# EXPERIMENTS WITH ABE SCHEMES ON OPENSTACK FUEL IAAS PLATFORM

G. Bhanu Prakash*

Department of CSE, NITTE Research and Education Academy
Bangalore, Karnataka, India
bhanuprakash.gopularam@gmail.com

Nalini N

Department of CSE, NMIT, Bangalore, Karnataka, India
nalinaniranjan@hotmail.com

Chandramohan Ajmeera

Department of CSE, Osmania University, Hyderabad, Telangana, India
cm.ajmeera@gmail.com

**Abstract**

Organizations are embracing cloud technologies for elasticity and reduced operational costs. Building public cloud systems requires significant planning and upfront costs due to hardware and software requirements. Security is challenging in cloud context due to increased attack surface which can result in serious vulnerabilities if not properly planned. Traditional cryptographic schemes such as PKI help in data protection but they involve complexity in management and lack flexibility in designing access control policy. In this paper, we evaluate Attribute-based encryption schemes for logs preservation and assess performance for various configurations in cloud environment viz., OpenStack. OpenStack is a public cloud software which enables building Infrastructure-as-a-Service platform leveraging the hardware resources, it is large collection of independent components tied together to provide unified service to end users.

OpenStack Fuel is a testing framework used by many organizations in automating process of deployment, configuration and testing the OpenStack software. In OpenStack setup lot of sensitive information generated by server components is stored in log files, if the data is not properly preserved it can result in leakage. We use ABE schemes for logs preservation as they overcome challenges associated with certificate management and at the same time provide more flexibility to users compared to traditional cryptosystems. In this paper, we describe cloud infrastructure setup using OpenStack and OpenStack Fuel. We provide reference architecture for building a security framework based on Attribute-based encryption schemes.

***Keywords*****:** data confidentiality; OpenStack; attribute-based encryption; multi-tenancy; network security

## 1. Introduction

Main characteristics of cloud like on-demand access, rapid elasticity, and metered services make cloud attractive for business operations. Cloud services are mainly categorized as *Infrastructure-as-a-service* which provides virtual resources on top of bare metal hardware and hypervisor, *Platform-as-a-service* provides development environment or services which help building applications and finally *Software-as-a-service* that provides application software for the users. The IaaS cloud services provide maximum flexibility in terms of configuration options but to provide security it requires great commitment from cloud consumers compared to other cloud service models. Cloud participants constitute multiple users like cloud consumer which is end-user, cloud service provider which provides cloud services and cloud auditor an independent entity for assessment of services offered [2]. Large cloud deployments have few more participants like cloud broker which help in cloud service delivery and cloud carrier connects cloud provider with cloud consumer. The cloud service provider manages infrastructure and offers various services to cloud consumers [6].

In this paper, we experiment with Attribute-based encryption schemes for data storage as they provide data confidentiality along with flexibility in sharing the data unlike the traditional cryptosystems based on PKI infrastructure. For performing computationally intensive operations, we leverage OpenStack infrastructure configured using Fuel-Jenkins plugin to assist in large dataset encryption and decryption. The Identity and Attribute-based encryption techniques are also called as ID-PKC techniques for brevity.

### 1.1. *Key Contributions*

In this paper, we describe Platform-as-a-Service cloud deployment built using OpenStack along with infrastructure automation tool known as Fuel. The Jenkins plugin for OpenStack Fuel provides a job execution framework for running computationally intensive tasks like data encryption, decryption.

For data security, we experiment with ID-PKC or Attribute based encryption schemes such as Ciphertext-Policy Attribute Based Encryption proposed in [3] referred as BSW scheme for network telemetry encryption. The CP-ABE schemes allow embedding access control policy structure within ciphertext therefore allows implementation of Role-based access control policies for data confidentiality [8, 14]. Later the data is decrypted by authorized users or services having keys with valid access structure. We evaluate cryptographic scheme performance for various operations like setup, key extraction, encryption and decryption and provide analysis of the approach. We provide details for setup of large scale infrastructure using OpenStack cloud platform. We use automated jobs for encryption of large network telemetry data using distributed infrastructure having multiple master-slave machines working on data blocks in parallel. The proposed approach uses OpenStack infrastructure with multiple nodes to perform encryption of large datasets.

## 2. Preliminaries and Background

### 2.1. *Identity and Attribute-based Encryption*

Identity-based cryptography uses publicly identifiable information such as email-id, organization-id for public key generation. The NetFlow record fields such as device-identifier, service-name, datetime-of-event can be used as identifiers in public key generation. It was initially proposed by Adi Shamir and first practical scheme was presented by Boneh and Franklin [4]. The Attribute-Based encryption is sub-topic of fuzzy-IBE proposed by Sahai [13], the user with secret key can decrypt encrypted text using new identity only when there is significant overlap of key attributes. As publicly identifiable information is used in public key generation the user is relieved from overhead of certificate management. Attribute-based encryption was introduced by Sahai et. al [3] where user identifier information determines the access level. The ABE schemes were extended by Goyal [11] as Key-policy Attribute-Based Encryption (KP-ABE) where attributes decorate ciphertext and attributes having access structures are embedded in secret keys.

Table 1: Inputs and outputs for various operations of Ciphertext-Policy ABE and Key-Policy ABE

| CP-ABE algorithm | Description |
|---|---|
| Setup | Initialize all the system parameters<br>Input: security parameter k<br>Output: Public Key PK along with master secret key MSK |
| Key Generation | Calculate private key from PKG master secret and an identity using system parameters.<br>Input: set of attributes γ, master secret key MK<br>Output: secret key SK using which message can be decrypted which is encrypted using access structure T given that γ matches the access structure T |
| Encryption | Encrypt data using user public key generated from system parameters and identity<br>Input: message M, public key PK, along with access structure T<br>Output: cipher text CT |
| Decryption | Decrypt the contents using user private key generated from PKG master secret and user identity<br>Input: user secret key SK, set of attributes γ and cipher text E<br>Output: Actual message given that γ satisfies the access structure T part of encrypted text |

Bethencourt [3] gave implementation of Ciphertext-policy Attribute-Based Encryption (CP-ABE) in the same year which is a complimentary to KP-ABE, the attributes describe user key and the formulas part of credentials are embedded into cipher text. With CP-ABE the role-based access control policies [3] can be embedded into the cipher text. Ostrovsky [11] proposed KP-ABE for non-monotonic access structures. The initial implementation of Ciphertext-policy ABE used user access information embedded into ciphertext and attributes are associated with user keys which decide access level. The ABE schemes allow more flexibility in data sharing as role-based access information associated with user determines decryption mechanism [9]. For network telemetry data encryption and distribution, the associated data fields for example [*date-time, event-type, IP-address*] determine the access privileges.

### 2.2. *Advantages of ID-PKC Cryptography*

The attribute-based encryption schemes like CP-ABE and KP-ABE are variants or superset of Identity based encryption schemes (ID-PKC). In ID-PKC the public keys are derived from user or system identities and it eliminates the need for certificates. Following are some of the advantages over traditional PKI cryptosystems

- Certificate Management: ID-PKC schemes are certificate-free and directory-free and thus enable flexibility in communication. In PKI certificate management involves lot of overhead in maintaining valid certificates, frequent synchronization with certificate authority either using CRL (certificate-revocation lists) or OCSP (online certificate status protocol) is required.
- Automatic Revocation: The system can include validity period within keys which automatically invalidates the private key after specified duration.
- Support of Key recovery: The trusted authority can calculate private key of any user and facilitate content scanning and thus eliminates need for external certificate authority.

### 2.3. *OpenStack Deployment using Fuel*

OpenStack is an open source cloud platform used in building Infrastructure-as-as-service cloud model. RackSpace and NASA jointly collaborated in building the platform and in 2012 a consortium of 500 companies formed for development. OpenStack follows modular architecture with different components like Nova (compute service), Neutron (networking service), Cinder (block storage service), Horizon (dashboard service) and importantly Keystone component provides centralized repository with identity information of users which is useful in authentication and authorization of users or services [7].

Fuel is deployment and management tool for OpenStack. Until Fuel was developed there was no proper orchestration infrastructure available for OpenStack platform. Some of key features of Fuel include hardware discovery and configuration, view logs in real-time, support for high availability and non-high availability deployments. Fuel comprises of several components independently tied together.

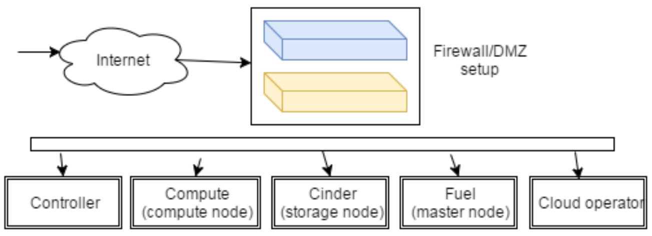

Fig. 1. OpenStack deployment using Fuel

Fuel software provides thin layer or programmatic interfaces as shown in Fig. 1 uses various tools like Cobbler, Puppet scripts for management and execution of tasks and automation of routine configuration tasks. Fuel architecture comprises several independent components tied together. Nailgun component is heart of Fuel. It manages deployment data in a repository and exposes REST API to third parties. Astute is part of Fuel which is work horse or processing engine for Nailgun which executes the tasks using the hardware resources.

- **Cobbler** is provisioning service for agent creation and Puppet allows deployment of the services
- **OSTF** is OpenStack Testing Framework used for post-deployment verification of OpenStack

Fuel infrastructure supports running nightly tests, integration tests, centralized log storage, Dockers servers monitoring service. Continuous Integration (CI) is a development paradigm where the code changes are tested immediately and deployed if no issues found. Fuel deployment consists of multiple nodes having Jenkins installation connected in master-slave architecture. Jenkins master node handles all tasks related to build system and does the job execution. Initial configuration tasks are handled by puppet software which essentially runs the configuration scripts on slave machines. The Jenkins master-slave connectivity happens after SSH-key exchange and finally slave.jar file is placed inside the slave machines to make them ready for jobs execution. The jobs to be executed using Jenkins architecture are written using YAML based templates.

### 2.4. *OpenStack Fuel Network Logs*

OpenStack services generate extensive logs which contain configuration and job execution information. Log files are generated by various OpenStack components such as Horizon, KeyStone, Neutron and Fuel [12, 15] as shown in Table 2. These logs contain sensitive information related to deployment helpful in troubleshooting and analysis. The network telemetry constitutes NetFlow records containing network activity [5]. Network telemetry is generated by network devices like switches, routers, proxy servers are valuable in network monitoring and diagnostics [10]. A typical network interface 100 Mbps generates about 12.5 MB per second or 45 Terabytes per hour. Even with random sampling it would result large amount of data and encryption of such data involves large computation overhead.

Table 2: OpenStack service log file name and location

| Component | Log files |
|---|---|
| Horizon | /var/log/apache2/horizon_access.log<br>/var/log/apache2/horizon_error.log |
| Keystone | /var/log/apache2/error.log<br>/var/log/apache2/access.log<br>/var/log/apache2/keystone_wsgi_admin_access.log<br>/var/log/apache2/keystone_wsgi_admin_error.log |
| Netron | /var/log/openvswitch |
| Fuel Master node | /var/log/remote/<NODE_FQDN_OR_IP>/attrd.log<br>/var/log/remote/<NODE_FQDN_OR_IP>/crmd.log<br>/var/log/remote/<NODE_ FQDN _OR_IP>/cib.log |

Semantic representation of sample log entry generated by OpenStack Keystone service:

[*Date time of event, Event category, ProcessID, IP address, Description*]

Sample log file entry:

```
2016-10-22 15:49:31 INFO[5436] (node) Interfaces are locked for update on node R1N15
[pid: 853|app: 0|req: 5867/42914] 10.120.45.168 () {36 vars in 527 bytes} [Thu Oct 22 15:49:31 2016] PUT /api/nodes/agent/ => generated 10 bytes in 131 msecs (HTTP/1.1 200) 4 headers in 185 bytes (2 switches on core 0)
2016-10-22 15:49:31 INFO[5436] (node) Volume information is locked for update on node R1N15
2016-10-22 15:49:30 INFO[5436] (node) Interfaces are locked for update on node R1N03
[pid: 853|app: 0|req: 5866/42913] 10.120.45.168 () {36 vars in 528 bytes} [Thu Oct 22 15:49:30 2016] PUT /api/nodes/agent/ => generated 10 bytes in 120 msecs (HTTP/1.1 200) 4 headers in 185 bytes (2 switches on core 0)
```

The fields like *Date-time of event, Event category, ProcessID* along with user access information such as *designation, department-id* are used for public key generation and data is encrypted before storage, so that other end-point component or services having proper keys can only access them.

## 3. Methodology

### 3.1. *Infrastructure Setup*

The infrastructure setup involves master node having Jenkins plugin and client nodes connected in master-slave architecture. The master node accepts tasks in the form of jobs and takes help of slave machines to perform computation jobs. The machine configuration is done using configuration data present in Cobbler scripts. The software installation on slave machines is performed automatically using puppet automation scripts. Our set up had 1-master node and 3-slave machines which also support high-availability. Since OpenStack Fuel supports pluggable architecture, the nodes are added or removed dynamically without causing service outage.

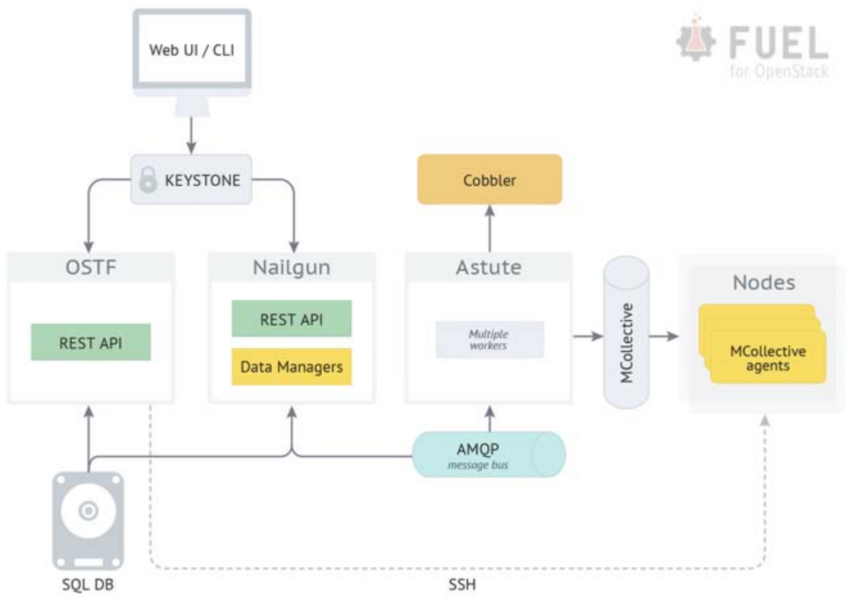

Figure 2: OpenStack Fuel components and architecture [15]

Jenkins CI server is configured on master node allows monitoring job status and test results. Jenkins Job builder (JJB) manages jobs and tasks are written in yaml markup language. In order run the jobs on slave nodes, the Jenkins jobs configuration data is downloaded from git repository and required packages installed.

(1) Create a local DNS service, update configuration files (present in /etc/resolv.conf)

(2) Install Jenkins job builder and install required tools

*apt-get install -y git python-tox*

(3) Clone the jenkins job files

*git clone https://github.com/fuel-infra/jenkins-jobs.git*

(4) Create jenkins_jobs.ini file with job configuration data and startup schedule information

(5) Upload JJB jobs to Jenkins master node

*jenkins-jobs --conf conf/jenkins_jobs.ini update servers/fuel-ci:common*

(6) Build OpenStack ISO using Jenkins and deploy on target servers

Master-slave setup has three machines for jobs execution. Once environment is setup Jenkins jobs are created to run the automation tasks. Once master node is online it accepts connections from slave machines. The slave machines should have proper master SSH-key to communicate with master node. The Jenkins master web interface is used for creating slave nodes, which are identified using FQDN (fully qualified distinguished name).

The Jenkins master node slices the test dataset into number of fragments equal to number of compute or slave machines available. The job execution on fragmented datasets and results are returned to the master node. The master node is responsible for results aggregation.

## 3.2. *Test Data*

The initial network telemetry dataset had 10 million records aggregated from few hundred network devices. For experimental purposes we created dataset records in increasing order [*20,000; 40,000; 60,000;.. upto 2,00,000 records*] and performed assessment. We used these dataset files for performance benchmarking. For experimentation, we used X64 machines having Ubuntu 14.10 (Utopic Unicorn) as in Table 3.

For implementation of cryptographic schemes, we used Charm cryptographic library [1] version v0.43. We have chosen this library for our implementation because the framework has basic support for pairing generation and cryptographic routines upon which new schemes can be built rapidly. Some of the other libraries used are OpenSSL 1.0.1, GMP 6.0.0a and Pairing based cryptography library version 0.5.14 authored by Stanford.

Table 3: Details of Test Machine Setup used in Experiments

| System Parameter | Configuration |
|---|---|
| Processor | Intel Core i5 4310u CPU |
| OS | Ubuntu 14.10 (Utopia Unicorn) |
| Memory | 8 GB RAM |
| Disk size | 40 GB disk |
| Architecture | X-64 |
| Speed | 2 GHz |
| PBC (Pairing Based Library) | Pbc 0.5.14 |
| Python | Version 3 |

## 4. Methodology and Experimental Evaluation

Here we analyze the performance of operations such as algorithm setup, key generation, encryption, decryption using master-slave architecture single slave node and multiple slave nodes. The efficiency of algorithm depends on number of factors such as number of independent pairings, general exponentiations and hashing operations.

### 4.1. *Algorithm Setup and Key Generation*

The key server generates key-pair. Pairing is said to be symmetric if elements (G1, G2) are equal else pairing is called asymmetric. For experimentation, we used symmetric pairing SS512 curve with 512-bit base field. The pairing generation using symmetric curves is faster than (sometimes 50% more performant) asymmetric curves [9] such as MNT159 and MNT224 due to reduced element space.

Table 4: ABE Algorithm Setup and Key generation time

| Algorithm | Time (milli-seconds) |
|---|---|
| Setup | 38.216 |
| Key generation | 22.098 |

Initially master keys are generated with help of Private key generator routine. The setup uses 160-bit elliptic curve generated from a super singular curve $y^2 = x^3+x$. The setup and key generation operations are one time operations which are done much before the actual communication or during off peak hours in cloud environment. The performance of these operations is less significant as they can be performed during off peak hours in a data center. The traditional Secure Socket Layer (TLS/SSL) protocol is used for passing secret keys.

### 4.2. *Encryption*

Initially we experimented with Jenkins master having single slave node. The log entry fields such as *EventID, ProcessID, Node-ID* along with organization access structure information is used in creation of user public key. The jobs are delegated to the slave node and after completion the results are aggregated at master node and persisted in the backup medium.

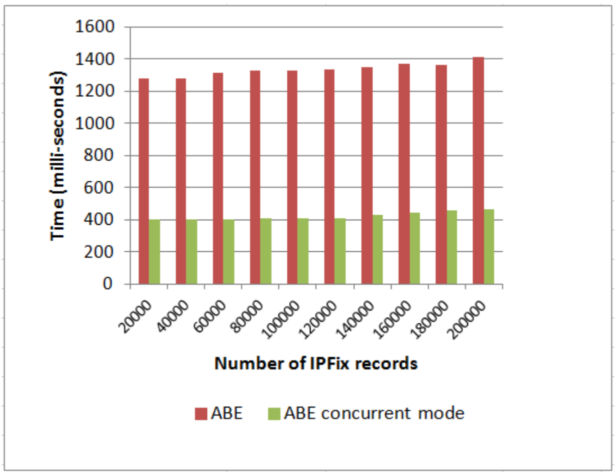

Fig. 3. Time comparison chart for Encryption

## 4.3. *Decryption*

Data decryption happens in two-steps, in initial phase the match of user identifiers happens only if match routine succeeds the content decryption is performed using user secret key. The match routine for a data record does data decryption on data (*PK, sk, ciphertext*) and the decoded content has *FLAG* field as prefix. If the match routine is successful the content is decrypted using user secret key *sk* and public key *PK*, the decrypted text contains symmetric key using which decryption is performed once more. This functionality is called "Symmetric Crypto Abstraction" where majority data encryption is performed using symmetric key algorithm like AES with CBC mode and symmetric keys are encrypted using the asymmetric algorithm. If match routine is unsuccessful then record processing is stopped and proceeds with next record processing.

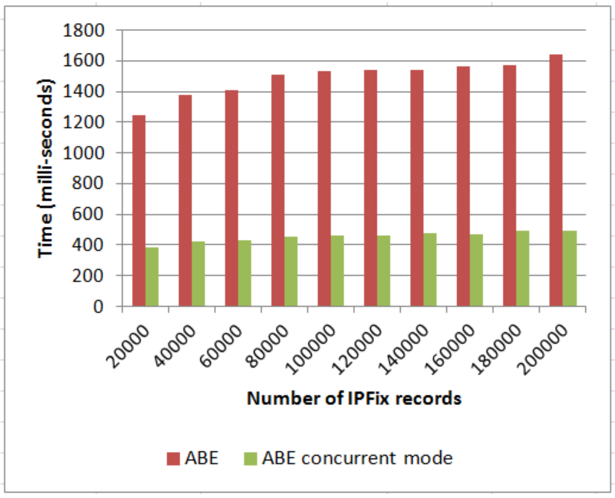

Fig. 4. Time comparison chart for Decryption

## 4.4. *Discussion*

The current implementation uses PKI infrastructure for master-slave communication. Using PKI for trust establishment and securing communications is beneficial but it involves overhead in management of certificates and requires frequent interaction with certificate authority. The efficiency of key-generation and setup algorithm not significant as it is one time operation and can be performed much before the communication. The encryption algorithm performance is dependent on number of leaf nodes in access structure, needs two exponentiations per leaf node, it is O($2*l$) where $l$ is number of leaf nodes in policy tree. Given multi-node setup the computation can be divided among the slave machines in Jenkins grid-hub setup, therefore the computation time reduces to O($2*l/k$) where k is number slave machines in OpenStack setup. Similarity decryption operation requires O($2*p+l$) where p is number of pairings and l is exponentiations. Likewise, when multi-node setup is used the decryption time reduces proportionately to O($(2*p+l)/k$) where k is number slave machines. The ID-PKC schemes provide more advantages compared to traditional PKI infrastructure, the deployment requires fewer components (no dependency on certificate authority) and provides certificate free communication among clients. Following are some advantages associated in using ID-PKC schemes over PKI in cloud environment.

- Each API endpoint in OpenStack has verification information like certificates, revocation list, which help in authorizing the user requests. There is large overhead in maintaining certificates in end points, and maintenance of individual keys is relieved to some extent using caching and TPM (Trusted party module)
- Basic implementation of OpenStack security framework suffers from single point of failure for keystone service, as it validates all incoming requests and may suffer from scalability problem. Usually cloud applications adopt Token scoping scheme as it enables Keystone to handle large number of users. Key Manager back end storage is ideally distributed in different geographic locations for availability
- The end-points need frequent synchronization with Keystone server
- If communication involves multiple end-points the user need to deal with multiple tokens and associated certificates, the certificate management is complex as each end-point needs to maintain local copy of trust store of certificates.

## 5. Conclusion and Future Works

In cloud network, there is a need for preserving sensitive data generated from network devices, the data privacy is difficult unless end-to-end mechanism is used considering cloud dynamicity. The network devices emit sensitive information which is useful for monitoring the state of cloud and post-mortem analysis of network events. The existing mechanisms do not offer flexibility to share data selectively between cloud users and users need to deal with complexity and incur significant cost. Our approach provides a scalable framework for large data processing, it leverages OpenStack cloud platform with master-slave architecture built using Fuel-Jenkins software for load distribution. The security mechanism described using Ciphertext-Attribute Based Encryption scheme can be further be extended using features like oblivious search or hidden keyword search which is useful in applications like cloud data analytics and using predicate encryption for monotonic and non-monotonic access structures.

**Acknowledgments**

We would like to thank Oracle Corporation for supporting this work and would like to thank crypto stack exchange community for insightful discussions and suggestions.

## References

[1] Akinyele, J; Garman, C; Miers, I; Pagano, M; Rushanan, M; Green, M; Rubin, A (2013): Charm: a framework for rapidly prototyping cryptosystems, Journal of Cryptographic Engineering, vol. 3, no. 2, pp. 111–128
[2] Armbrust , M et al. (2009): Above the Clouds: A Berkeley View of Cloud Computing, Technical Report No. UCB/EECS-2009-28
[3] Bethencourt, J; Sahai, A; Waters, B (2007): Ciphertext-policy attribute based encryption, Security and Privacy, pp. 321–334, IEEE Symposium
[4] Boneh, D; Franklin, M (2007): Identity-based encryption from the weil pairing, Advances in Cryptology. Springer, pp. 309–320
[5] Broder, A.; Kumar, R.; Maghoul, F.; Raghavan, P.; Rajagopalan, S.; Stata, R.; Tomkins, A.; Wiener, J. (2000): Graph structure in the Web. Computer Networks, **33**(1–6), pp. 309–320.
[6] Chen, D; Zhao, H (2012): Data Security and Privacy Protection Issues in Cloud Computing, Computer Science and Electronics Engineering (ICCSEE), pp. 647-651
[7] Cui, B; Xi, T (2015): Security Analysis of Openstack Keystone, Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 283–288
[8] Dara, S; Gopularam, B; Muralidara, V. N; Niranjan, N (2015): Experimental Evaluation of Network Telemetry Anonymization for Cloud Based Security Analysis, IEEE Cloud Computing for Emerging Markets, pp. 1-7
[9] Goyal. V; Pandey, O; Sahai A; Waters, B (2006): Attribute-based encryption for fine-grained access control of encrypted data, Proceedings of the 13th ACM conference on Computer and communications security. pp. 89–98
[10] Mike, C; Netflow security monitoring for dummies, Wiley publications
[11] Ostrovsky, R; Sahai, A;,Waters, B (2007): Attribute-based encryption with non-monotonic access structures, Proceedings of the 14th ACM conference on Computer and communications security, pp. 195–203
[12] Ristov, S; Gusev, M; Donevski, A (2013): OpenStack Cloud Security Vulnerabilities from Inside and Outside, Cloud Computing, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, pp. 95-101
[13] Sahai, A; Waters, B (2005): Fuzzy identity-based encryption, Advances in Cryptology–EUROCRYPT. Springer, pp. 457–473
[14] Xu, D; Luo, F; Gao, L; Tang, Z (2013): Fine-grained document sharing using attribute-based encryption in cloud servers, Innovative Computing Technology (INTECH), pp. 65-70
[15] http://docs.openstack.org/developer/fuel-docs/index.html