, our implementation is vital for the shorter length of keys, but on the other side, it is slow for the longer length of keys. Also, we can say that it is almost unusable for key length above 4096 bits.

5 CONCLUSION

As you can see, regardless of a low computation power of *Raspberry Pi* (ours is B model with ARM 1176 processor on 700 MHz, 400 MHz SDRAM and price of US $25), it is possible to take measurements from sensors, prepare and crypt those data and send them to the cloud in the process of overseeing live system. These data are safe on the cloud because of crypting and it is possible to make some calculation on them because of the homomorphic feature of applied cryptosystem. These calculations can be used for system brief or as an input for some reaction needed to be applied to the system.

In our future work, we plan to implement Paillier cryptosystem on Android mobile phone and use some of the sensors which can be connected over Bluetooth and existing sensors on today's mobile phones. We want to send data to the cloud and see if it is real to make a prototype of an m-health system with homomorphic properties and responses on both, patient and supervision level.

ACKNOWLEDGEMENTS

Paper is a part of scientific research on Military Academy in Belgrade on VA-TT/3/18-20 project Manage of access control to protected resources of computer networks in the Ministry of Defense and the Army of Serbia on the basis of multimodal user identification.

6 REFERENCES

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., & Katz, R. (2009). Above the Clouds: A Berkeley View of Cloud Computing. UC Berkeley Reliable Adaptive Distributed Systems Laboratory White Paper. https://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf/ (18.03.2015)

[2] Rittinghouse, J. W. & Ransome, J. F. (2010). *Cloud Computing Implementation, Management and Security*. CRC Press Taylor & Francis Group, New York.

[3] Tomić, D., Orizović, D., & Car, Z. (2013). Cloud solutions for high performance computing oxymoronor realm. *Tehnički vjesnik/Technical Gazette, 20*(1), 177-182.

[4] Khalil, I. M., Khreishah, A., & Azeem, M. (2014). Cloud Computing Security: A Survey. *Computers, 3*(1), 1-35.

https://doi.org/10.3390/computers3010001
[5] Cho, Y. C. (2015). Implementation and analysis of website security mining system, applied to universities' academic networks. *Tehnički vjesnik/Technical Gazette, 22*(2), 279-287. https://doi.org/10.17559/TV-20150310110158
[6] Zissis, D. & Lekkas, D. (2012). Addressing cloud computing security issues. Future Generation Computer Systems, 28, 583-592. https://doi.org/10.1016/j.future.2010.12.006
[7] Chen, D. & Zhao, H. (2012). Data Security and Privacy Protection Issues in Cloud Computing. *International Conference on Computer Science and Electronics Engineering* / Hangzhou, China, 647-651. https://doi.org/10.1109/ICCSEE.2012.193
[8] Ahmad, I. & Khandekar, A. (2014). Homomorphic Encryption Method Applied to Cloud Computing. *International Journal of Information & Computation Technology, 15*(4), 1519-1530.
[9] Dijk, M. V. & Juels, A. (2010). On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing. *Cryptology ePrint Archive*. https://eprint.iacr.org/2010/305.pdf (25.03.2015)
[10] Brenner, M., Perl, H., Smith, M. (2012). Practical Applications of Homomorphic Encryption. *Proceedings of the 7th International Conference on Security and Cryptography* / Rome, 5-14.
[11] Moore, C., Hanley, N., McAllister, J., O'Neill, M., O'Sullivan, E., & Cao, X. (2013). Targeting FPGA DSP Slices for a Large Integer Multiplier for Integer Based FHE. Lecture Notes in Computer Science, 7862, 226-237. https://doi.org/10.1007/978-3-642-41320-9_16
[12] Rivest, R., Adleman, L., & Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. *In Foundations of Secure Computation*, 169-180.
[13] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. *In Michael Mitzenmacher, editor, STOC, ACM*, 169-178. https://doi.org/10.1145/1536414.1536440
[14] Gentry, C. (2009). A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University, http://crypto.stanford.edu/craig (01.04.2015)
[15] Dijk, M., Gentry, C., Halevi, S., & Vaikuntanathan, V. (2010). *Fully Homomorphic Encryption over the Integers*. Eurocrypt.
[16] Mollin, R. A. (2003). *RSA and Public-Key Cryptography*. CRC Press.
[17] ElGamal. T. A (1984). Public key cryptosystem and a signature scheme based on discrete logarithms. *In Proceedings of the Annual Cryptology Conference - Advances in Cryptology (CRYPTO)* / Paris, 10-18.
[18] Paillier, P. (1999). *Public key cryptosystems based on composite degree residuosity classes*. Eurocrypt, Springer-Verlag. https://doi.org/10.1007/3-540-48910-X_16
[19] Goldwasser, S. & Micali, S. (1984). Probabilistic encryption. *Journal of computer and system sciences, 28*(2), 270-299. https://doi.org/10.1016/0022-0000(84)90070-9
[20] Raspberry Pi Foundation: http://www.raspberrypi.org/ (18.04.2017)
[21] Samanthula, B. K., Elmehdwi, Y., & Jiang, W. (2014). k-Nearest Neighbor Classification over Semantically Secure Encrypted Relational Data, *Technical Report*. Department of Computer Science, Missouri S&T.
[22] Goldreich, O. (2004). *The Foundations of Cryptography. Encryption Schemes* / Cambridge, England: Cambridge University Press, 373-470. https://doi.org/10.1017/CBO9780511721656
[23] Goldwasser, S., Micali, S., & Rackoff, C. (1989). The knowledge complexity of interactive proof systems. *SIAM Journal of Computing*, 18, 186-208. https://doi.org/10.1137/0218012
[24] Damgard, I. & Jurik, M. (2001). A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. *In Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography* / Springer-Verlag, 119-136.
[25] Damgard, I. & Jurik, M. (2003). A length-flexible threshold cryptosystem with applications. *In Proceedings of the Australasian conference on Information security and privacy* / Springer-Verlag, 350-364. https://doi.org/10.1007/3-540-45067-X_30
[26] Fouque, P. A., Poupard, G., & Stern, J. (2001). Sharing decryption in the context of voting or lotteries. *In Proceedings of the 4th International Conference on Financial Cryptography* / Anguilla, 90-104. https://doi.org/10.1007/3-540-45472-1_7
[27] Savić, D. & Trikoš, M. (2011). The usage of smart card during creation of VPN connection. *Proceedings of 17th Conference on Computer Science and Information Technology YU INFO 2011* / Kopaonik.

Contact information:

Dejan SAVIĆ, PhD
Ministry of Defence Republic of Serbia
33 Kneza Miloša St., 11000 Belgrade, Serbia
E-mail: dejan.m.savic@mod.gov.rs

Mladen TRIKOŠ, PhD Candidate
Military Academy, University of Defence
33 Pavla Jurišića Šturma St., 11000 Belgrade, Serbia
E-mail: mladen.trikos@va.mod.gov.rs

Mladen VEINOVIĆ, PhD, Full Professor
Singidunum University
32 Danijelova St., 11000 Belgrade, Serbia
E-mail: mveinovic@singidunum.ac.rs

Dejan SIMIĆ, PhD, Full Professor
Faculty of Organizational Sciences, University of Belgrade
154 Jove Ilića St., 11000 Belgrade, Serbia
E-mail: simic.dejan@fon.bg.ac.rs