Paper

A Cloud-aided Group RSA Scheme in Java 8 Environment and OpenStack Software

Agnieszka Jakóbik

Faculty of Physics, Mathematics and Computer Science, Tadeusz Kościuszko Cracow University of Technology, Cracow, Poland

Abstract—In this paper a RSA based security system enabling the group of users to upload the single masked message to the cloud environment is proposed. Data stored are encrypted using RSA algorithm. The data receiver is able to encrypt the message retrieved from the cloud environment using private key. Two different separate RSA systems are used. The presented approach is divided into three parts. First, an RSA key is generated for each sender. Then masking the message by newly chosen mask proposed individually by every member, additionally encrypted by individual RSA private key of each member is proceed. Next, encrypting the gathered message inside the cloud environment, using the public key of the receiver is executed. In the third step, the message is decrypted by the receiver using his private RSA key. The scheme reduces the computational load on users side and transfers calculations and storage effort to the cloud environment. The proposed algorithm was developed for storing and sending the data that originally are produced by a group of users, but the receiver of the data is single. It was implemented using Java 8 and OpenStack software. Numerical test of different key length for RSA are presented.

Keywords—cloud computing, confidentiality, RSA cryptosystem.

1. Cloud Computing

Cloud computing is based on a business model in which resources are shared among at the network, host, and application level. It provides massive scalability and the ability to store large amount of data with efficient computational power. The cloud computing offers resources such as virtual-machine disks, image libraries, file storages, firewalls, mailing systems, load balancers, IP addresses, virtual local area networks, software bundles, operating systems, programming languages execution environments, databases, and Web servers [1].

Users may access to cloud environment using client devices, such as desktop computers, laptops, tablets and smart-phones. They are thin clients because cloud services do not require dedicated software on the client side (Fig. 1). Clients' software transfers calculation effort to the machine in the cloud. The most widely used examples of cloud computing are Google Cloud, Microsoft Cloud, Amazon Cloud and Adobe Creative Cloud [2]–[5].

The following features of cloud based IT systems distinguish them from traditional services and resources:

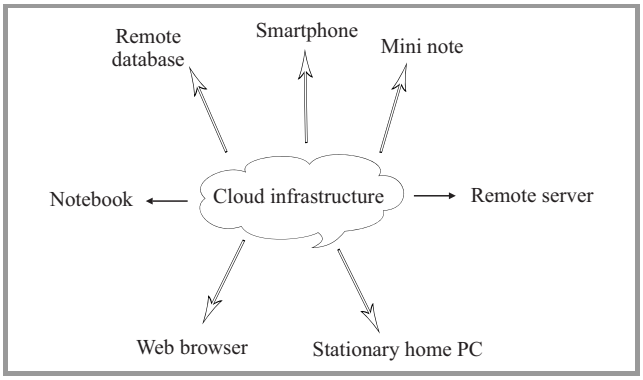

Fig. 1. Features of Cloud Systems.

- multitenancy – unlike previous computing models, which assumed single user resources in cloud model resources they are shared;
- massive scalability – traditional models might have hundreds or thousands of systems, cloud computing provides the ability to scale to tens of thousands of systems, and the ability to massively scale bandwidth and storage space;
- flexibility – users can increase and decrease their computing resources on demand;
- pay as you go – users pay only for using services and for the time they require them;
- self-provisioning of resources – users may introduce additional systems, i.e. processing capability, software, storage space and network resources [6];
- lower cost – cloud models have low upfront IT investment, and modular IT architecture. No infrastructure on the client side is required.

2. Security Issues in Cloud Environment

Cloud environment may be an easy target for hackers and organizations due to system availability and third-parity data usage. Moreover, user must protect the infrastructure by using safe connection to the cloud. Therefore, not every aspect of system security is fully controlled by cloud environment. The service provider have to deliver the

authentication and authorization techniques. Individuals may be assigned to different levels of privileges or may act together sharing the resources. Data and services in the clouds may be affected by numbers of attacks from distributed denial of service, phishing, data loss or leakage, unauthorized sharing of the technology and the abuse of the resources [6]. The security objectives are a key factor for decisions about choosing the cloud vendor. Nowadays, a lot of outsourcing information techniques, services and applications, and other resources has to be located inside a cloud computing environment, due to data size, speed and diversification. The user of the data benefits from procedures implemented inside the cloud. These may include given tools for authorization, authentication or data security support technique.

Cloud platform offers Infrastructure as a Service (IaaS), Platform as a service (PaaS) and Software as a service (SaaS). All three service models may be a target for the threats [7].
Infrastructure Security at the network level requires:

- ensuring the confidentiality and integrity of user data in transit to and from public cloud provider,
- ensuring proper access control (authentication, authorization, and auditing) to resources,
- ensuring the availability of the Internet resources in a public cloud that are being used by user or have been assigned to user by cloud provider [6].

The proposed scheme may be used in two first above levels.
Infrastructure security at the host level requires:

- for PaaS and SaaS models – hiding the host operating system from end users,
- the abstraction layer is not visible to users and is available only to the developers,
- host security responsibilities in SaaS and PaaS services are transferred to the cloud model [6].

The proposed scheme fulfills above requirements.
The data in cloud models should be treated deferentially according their status: data in transit, data at rest, data being processed. Additionally, multitenancy, data lineage, data provenance, data remanence have to be considered. Data in transit should be encrypted during transfer to and from a cloud provider. Encrypting data at rest is strongly suggested. Data lineage is important for an auditor's assurance. Integrity of data ensures that data has not been changed in an unauthorized manner or by an unauthorized person. Provenance means that the data is computationally accurate. Data remanence refers to the policy of treating data that are not used, not actual or has been erased by user from his applications [6]. In the proposed scheme data in transit are treated differently from data in rest, data lineage is strictly defined, processing of the data and data provenance are supported only by cloud software. Data remanence depends on both: user and cloud infrastructure. Data at rest stored for a long time inside cloud infrastructure should be encrypted using strong cryptography.

3. OpenStack Software

OpenStack is a cloud operating system that enables computing, storing, and networking resources [8]. It is open-source software for IaaS. Managing the IT infrastructure for the proposed algorithm was possible by using communication interface. Virtualized resources were used for calculating the results. They were available by prepared GUI environment. It consisted Compute (Nova service) module responsible for arranging, managing and providing virtual machines. Prepared working environment provided massively scalable, on demand access to computing resources. Object Store (Swift service) was used for managing storage system and Image Service (Glance service) was responsible for uploading and discovering data. Services were manageable from Horizon dashboard.

4. RSA Cryptosystem

RSA (proposed by Ron Rivest, Adi Shamir and Leonard Adleman [9]) is the public key algorithm for encrypting and decrypting the data. Encryption enables the communication to be private. Each message is encrypted before transmitting it to the receiver. The receiver knows the proper function to obtain the original message. Only the authorized user can have an access to the data stored. RSA algorithm may be used also for digital signatures evaluating [10]. Every plain text written in natural language is mapped into integer number and encrypted using arithmetic algorithm that proceeds on big numbers. RSA algorithm involves three stages:

- Key generation starts from choosing two different large random prime numbers p and q. Then, calculating $n = *pq$ the modulus for the public key and the private keys. Next step is to calculate the totient: $\phi(n) = (p-1)(q-1)$. After that one have to choose an integer e such that $1 < e < \phi(n)$ and e, is co-prime to e, i.e.: e, and $\phi(n)$ share no factors other than 1; $gcd(e, \phi(n)) = 1$. The public key is made of the modulus n and the public exponent e. The private key is made of the modulus n and the private exponent d, which must be kept secret;
- To encrypt the message M for particular receiver the sender is using public key of receiver. First the sender turns M into a number m (smaller than n) by using a reversible protocol known as a padding scheme [11]. He then computes the cipher text c:
$$c = m^e \mod n; \quad (1)$$
- The receiver can recover m, from c, by using his private key d:
$$m = c^d \mod n. \quad (2)$$
Given m one can recover the original message M by applying the reverse padding scheme.

It is currently recommended that n should be 1024 or 2048 bits long [12]. The RSA algorithm is used for example as a part of security system inside Google Cloud [13], [14], and in many handshaking protocols in the Transport Layer Security (TLS) [15].

5. Proposed Cloud-aided Group RSA Scheme

Proposed cloud-aided group RSA scheme is based on the procedure [16]. Additional stage was added to secure the first step of the algorithm before data are send to the cloud computing center. Moreover, proposed algorithm enables sending not only the single message (as in [16]) but the message may be composed from different parts coming from different senders. Proposed algorithm was designed for two separate data centers. Separating responsibilities increased the security of the cloud infrastructure itself. Furthermore, it made controlling the users privileges much easier.

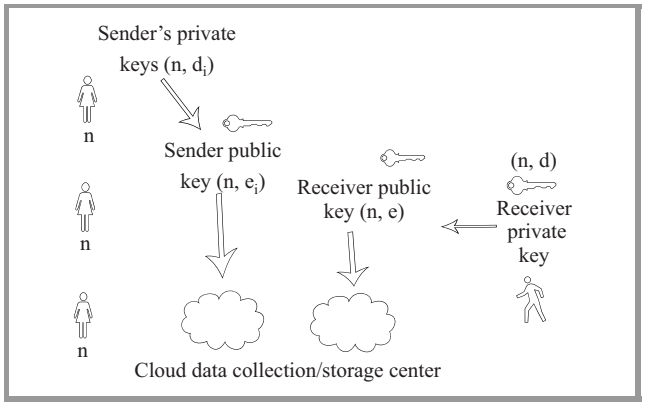

Fig. 2. Key preparation during stage 0.

5.1. Stage 0 – Organizing Infrastructure

Let's assume that a group of users (called senders) $S_1, S_2, \ldots, S_T$ is sharing the same file or data in the cloud infrastructure. They may modify and send it to receiver chosen from outside this group. All parts are gathered to portion of data called a message m in the cloud environment so that the receiver R may retrieve it on demand. The message is stored in the cloud storage center (CSC) in the encrypted form. Four actors are considered: group of senders, cloud data collection center (CDCC), cloud data storage center (CDSC), receiver of the data. The group of senders contacts only with CDCC, receiver only with CDSC. CDCC and CDSC are contacting each other (Fig. 2).

5.2. Stage 1 – Key Preparation and Data Gathering

The receiver generates his RSA private key (d,n) and public RSA key (e,n) and sends public RSA key to the CDSC. The CDSC sends to CDCC parameters n and e. CDCC sends parameter n to each sender. Each sender $i = 1, 2, \ldots, T$ generates the own private RSA key (d_i,n), and public RSA key (e,n). Then he sends public RSA key to the CDCC. Only the public keys are transferred. Then, sending decrypted data to CDCC is made. To do so, each sender is passing his part m_i of the message $m = m_1 * m_2 * \cdots * m_T$. He generates new random number a_i such that $a < n$ and computes $b_i = (m_i * a_i)^{d_i} \mod n$. Then he sends b_i to CDCC. Additionally $a_0 = a_1 * a_2 * \cdots * a_T \mod n$ is added. In case only one member is sending his message, all the a_i but his are set to 1 (Fig. 3).

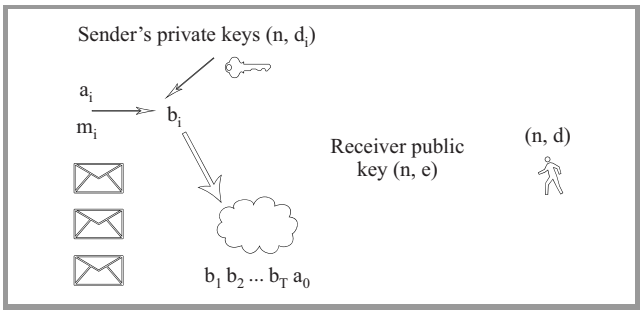

Fig. 3. Gathering messages from recipients inside CDCC during stage 1.

5.3. Stage 2 – Processing Data in CDCC

CDCC applies public key of each sender e_i to proper part of data and then applies public key of the receiver e:

$$v_0 = {a_0}^e, \tag{3}$$

$$v_i = (b_i^{e_i})^e \mod n =, \tag{4}$$

$$\left((m_i * a_i)^{d_i * e_i}\right)^e \mod n = (m_i * a_i)^e \mod n. \tag{5}$$

After that stage, data are composed into $V = (v_0, v_1, v_2, \ldots, v_T)$ vector. Then this vector is permuted by using random permutation P: $V_{perm} = (v'_0, v'_1, v'_2, \ldots, v'_T)$. Vector V_{perm} is sent to CDSC. The data from the group of senders is located in CDSC (Fig. 4).

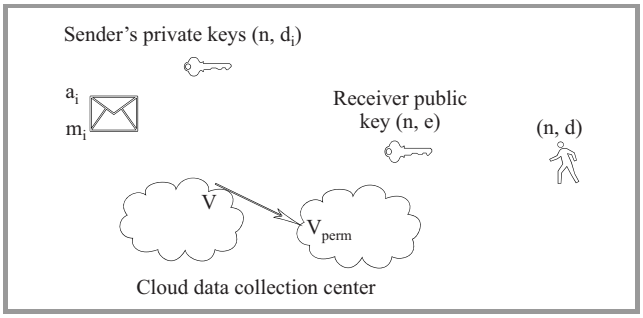

Fig. 4. Mixing messages from recipients inside CDCC into single message during stage 2.

5.4. Stage 3 – Processing Data in CDSC

CDSC applies inverse permutation P^{-1} to V_{perm}, obtaining $V = (v_0, v_2, \ldots, v_T)$ vector. Then computes:

$$c = (v_1 * \cdots * v_T) * v_0{}^{-1} \mod n = , \quad (6)$$

$$(m_1 * m_2 * \cdots * m_T)^e \mod n = , \quad (7)$$

$$m^e \mod n . \quad (8)$$

This value is stored until the receiver would like to retrieve it (Fig. 5).

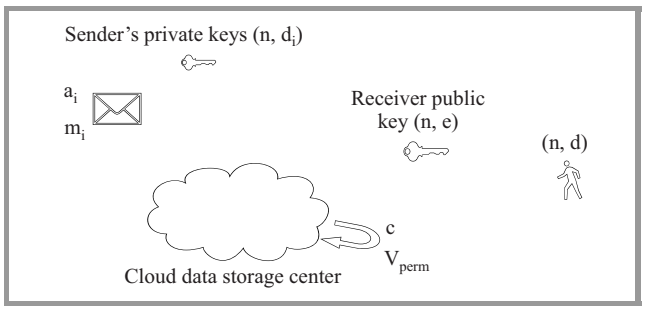

Fig. 5. Storing data inside CDSC in the form of single message during stage 3.

5.5. Stage 4 – Downloading Decrypted Data to CDSC and Encryption by Receiver

Receiver uses his own private key d to calculate:

$$m = c^d \mod n = (m^e)^d \mod n = m \mod n . \quad (9)$$

Given m one can recover the original message M by using reverse padding scheme.

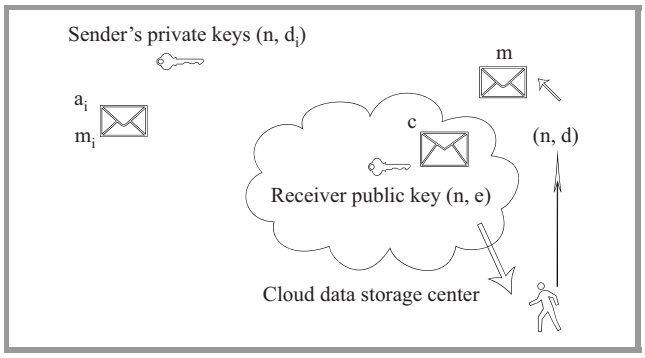

Fig. 6. Retrieving message from CDCC and encryption during stage 4.

Data is more vulnerable to unauthorized modification or appropriation when it is in storage than it is being processed. This is the reason, why the strongest RSA keys should be applied during stages 2 and 3. It also moves storage afford the data into cloud environment. Masks $a_1, a_2, \ldots, a_T$ are different for each message. There is no need for generating new RSA key for every single message. Using the masks, the true message m_i is indistinguishably from empty message m_i during sending stage 1. If particular sender is not generating his part computational afford is low, because masks may be much smaller then messages. The group is invisible to the receiver. The receiver does not know who from the group had sent the particular part of the message or even that the message was composed from parts (Fig. 6).

6. Experimental Results in Java 8 Environment and OpenStack Software

CDCC operation were calculated using several units of PCs connected into one network. Such a solution enables gathering the data from users by independent units. Additionally, the units were used as direct way of the data uploading. The uploading the data for lean clients was made by Web browser application running on each unit. Cloud data storage operations were performed using single OpenStack instance [17].

Java 8 environment was chosen for three main reasons. Firstly, for the convenience of the security package. Secondly, for the possibility of usage the BigInteger library that was necessary for computing on such large numbers. Finally, Java 8 provided secure pseudo-random number generator for calculating e, p and q values.

The RSA part of the scheme was implemented using the following Java 8 classes, methods and interfaces from Java security package:

- Key – interface models the base characteristics for the keys,
- KeyFactory – key factories are used to convert keys into key specifications (transparent representations of the underlying key,
- KeyPair – serves as a container for public and private keys,
- KeyPairGenerator – a class used to generate key-pairs for a security algorithm,
- GeneratePrivate – method generating a private key from the provided key specification,
- GeneratePublic – method generating a public key from the provided key specification,
- Interface KeySpec – transparent specification of the key that sets a cryptographic key. If the key is stored on a hardware device, its specification may contain information that helps identify the key on the device [18].

To generate the keys from the Key Factory the following Java 8 code was implemented:

```
KeyPairGenerator key_par_gen =
  KeyPairGenerator.getInstance("RSA");
key_par_gen.initialize(2048);
KeyPair kp =
  key_par_gen.genKeyPair();
PublicKey publicKey =
  key_par.getPublic();
PrivateKey privateKey =
  key_par.getPrivate();
KeyFactory factory_rsa =
  KeyFactory.getInstance("RSA");
RSAPublicKeySpec pub =
  factory_rsa.getKeySpec
   (publicKey, RSAPublicKeySpec.class);
RSAPrivateKeySpec priv = factory_rsa.
  getKeySpec
   (privateKey,RSAPrivateKeySpec.class);
```

To retrieve the values of the key the following Java 8 code was proposed:

```
BigInteger
n = (BigInteger)
object_input_stream.readObject();
BigInteger
e = (BigInteger)
object_input_stream.readObject();
KeyFactory factory_rsa =
KeyFactory.getInstance(``RSA");
if (keyFileName.startsWith(``public"))
 return factory_rsa.
 generatePublic
 (new RSAPublicKeySpec(n, e));
else
 return factory_rsa.
 generatePrivate
 (new RSAPrivateKeySpec(n, d));
```

Public class called *Cipher* was used to encrypt and decrypt the messages during stage 1 [19].

- javax.crypto.Cipher – class provides encryption and decryption;
- javax.crypto.CipherInputStream – constructs a CipherInputStream from an InputStream and a cipher;
- javax.crypto.CipherOutputStream – constructs a CipherOutputStream from an OutputStream and a cipher.

Keys were stored using secured disk space. There were retrieved on demand when new instance of RSA algorithm was created:

```
Key public_Key =
readKeyFromFile("public.key");
    Cipher cipher =
    Cipher.getInstance("RSA");
cipher.init
  (Cipher.ENCRYPT_MODE, public_Key);

 Key private_Key =
 readKeyFromFile("private.key");
   Cipher cipher =
   Cipher.getInstance("RSA");
cipher.init
(Cipher.DECRYPT_MODE, private_Key);
```

7. Numerical Tests

Two physically separated data centers were used. Cloud data collection center was located in Institute of Computer Science at Cracow University of Technology in Poland, and Cloud data storage center was located in Cloud Competency Center at National College of Ireland [20].

Cloud data collection center consisted of 28 units of the same characteristic: AMD FX-6300 6-cores processor, 2 GHz, cache size 2048 KB, Windows 8.1 Enterprise or Linux Fedora rel. 22. This solution enables uploading the data remotely using Web application and mobile phones or tablets. Additionally 28 users may upload their data in the same time personally while sitting in front of the units. The CDCC used Java 8 Web application, described in Section 5. The CDSC was working under OpenStack software. Single instance was used. The OpenStack instance m1.large was configured as follows: RAM 8 GB, 4 VCPU, 80 GB HDD [21], with Ubuntu operating system and OpenSSL library [22]. The experimental server specified in [23] was used. The characteristics of the virtual machine used for calculation were: OpenStack Nova system, memory 96 KB BIOS, processor Intel Xeon E312xx (Sandy Bridge), 8 GB system memory. The decryption was made by data storage center on users demand. The data was sent to the end user via secure channel.

Table 1
Mean encryption time in CDCC, data file size 10 MB

Key length	1024 bit	2048 bit
Real time	1 m 56.112 s	2 m 25.334 s
User time	1 m 17.357 s	2 m 21.434 s
System time	1 m 14.711 s	2 m 20.724 s

Table 2
Mean decryption time in CDSC OpenStack instance, data file size 10 MB

Key length	1024 bit	2048 bit
Real time	0 m 31.484 s	0 m 45.572 s
User time	0 m 25.984 s	0 m 37.245 s
System time	0 m 26.226 s	0 m 40.378 s

According to the recommendation for key management [24], secure 1024 or 2048 bit keys were used. The chosen results of experiments are presented in Tables 1 and 2.

It was found profitable to proceed long messages in chunks. Chunks were generated by dividing the message into equal parts. Specified number of bits was considered. Chunks was coded and decoded in parallel mode. The time of calculation is highly dependable on the busyness of the computational unit. That is why, mean encryption time in CDCC unit is presented.

It was stated that enlarging key length from 1024 to 2048 bits resulted in computational time increasing about:

- 34% for Data Collection Unit real time,
- 30% for OpenStack instance real time.

Instead of multiplication of fragmentary messages other operations may be performed, then equivalent methods for generating a_0 have to be adopted.

8. Conclusions and Future Development

Proposed RSA based security system enables the group of users sending the single message to the cloud environment. Inside cloud environment data are stored encrypted by RSA algorithm. Receiver of the data is able to encrypt the message retrieved from the cloud environment on demand, regardless when message was send. Two different separate RSA systems are used according to the computing capabilities. Simple, computationally non-demanding procedure was proposed for a thin client in the form of masking. Parallel computations using strong secure keys were incorporated for cloud environment. The RSA key generating and masking the message was proposed for each sender. Such a procedure enabled treating data at rest and data being processed in a different way, according to the security requirements. It also allowed minimizing computational afford on the user's side and using the benefits of cloud computing security infrastructure.
Separating senders from receivers allowed dividing thin and not secured clients from data store center. Grouping senders and receivers in different location simplified the management of users' rights and privileges.
Additionally, encrypting the gathered message inside the cloud environment enables to hide the group from the receiver. Such a scheme may be used during electronic voting, electronic reviewing, and team working on the same document. Decryption the message by the receiver using his private RSA key ensures that if the private key was kept secret no one but receiver could read the massage stored inside cloud environment.
Future development of research is planned. The investigation on the influence of differed OpenStack instances types is considered. Also, exploring more advanced methods for lean clients. The masking method is planned to be replaced by fragmenting and mixing the message. Further investigation on increasing computational efficiency is necessary.

Acknowledgement

The author would like to express great gratitude for Horacio González-Véle, the Head of NCI Cloud Competency Center in Dublin for enabling tests on OpenStack software.

References

[1] J. Rittinghouse and J. Ransome, *Cloud Computing: Implementation, Management, and Security*. CRC Press, 2009.

[2] Google Cloud Platform [Online]. Available: https://cloud.google.com/

[3] Amazon Drive [Online]. Available: https://www.amazon.com/clouddrive/home

[4] Microsoft Cloud [Online]. Available: http://www.microsoft.com/enterprise/microsoftcloud

[5] Adobe Creative Cloud [Online]. Available: http://www.adobe.com/pl/creativecloud.html

[6] T. Mather, S. Kumaraswamy, and S. Latif, *Cloud Security and Privacy An Enterprise Perspective on Risks and Compliance*. O'Reilly Media, 2009.

[7] W. Jansen and T. Grance, "Guidelines on security and privacy in public cloud computing", 2011 [Online]. Available: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-144.pdf

[8] OpenStack website, https://www.openstack.org

[9] R. L. Rivest, "Cryptography", in *Handbook of Theoretical Computer Science*, J. Van Leeuwen, Ed. MIT Press, 1990, vol. A, pp.717–755.

[10] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[11] PKCS #1 RSA ver. 2.2 Cryptography standard, RSA Laboratories, Oct. 2012 [Online]. Available: http://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf

[12] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[13] Google Cloud Documentation [Online]. Available: https://cloud.google.com/storage/docs/authentication

[14] Amazon Elastic Compute Cloud User Guide for Linux [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html

[15] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol, Version 1.2", Network Working Group, Aug. 2008 [Online]. Available: http://tools.ietf.org/html/rfc5246

[16] C.-H. Lin, C.-Y. Lee, and T.-W. Wu, "A cloud-aided RSA signature scheme for sealing and storing the digital evidences in computer forensics", *Int. J. Secur. & and Its Appl.*, vol. 6, no. 2, 2012.

[17] OpenStack Website, https://www.openstack.org/

[18] Java Platform Standard Edition 8 Documentation, Oracle, 2015 [Online]. Available: https://docs.oracle.com/javase/8/docs

[19] Java Platform, Standard Edition 8 API Specification, Oracle, 2015 [Online]. Available: https://docs.oracle.com/javase/8/docs/api/javax/crypto/Cipher.html

[20] National College of Ireland Website, https://www.ncirl.ie/

[21] D. Grzonka, "The analysis of OpenStack cloud platform: features and performance", *J. Telecommun. Inform. Technol.*, no. 3, pp. 52–57, 2015.

[22] OpenSSL, Cryptography and SSL/TLS Toolkit [Online]. Available: https://www.openssl.org

[23] D. Grzonka, M. Szczygieł, A. Bernasiewicz, A. Wilczyński, and M. Liszka, "Short analysis of implementation and recources utilization for the OpenStack cloud computing platform", in *Proc. 29th Eur. Conf. Modelling & Simul. ECMS 2015*, Albena (Varna), Bulgaria, 2015.

[24] E. B. Barker *et al.*, "Recommendation for Key Management, Part 1: General (Revised)", NIST Special Publication 800-57, National Institute of Standards & Technology, Mar. 2007 [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf

Agnieszka Jakóbik (Krok) received her M.Sc. in the field of stochastic processes at the Jagiellonian University, Cracow, Poland and Ph.D. degree in the field of neural networks at Tadeusz Kosciuszko Cracow University of Technology, Poland, in 2003 and 2007, respectively. From 2009 she is an Assistant Professor at Faculty of Physics, Mathematics and Computer Science, Tadeusz Kościuszko Cracow University of Technology. Her main scientific and didactic interests are focused mainly on artificial intelligence: artificial neural networks, genetic algorithms, and additionally on parallel processing and cryptography.

E-mail: agneskrok@gmail.com
Faculty of Physics, Mathematics and Computer Science
Tadeusz Kościuszko Cracow University of Technology
Warszawska st 24
31-155 Cracow, Poland