2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)
Bari, Italy. October 6-9, 2019

Enhancing Data Security in Cloud using Random Pattern Fragmentation and a Distributed NoSQL Database

Nelson L. Santos, Bogdan Ghita, Giovanni L. Masala

Abstract— **The cloud computing model has become very popular among users, as it has proven to be a cost-effective solution to store and process data, thanks to recent advancements in virtualization and distributed computing. Nevertheless, in the cloud environment, the user entrusts the safekeeping of its data entirely to the provider, which introduces the problem of how secure such data is and whether its integrity has been maintained. This paper proposes an approach to the data security in cloud by utilizing a random pattern fragmentation algorithm and combining it with a distributed NoSQL database. This not only increases the security of the data by storing it in different nodes and scramble all the bytes, but also allows the user to implement an alternative method of securing data. The performance of the approach is compared to other approaches, along with AES 256 encryption. Results indicate a significant performance improvement over encryption, highlighting the capabilities of this method for cloud stored data, as it creates a layer of protection without additional overhead.**

I. Introduction

The paradigm of cloud computing has been well received by different communities, as its users are able to reduce costs associated with storage, maintenance, computing power, and focus on the development [1]. Despite the many benefits brought by this technology, many threats have also emerged. Cloud data centres are increasingly becoming targets of attacks not only from outside attackers, but also malicious inside users [2]. What is more, the cloud provider is responsible for both management and safekeeping of the user data and, in most cases, does not disclose such procedures to its users [3-5]. Encryption is widely used to secure the data in the cloud, however, encryption algorithms expose data once they are compromised [6], not to mention the encryption process adds overhead, rendering this approach less appealing for data driven environments, such as big data or internet of things [7][8]. This paper approaches the problem by proposing the use of a fragmentation algorithm, combined with a distributed NoSQL (Not only SQL) database to secure data stored in the cloud. The data is fragmented into chunks, which are scrambled and stored in the database, which is also distributed across different nodes. This provides a faster alternative to secure data in the cloud and this distributed approach allows the data to be processed simultaneously, taking advantage of the high resources offered by cloud computing and thus facilitating its adoption in this environment. Scenarios suitable for the proposed method lie mainly on environments where speed is paramount and the client resources are limited. This includes mobile cloud computing, internet of things (IoT), including medical devices that compose a wireless body area network [9], as resources such as battery power, processor speed and memory capabilities, affect greatly the capabilities of the device. Another area of application for the proposed method would include backup and storage of data in public clouds, where the provider is entrusted with the safeguarding of the data, without disclosing its procedures to the client [5]. The data will reside in different nodes and, in the unlikely event the cloud gets compromised, attackers would not be able to reconstruct the data even if the attacker is able to access all the database nodes. The method proposed fits in the bitwise category, as described by [10], in which the method can be applied to any data type, increasing its usability and scenarios of application.

The paper will start by analysing the related work concerning data security on the cloud, followed by a detailed description of the proposed method. Afterwards, the proposed method will be compared to similar approaches with regards to performance. Finally, the results will be presented and discussed, to increase the awareness of the benefits and drawbacks of using alternative approaches to encryption to secure data in the cloud.

II. Related work

A. Data Anonymization

One of the many approaches evaluated by the research community to secure data in the cloud revolves around anonymization of stored data. A review of various well-known anonymization algorithms identified that K-anonymity prevents linkage between records by generating large equivalence classes; however, if records of the same class have similar values on a sensitive attribute, an attacker can identify an individual [11]. L-Diversity, although overcoming this drawback, proved to be difficult to achieve and insufficient in preventing the disclosure of attributes. To overcome this, t-closeness was proposed, however the amount of useful information that can be extracted after applying it is very limited.

A publication by Goswami and Madan [12] compared and contrasted different techniques using Map Reduce for their advantages and disadvantages. Such techniques included [13], which proposed a two-phase top-down specialization using K-

N. L. Santos is with the School of Computing, Electronics and Mathematics, University of Plymouth, Plymouth, PL4 8AA, United Kingdom (email: nelson.santos@plymouth.ac.uk)

B. Ghita is with the Centre for Security, Communications and Network Research, University of Plymouth, Plymouth, PL4 8AA, United Kingdom (email: bogdan.ghita@plymouth.ac.uk)

G. L. Masala is with the School of Computing, Mathematics and Digital Technology of the Manchester Metropolitan University, Manchester, M15 6BH, United Kingdom (email:g.masala@mmu.ac.uk)

978-1-7281-4569-3/19/$31.00 ©2019 IEEE

anonymity that used the full capability of MapReduce for data anonymization. However, according to the authors [12], it was susceptible to overhead errors due to actions such as splitting and key-value pair sorting. Another investigated method was [14], which proposed MapReduce with optimal balancing scheduling anonymization that improved the data locality problem in map reduce. Nevertheless, beyond its security issues, the method proved challenging to apply on a big data environment. The authors of [15] also proposed a top down specialization using MapReduce, including a more accuracy constraint MapReduce framework for data anonymization, but the proposed method had reduced extensibility and fault tolerance.

Furthermore, [16] replaced location coordinates with semantic categories, a technique known as semantic labelling, to achieve data anonymization; this is effective but can only be used in locations that can be mapped to semantic vocabulary. In addition, the categories needed to be decided in advanced and without the possibility of adding categories during runtime or changing existing categories in real time.

B. Encryption

When addressing the security, trust and privacy of data in cloud computing, the most common approach is the use of encryption [2]. Dahya and Rani [17] combined DES and AES using RSA to increase the protection of sensitive data (username and password) in the cloud using symmetric tokens. Similarly, [18] proposes a Hybrid Cryptographic System that combines symmetric and asymmetric encryption, along with hashing and salting techniques at various levels to protect data in the cloud. However, using such high number of encryption mechanisms affects the efficiency of the system. Furthermore, their current implementation does not support multi cloud environments or any recovery features that would prevent data loss. Potey, Dhote and Sharma [19] proposed the use of full homomorphic encryption in the cloud in order to allow users to compute their data, residing on a Dynamo DB, in a public cloud whilst encrypted. Despite this advances, homomorphic encryption algorithms, similar to symmetric algorithms, add unwanted overhead and consume vast resources. What is more, there is also the need to evolve current querying algorithms under the full homomorphic encryption scheme [19]. Correspondingly, [20] proposes a scheme for data storage by combining symmetric encryption and erasure codes.

Despite being well researched and widely used, symmetric algorithms require the exchanging of the secret key [21], which, if captured, would render the mechanism ineffective. Additionally, encryption adds unwanted complexity computing overhead, hindering therefore its use on applications with limited resources, such as mobile phones [22] or internet of things, where it would impact the CPU and memory usage, therefore dramatically reduce battery life. From a slightly different perspective, environments where speed is paramount, such as big data or real-time applications, are also affected by encryption as the client cannot run queries on encrypted data [23]. Even with the existence of homomorphic encryption, which allows encrypted data to be processed, the large key size and low calculation efficiency, hinders its practicality in cloud computing [24][25].

C. Data Fragmentation

Data fragmentation as a concept can be found in the literature as far back as the late 70s [26]. It has proven to ensure data security at much lower costs allowing multiple fragments to be accessed simultaneously by exploiting concepts in parallel computing, [23]. However, its adoption is yet to be widespread [10], as it was mainly adopted in relational databases [27, 28] and multi-cloud architectures [29]. Kapusta and Memmi [10], provide a wide range survey of different data protection mechanism using fragmentation, where the authors categorize different approaches into bitwise and structure wise fragmentations. In [30], the authors analyse the performance of different data fragmentation algorithm and contrasts with the use of encryption. The techniques include a predefined fragmentation, a random pattern fragmentation and a combination of random pattern fragmentation with AES encryption. Results from that research indicated a trade-off between performance and security and offered a range of environments where the mechanism could be applied effectively. However, the evaluated mechanisms did not provide any means of data management, not to mention the research was limited to a single instance in the cloud, creating therefore a single point of failure.

In [31], the authors propose a fragmentation and dispersal technique of cypher texts obtained using block ciphers. Similarly, [6] combined different encryption algorithms with a distribution system, which distributes a database across different clouds, based on the level of encryption applied. Bahrami and Singhal proposed a lightweight method that allows mobile clients to store .JPEG images on multiple clouds [32]. In their approach, the data is scrambled using a pseudo-random permutation based on the chaos system, but the mechanism can only be applied for jpeg image files and does not work with multiple file types. Some authors implemented a database in addition of a fragmentation technique to add more management to the data. For instance, [33] introduces a distributed MongoDB database to store the fragmented data. Similarly, [34], demonstrates a solution, where the data is randomly fragmented before being stored in a NoSQL database. However, the NoSQL database proposed by the author was hosted on a single instance, inducing therefore the problem of a single point of failure.

This work will introduce a combination of the random pattern fragmentation algorithm and an Apache Cassandra database [35], where the objective is to split the data into chunks and utilize the database not only to add management to the data, but also add a layer of security as the fragmented data stored on it, will be distributed across different nodes.

III. Methodology

As mentioned previously, this paper aims to increase the security of cloud stored data by employing data fragmentation and a distributed database. The method aims to identify an alternative data security solution for cloud computing, where the data is divided into multiple chunks and scrambled into split files. Those split files, in turn, are inserted into an Apache Cassandra database, which is distributed across multiple nodes (virtual instances in the cloud). This technique allows the data to be dispersed across and in the events a node gets

compromised, the attacker would not gain complete access to the data. Furthermore, in the unlikely event all nodes get compromised, the attacker would only be able to reconstruct the data with either the pattern key stored in the client, or using brute force, which would take considerable time to be reconstructed. Moreover, the proposed method also allows for the nodes to be stored in different cloud providers, increasing significantly the security of the data.

A. Random Pattern Fragmentation (RPF)

The random pattern fragmentation algorithm, as seen in figure 1, splits of the original file into N chunks determined by the users. The chunks are then scrambled in a random order and inserted into special files (split files) that contain metadata, of what is being stored, such as extension and size. The number of split files is also determined by the user and the chunks are serialized into arrays of raw bytes. Finally, the split files are then sent to the database, where each split file is saved as a row in the table. Unlike other related approaches, such as [32] and [36], the proposed method does not track the header and footer of the file, nor it adds padding to chunks to ensure they are all the same size. This is due to the unwanted performance overhead that both practices introduce. Rather, the proposed method relies on a combination of the metadata in the split file and the order of the pattern stored in the client machine, to determine the correct order of the chunks. It is also important to note that all communications between the client machine and the database occur via a virtual private network (VPN), encrypting therefore all the data in traffic.

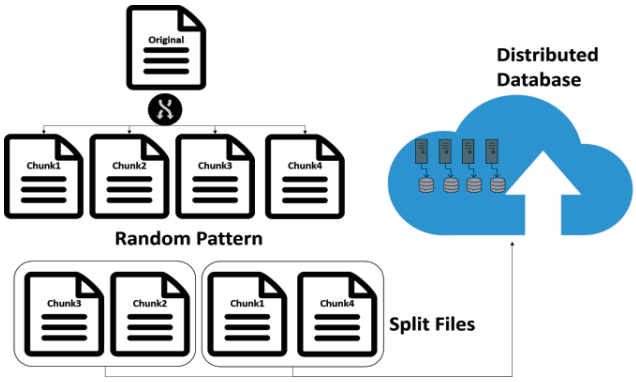

Fig.1 Proposed random pattern fragmentation algorithm during the fragmentation stage.

During the reconstruction stage, as seen in figure 2, the database is queried on the metadata held on the split file. The split files are then downloaded in the client machine, where the serialized chunks are aligned and re-organized based on the pattern stored in the client machine and the metadata that each split file contains. This process includes creating a dictionary datatype containing the unique id assigned to the chunk and the raw bytes containing the data. Once in the correct order, the chunks are then converted to a byte array and de-serialized and the original file is stored in the client device. Similar to the previous stage, all communications are secured through a VPN.

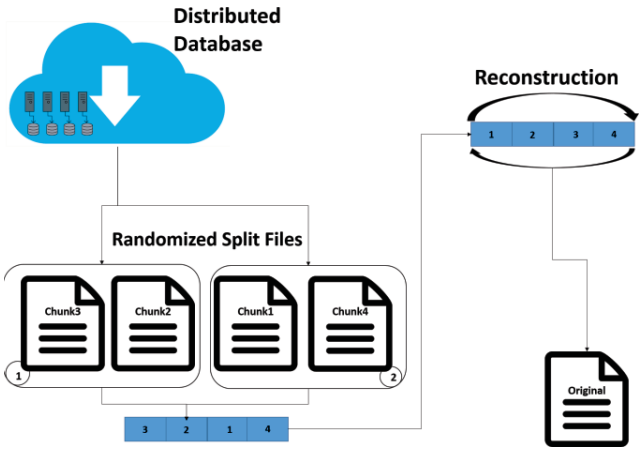

Fig 2. Proposed Random Pattern Fragmentation method in the reconstruction stage

B. Cassandra Distributed Database

Apache Cassandra is an open source NoSQL database that stores and handles large data on commodity servers, whilst maintaining its service availability high without any single point of failure [37]. Cassandra is a wide column store, which combines a key-value and tabular database management systems and consists of a number of nodes that communicate in a peer-to-peer fashion, without a master node. Within the database, distribution is performed using an internal component named *partitioner*, a hashing mechanism that computers a numerical token on the primary key of a table row, and assigns it to a node in the cluster. The database is natively distributed, allowing the addition of nodes or datacentres with minimal downtime. This built-for-scale architecture allows the database to handle large amounts of data and concurrent operations. Such factors, along with its support of multiple data types, led to Cassandra being the database of choice for this project.

In the proposed approach, the database will store the split files, which contain the chunks in raw bytes and their metadata; when the user selects the desired number of split files, the same number of tables is automatically created to store them. The process of the insertion into the database can be described as follows:

- The user describes the desired number of split files and a corresponding number of tables is created
- When the client program completes the fragmentation and has the split files ready for upload, separate threads are created to handle the insertion into the database concurrently.
- The split files are inserted and the chunks (in byte arrays) are stored as Binary Large Objects (BLOBs).

For the download of the split files, the steps would consist of:

- A query with the details of the file is created and sent to the database
- For each split file described by the client, a separate thread is created to handle the download of all the split files concurrently

- When all the files are downloaded from the database the connection is closed.

IV. Experiment and Results

For the experiment, a dataset of four datatypes (.bmp, .jpeg, .pdf and .docx) is crated, with each file containing around 100 KB in size. Despite the user having the choice of selecting different chunk sizes and number of split files, for the purpose of the experiment, all the data was gathered with 1000 byte chunks and 2 split files. A Bitnami Cassandra stack [38], was launched on a Microsoft Azure [39] infrastructure, containing three Standard D1 v2 virtual machines, with 1 VCPU and 3.5 GB of RAM, hosting Linux Ubuntu Server 16.04 LTs [40]. To secure the connection between the client program and the database nodes, a Virtual Private Network [41] is used. The tables were created in advance of the experiment, where a user was assigned two tables, representing the chosen number of split files. This method was preferred as a latency was identified when creating a new table, due to all the different components responsible for data distribution and replication are being created/synchronized across all nodes. The data stored in the table included the name of the user, the file name and extension, along with a serialized blob (Binary Large Object) of the split file, containing the randomized chunks inside. After chunks have been created, randomized, the split files are created and inserted to the database asynchronously, with each file having a unique connection to the database and accessing its corresponding table. Similarly, to reconstruct the file, the database is accessed asynchronously and the split files are obtained simultaneously, then combined and reorganized according to the pattern key stored in the client machine.

To compare the results, a program was created using the PyCryptodome [42] library to perform encryption and decryption functions in the same dataset files using the Advanced Encryption Standard (AES) with 256 bit blocks. A single encrypted file was uploaded to a virtual machine with similar specifications as a single Cassandra node, using the Secure Copy Protocol [43]. Afterwards, the same file was downloaded and decrypted by the program, with the downloaded file being saved at a directory specified by the user.

Additionally, the method proposed by [34], using an Apache CouchDB database [44] with random pattern fragmentation was also analysed, given the similarities to the proposed approach, when compared to other methods in the literature review. In this approach, the file was fragmented, shuffled and inserted into split files. Each document represented a split file and it was uploaded accordingly. In the reconstructing stage, the documents were retrieved and the split file rebuilt into chunks and rearranged accordingly before being stored back into the local machine. Finally, a single file was uploaded in a virtual machine with similar specifications, using the SCP protocol. This would represent the average time to send and receive a file from the cloud, without applying any techniques.

The experiment will consider the total time taken from processing the initial file, along with fragmenting, inserting to the database, downloading and reconstructing back the original file (latency). External factors such as the fluctuations on the network or CPU cycles will be considered as deviation on the calculations. In contrast, the latency from processing the file, encryption, sending, receiving and decrypting will be considered for the comparison program. The client device used had and Intel Core i7-5650U CPU with 16GB of RAM, running on a 64-bit Windows 10 operating system.

Initial results, shown in figure 3 and table 1, highlight the significant improvement in latency when using the proposed method versus its counterparts. The average latency of the proposed method was around 0.56 seconds. In contrast, the approach using CouchDB averaged 1.57 seconds, whilst AES mean latency is 1.6 seconds. The single file upload averaged 1.44 seconds across all data types. The latency does not seem to vary between file types, with standard deviation values being 0.02 for Cassandra, 0.01 for the CouchDB and 0.03 for both the AES encryption and the single file upload. It is important to notice that for this experiment, as explained earlier, the time taken to create tables for an individual user was not taken into account, as they were created in advance. In fact, the user can submit many files during the session, and this added latency is only counted at the beginning of the session and not for each file sent. Nevertheless, it can be seen on table 1 that on average, the database takes 0.70 seconds to create both tables that store the split files.

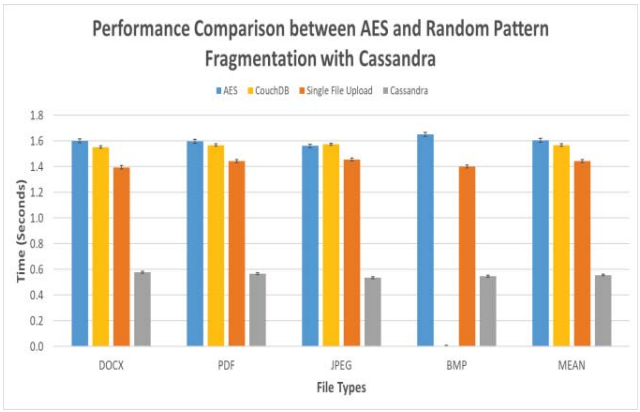

Fig.3 Performance comparison of proposed methods with other approaches

Table 1 Detailed performance comparison of all methods

File Type	Cassandra (s)	Couch DB (s)	AES (s)	Single File (s)	Chunk Len. (s)	Table Creation (Cassandra) (s)
DOCX	0.57	1.55	1.596	1.39	1000	0.72
PDF	0.56	1.56	1.561	1.44	1000	0.69
JPEG	0.53	1.57	1.651	1.45	1000	0.73
BMP	0.54	N/A	1.601	N/A	N/A	0.67
MEAN	0.56	1.57	1.60	1.44	1000	0.70
St. Dev.	*0.02*	*0.01*	*0.03*	*0.03*	1000	0.02

A significant improvement in performance comes from the underlying workflow. Given that the file is fragmented,

processing is done simultaneously on the split files, either in the client machine or the database. Such asynchronous behaviour allows different components to be processed quicker and more efficiently, unlike encryption, where the processing is sequential on a block-by-block basis. Moreover, in [34], despite some asynchronous methods being used, the database was hosted on a single server, not only restricting the available resources, but also increasing the risk of total data loss, as this architecture would represent a single point of failure. Cassandra's distributed architecture also features a data replication technique that spans across the cluster, which allows data to be easily recovered, in case a node encounters any problems.

The proposed method does not aim to replace encryption. Rather, fragmenting the data and concurrently sending the fragments into the cloud, provides an alternative to securing the data in the cloud in a more bespoke manner. As seen in table 2, random pattern fragmentation provides enough security without consuming many resources, making it ideal for usage in environments such as mobile phones, Internet of Things, or big data, where the devices possess very limited resources and performance is paramount.

Table 2 Performance and security comparison of all methods

Method	Security	Performance	Suitability
RPF + Cassandra	Med	High	Mobile, Big Data, IoT
RPF + CouchDB	Med	Med	Mobile, Big Data, IoT
AES	High	Low	High Security Environments

V. Conclusion

Cloud data security, privacy and trust has become a crucial issue that impacts the success of this paradigm. Traditional encryption mechanisms are not suited for the task of protecting data in the cloud, as the nature of unstructured vast volume of data, along with the exponential increase on demand for fast access to the data, increase the latency and add overhead to the processing of the data. Similarly, data anonymization techniques also proved to add unwanted overhead and, in some scenarios, proved insufficient to fully preserve the privacy of an individual. We have proposed a method that combines random pattern fragmentation with a wide-column NoSQL database. Current results indicate a higher performance when compared to its counterparts, which implies the usability of the proposed method in cloud computing, especially in scenarios with high speed needs and limited resources. A drawback in the current system lies in the management of the user tables in the database. The number of split files is predefined at the beginning and further changes are not allowed at runtime. This would allow the user to quickly assess the security level of the data and further distribute or split the data when needed. In addition, further improvements would need to be done to increase the usability of the proposed system in environments that need constant access to the data, or real-time access. Additionally, one ought to also account for the processing overheads brought by the proposed method, such as deploying configurations and starting the VPNs, as well as acquiring access through firewalls. Future work will also include the introduction of additional mechanisms of data recovery and further tests with bigger datasets and an environment encompassing different cloud providers. With cloud computing rapidly increasing and the users become more security-conscious, having a vast array of possibilities to secure the data not only deters attacks from occurring, but also drives the evolution of such technologies further.

VI. References

[1] M. Bahrami and M. Singhal, "The Role of Cloud Computing Architecture in Big Data", in *Information Granularity, Big Data, and Computational Intelligence*, 8th ed., Pedrycz and S. Chen, Ed. Springer, 2015, pp. 275-295.

[2] Z. Yan, R. Deng and V. Varadharajan, "Cryptography and Data Security in Cloud Computing", *Information Sciences*, vol. 387, pp. 53-55, 2017.

[3] Cloud Security Alliance, "Top Threats to Cloud Computing", CSA, 2010.

[4] P. Kumar, P. Raj and P. Jelciana, "Exploring Data Security Issues and Solutions in Cloud Computing", *Procedia Computer Science*, vol. 125, pp. 691-697, 2018.

[5] R. Hegarty and J. Haggerty, "Extrusion detection of illegal files in cloud-based systems", *International Journal of Space-Based and Situated Computing*, vol. 5, no. 3, p. 150, 2015.

[6] A. Alsirhani, P. Bodorik and S. Sampalli, "Improving Database Security in Cloud Computing by Fragmentation of Data", *2017 International Conference on Computer and Applications (ICCA)*, C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.

[7] G. Manogaran, C. Thota and M. Kumar, "MetaCloudDataStorage Architecture for Big Data Security in Cloud Computing", *Procedia Computer Science*, vol. 87, pp. 128-133, 2016.

[8] M. Potey, C. Dhote and D. Sharma, "Homomorphic Encryption for Security of Cloud Data", *Procedia Computer Science*, vol. 79, pp. 175-181, 2016.

[9] M. Li, W. Lou and K. Ren, "Data security and privacy in wireless body area networks", *IEEE Wireless Communications*, vol. 17, no. 1, pp. 51-58, 2010.

[10] K. Kapusta and G. Memmi, "Data protection by means of fragmentation in various different distributed storage systems - a survey", *arXiv:1706.05960v1*, 2017. [Accessed 14 March 2019].

[11] K. Parmar and V. Shah, "A Review on Data Anonymization in Privacy Preserving Data Mining", *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 5, no. 2, pp.75-79,2016.Available: https://www.ijarcce.com/upload/2016/february16/IJARCCE%2016.pdf. [Accessed 12 April 2019].

[12] P. Goswami and S. Madan, "Privacy preserving data publishing and data anonymization approaches: A review", *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 2017.

[13] Z. Priyanka, K. Nagaraju and Y. Venkateswarlu, "Data Anonymization Using Map Reduce on Cloud based A Scalable Two-Phase Top-Down Specialization", *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 2, no. 12, pp. 3879-3883, 2014. [Accessed 18 April 2019].

[14] R. Sreedhar and D. Umamaheshwari, "Big-Data Processing With Privacy Preserving Map-Reduce Cloud", *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 3, no. 1, pp. 343-350, 2014. [Accessed 18 April 2019].

[15] M. Balusamy and S. Muthusundari, "Data anonymization through generalization using map reduce on cloud", *Proceedings of IEEE International Conference on Computer Communication and Systems ICCCS14*, 2014.

[16] O. Barak, G. Cohen and E. Toch, "Anonymizing mobility data using semantic cloaking", *Pervasive and Mobile Computing*, vol. 28, pp. 102-112, 2016.

[17] N. Dahiya and S. Rani, "implementing multilevel data security in cloud computing", *International Journal of Advanced Research in Computer Science*, vol. 48, no. 8, pp. 146-152, 2017.

[18] A. Arora, A. Khann, A. Rastogi and A. Argarwal, "Cloud security ecosystem for data security and privacy", in *2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence*, Noida, India, 2017.

[19] M. Potey, C. Dhote and D. Sharma, "Homomorphic Encryption for Security of Cloud Data", *Procedia Computer Science*, vol. 79, pp. 175-181, 2016

[20] R. Wang, "Research on Data Security Technology Based on Cloud Storage", *Procedia Engineering*, vol. 174, pp. 1340-1355, 2017.

[21] A. Bhardwaj, G. Subrahmanyam, V. Avasthi and H. Sastry, "Security Algorithms for Cloud Computing", *Procedia Computer Science*, vol. 85, pp. 535-542, 2016.

[22] M. Bahrami and M. Singhal, "A dynamic cloud computing platform for eHealth systems", *2015 17th International Conference on E-health Networking, Application & Services (HealthCom)*, 2015.

[23] H. Dev, T. Sen, M. Basak and M. Ali, "An Approach to Protect the Privacy of Cloud Data from Data Mining Based Attacks", in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Salt Lake City, 2012, pp. 1106-1115.

[24] X. Song and Y. Wang, "Homomorphic cloud computing scheme based on hybrid homomorphic encryption", *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, 2017.

[25] Z. Mahmood and M. Ibrahem, "New Fully Homomorphic Encryption Scheme Based on Multistage Partial Homomorphic Encryption Applied in Cloud Computing", *2018 1st Annual International Conference on Information and Sciences (AiCIS)*, 2018

[26] A. Shamir, "How to share a secret", *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, 1979.

[27] T. Hong and J. Ren, "Fragmentation Storage Model: An Efficient Privacy Protection Technology", *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, 2017.

[28] G. Aggarwal et al., "Two can keep a secret: A distributed architecture for secure database services", in *Proc. CIDR*, 2005.

[29] J. Bohli, N. Gruschka, M. Jensen, L. Iacono and N. Marnau, "Security and Privacy-Enhancing Multicloud Architectures", *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 4, pp. 212-224, 2013.

[30] N. Santos, S. Lentini, E. Grosso, B. Ghita and G. Masala, "Performance Analysis of Data Fragmentation Techniques on a Cloud Server"in Proc, *International Journal of Grid and Utility Computing*.

[31] K. Kapusta and G. Memmi, "Enhancing Data Protection in a Distributed Storage Environment Using Structure-Wise Fragmentation and Dispersal of Encrypted Data", *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018.

[32] M. Bahrami and M. Singhal, "A Light-Weight Permutation Based Method for Data Privacy in Mobile Cloud Computing", in *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, San Francisco, CA, 2015, pp. 189-198.

[33] G. Masala, P. Riu and E. Grosso, "Biometric Authentication and Data Security in Cloud Computing", in *Computer and Network Security Essentials*, K. Daimi, Ed. Detroit: Springer, 2018, pp. 337-353.

[34] N. Santos and G. Masala, "Big Data Security on Cloud Servers", in *11th International KES Conference on Intelligent Interactive Multimedia: Systems & Services*, Goad Coast, 2018.

[35] "Apache Cassandra", *Cassandra.apache.org*, 2018. [Online]. Available: http://cassandra.apache.org/. [Accessed: 16- Dec- 2018].

[36] S. Lentini, E. Grosso and G. Masala, "A Comparison of Data Fragmentation Techniques in Cloud Servers", *Advances in Internet, Data & Web Technologies*, pp. 560-571, 2018.

[37] DataStax Academy, "What is Apache Cassandra", *DataStax Academy*, 2019. [Online]. Available: https://academy.datastax.com/planet-cassandra/what-is-apache-cassandra. [Accessed: 04- Apr- 2019].

[38] "Bitnami Cassandra Stack for Microsoft Azure", *Docs.bitnami.com*, 2019.[Online].Available:https://docs.bitnami.com/azure/infrastructure/cassandra/. [Accessed: 04- Apr- 2019].

[39] Microsoft Inc, "Microsoft Azure Cloud Computing Platform & Services", *Azure.microsoft.com*, 2018. [Online]. Available: https://azure.microsoft.com/en-gb/. [Accessed: 08- Dec- 2018].

[40] "The leading operating system for PCs, IoT devices, servers and the cloud|Ubuntu", *Ubuntu.com*,2019.[Online].Available:https://www.ubuntu.com/. [Accessed: 04- Apr- 2019].

[41] J. Lawas, A. Vivero and A. Sharma, "Network performance evaluation of VPN protocols (SSTP and IKEv2)", *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN)*, 2016. Available: 10.1109/wocn.2016.7759880 [Accessed 4 April 2019].

[42] "AES — PyCryptodome 3.8.1 documentation", *PyCryptodome*, 2019. [Online].Available:https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html. [Accessed: 13- Apr- 2019].

[43] CISCO, "Secure Copy", 2011.

[44] "Apache CouchDB", *Couchdb.apache.org*, 2019. [Online]. Available: http://couchdb.apache.org/. [Accessed: 18- Apr- 2019].