Access Control Mechanism for CouchDB

Ashwaq A. Al-otaibi [1†], **Reem M. Alotaibi** [1†] and **Nermin Hamza**[2††]
aalotaibi0553@stu.kau.edu.sa ralotibi@kau.edu.sa nermin.hamza@cu.edu.eg
[1†] Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
[2††] Faculty of Graduate Studies of Statistical Research, Cairo University, Cairo, Egypt

Summary
Recently, big data applications need another database different from the Relation database. NoSQL databases are used to save and handle massive amounts of data. NoSQL databases have many advantages over traditional databases like flexibility, efficiently processing data, scalability, and dynamic schemas. Most of the current applications are based on the web, and the size of data is in increasing. NoSQL databases are expected to be used on a more and large scale in the future. However, NoSQL suffers from many security issues, and one of them is access control. Many recent applications need Fine-Grained Access control (FGAC). The integration of the NoSQL databases with FGAC will increase their usability in various fields. It will offer customized data protection levels and enhance security in NoSQL databases. There are different NoSQL database models, and a document-based database is one type of them. In this research, we choose the CouchDB NoSQL document database and develop an access control mechanism that works at a fain-grained level. The proposed mechanism uses role-based access control of CouchDB and restricts read access to work at the document level. The experiment shows that our mechanism effectively works at the document level in CouchDB with good execution time.

Keywords:
Big data; Security issues; Access control; Fine-Grained Access control (FGAC); NoSQL database.

1. Introduction

The increasing amount of data has created enormous challenges for obtaining, storing, managing, and analysing data [1]. The term big data refers to data with 3V. They are high volume, velocity, and variety [2]. It is not possible to process big data using traditional data processing platforms [3].

Google and Facebook are among the companies that have discovered that relational databases are unable to handle big data and achieve the requirements of their users [3]. As a result, a new database called NoSQL reads as "Not Only SQL." The NoSQL provides data model flexibility, scalability, dynamic schemas, and efficient processing big data. It can process structure, semi-structured and unstructured data like documents, e-mail, and social media effectively. It utilizes distributed devices for data storage and retrieval.

Nowadays, organizations moved from Relation Database Systems (RDBS) towards NoSQL databases because of the increasing use of cloud computing and big data [3]. Compared with different databases, a NoSQL database is a good choice for big data applications.

The NoSQL databases based have been arranged on their popularity [4]. The top NoSQL databases are MongoDB, Hypertable, CouchDB, Redis, and Cassandra.

Despite the popularity of NoSQL databases and their advantages, these systems have critical security problems [5]. Weak data protection is the most important issue applied in the current mechanisms [6]. In data protection, access control is the fundamental requirement of any database system [5]. Access control is one of the main security utilities currently provided by RDBMSs and NoSQL databases [7]. Authorize users to access only a portion of the entire database called granular security [8].

Most NoSQL databases implement standard access control techniques at the level of coarse-grained. Several document-based databases allow users to access the entire database or nothing at all [9]. Improving levels of data protection in systems can increase their growth and usability. As big data platforms frequently handle sensitive data, access control mechanisms should work at the finest granularity levels [10].

Enforcement of a Fine-Grained Access Control (FGAC) into databases that store user personal data can be very useful [4]. FGAC is an essential requirement in many applications. NoSQL databases are classified into four models, which are key-value, column-based, document-based, and graph-based database [11]. A Few NoSQL

Manuscript received December 5, 2022
Manuscript revised December 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.12.14

databases provide native support of FGAC, such as Accumulo, which is a key-value database that implements an access control mechanism at the cell level. However, most NoSQL databases did not implement FGAC.

This paper proposes an access control mechanism that implements read access at the document level. The proposed mechanism works on Role-Based Access Control (RBAC) of CouchDB. It is used within a design document of the specified database. The proposed mechanism aims to allow authorized users to access specific documents within the database. The document access depends on specific permission rules. We evaluate the proposed mechanism using UNECE's Country Overview dataset. In particular, the main contributions of this paper are as follows:

- Propose an access control mechanism and implement authorization at a fine-grained level in the CouchDB database.
- Work on Role-Based Access Control (RBAC) of CouchDB.
- Implement read access to work at the document level.

The paper is organized as follows. Section II provides a review of existing access control models in different NoSQL databases. Section III and IV present the proposed access control mechanism and the experimental set-up. Finally, Section V shows the conclusion of the paper.

2. NOSQL Databases Access Control

Lack of access control mechanisms is one of the main security problems in some NoSQL databases.Vonitsanos et al. discussed the main security concerns in NoSQL databases. Access control and data protection can be considered as some of the major security issues in these databases [12].

Dadapeer et al. [3] analysed the security characteristics of MongoDB, Cassandra, CouchDB, HBase, and Redis. Authorization mechanisms differ in these databases. In general, NoSQL databases implement ineffective authorization techniques. Many of them did not perform authorization on a fine-grained level. The current section will present some research studies on NoSQL Database access control.

Kulkarni [13] proposed a fine-grained access control model for Cassandra and HBase. The proposed model works at different levels, such as a row, column, or column family level. However, the model cannot be operated with other databases.

Huang et al. [14] propose an Attribute-Based Fine-Grained Access Control (AGAC) technique, which supports five granularity levels and users' atomic operations. The AGAC mechanism aims to enhance the access control capacity of HBase. Also, it provides flexibility in the authorization. However, developing the access control mechanism of other NoSQL database types remains a huge challenge.

For graph-based databases, Morgado et al. [15] developed a model that implements access control for these databases. The model utilizes metadata and provides Data Definition Language (DDL) and Data Manipulation Language (DML) operations. The aim is to permit various applications to perform their access control model in case of using the graph-oriented database. It was applied to the Neo4j database. The model was able to prevent unauthorized access. However, the model should be executed in the core of Neo4j, and its performance evaluated to determine its feasibility. Also, the access control mechanism needs to be expanded to operate at finer granularity level.

To explore the possibility of implement granular security in the graph-based database, Crawford [16] utilizes graph concepts in the Neo4j database to discover a mechanism that allows access to data, whereas preserving the security. The mechanism depends on mathematical formulas to locate two-hop connections that go out and return to the network security layer. The user can detect these

connections without violating the security set by the security layer.

For document-based databases, a document-based database stores data as documents. Several research studies have focused on MongoDB. Colombo and Ferrari [17] suggested the combination of a purpose-based model into MongoDB. They design an enforcement monitor that operates as a proxy between the server and clients. The proxy monitors the stream of messages between clients and the server and may alter it. It can operate with any MongoDB deployment. The proposed model worked on the Role-Based Access Control (RBAC) [18] of MongoDB and supported a purpose-based policy. It implements access control at the document level. However, the model is specific to MongoDB. It should be generalized to work with other NoSQL databases. Furthermore, there is a need to improve the model to operate on the field level.

To design an access control mechanism for MongoDB that operates on the field level. The authors [19] improved the RBAC model of MongoDB to support both content and context-based access control policies. They developed a monitor that was determined to work with any MongoDB deployment. The monitor is a Wire protocol interpreter that works as a proxy. It monitors and may change messages between MongoDB clients and the server. However, the monitor cannot work for all types of queries in the same effective way. The overhead is low for find and
map-reduce queries, whereas it is significant for aggregate queries.

Longstaff and Noble [20] proposed an Attribute-Based Access Control (ABAC) [21] model for big data applications. The proposed model can work with a relational database, Hadoop, and NoSQL database. It depends on modifying queries and integrates the ABAC into the user transaction code. However, the mechanism specific for SQL queries. The SQL cannot process a variety of data from NoSQL databases.

To integrate ABAC into databases, Colombo and Ferrari [22] proposed a mechanism that implements various ABAC policies for documents on field level. The proposed mechanism rewrites SQL++ queries and works with each document-based database that provides SQL++. However, a tool is needed to simplify specifying policies and binding process. Furthermore, the model only works with the NoSQL databases that support SQL++.

In general, previous studies in document-based databases have almost exclusively concentrated on MongoDB. CouchDB is one of the most common NoSQL databases [23]. Fig 1 show the structure of CouchDB database. It is a document-based database that allows authorized users to read all documents in the database. The read access works on the database level. Each document is uniquely named and contains one or more fields. The database system maintains the documents metadata. Fields inside the document consist of values of different forms (number, Boolean, text, lists).

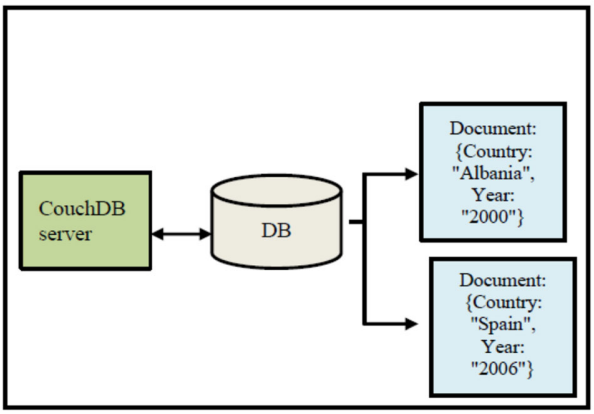

Figure 1: CouchDB database structure

For updating (edit, add, delete) and reading database documents, CouchDB supports a RESTful HTTP API by using various request forms such as put, get, post, head, copy, and delete. CouchDB data structure is a JavaScript Object Notation (JSON) object.

The CouchDB server has many databases. Each one consists of its own authorization rules. These rules determine which users are permitted to write and read documents, modify specific database configuration parameters, and create design documents. The server admin can set up the authorization rules and can modify them at any time. Database authorization rules set a user into one of two categories:

- Members who can access all documents, create documents, and alter any document unless design documents.
- Admins who can write and read all kinds of documents, alter which users are admins or users, and set specific configuration options for each database. The actions of a database admin are restricted to a specific database.

CouchDB provides a function to restrict document updates, which is called validate_doc_update. This function can be used to deny unauthorized requests to update documents. In CouchDB, no mechanism restricts access to documents. The members can access all documents in the database.

In this work, we propose an access control mechanism for CouchDB to restrict the document read access. The mechanism uses RBAC that is supported by CouchDB, and it implements access control at the document level. It depends on the user role and the role field created in the document. When a user requests a document, only allow access to that document if the roles are the same.

3. The Proposed Mechanism

There are two different scenarios for the proposed access control mechanism, and we will explain the admin scenario and then the user scenario.

3.1 Admin Scenario

The database admin assigns roles to the users, adds the field "role" in every document stored in the database, and creates an access control component in the specified database. Fig.2 shows the admin scenario of the proposed mechanism, and Fig3. illustrates the documents inside CouchDB database with role field.

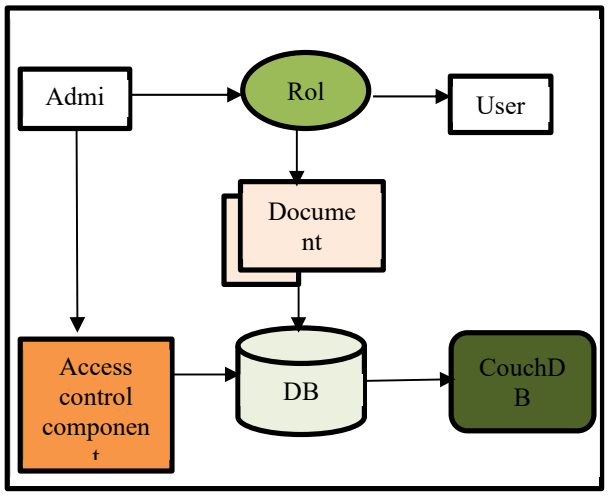

Figure 2: Admin scenario of the proposed access control mechanism

The access control component consists of a design document that contains a show function. The show function provided by CouchDB is used to represent documents in various formats. Also, it can be used to operate server-side functions without the need for a pre-existing document.

In this work, we use it to run the AccessbyRole function that implements access control at the document level by only allow users whose roles match the role field to access the requested document.

The proposed mechanism used the access control component when the user sends a get request for a document. Request a document using the show function work in this format:

GET/mydb/_design/mydesign/_show/myshow/doc_id

However, we want to request the document directly from the database with a simple GET request. Rewrites feature used to rewrite the specified path according to rules determined in the specific design document.

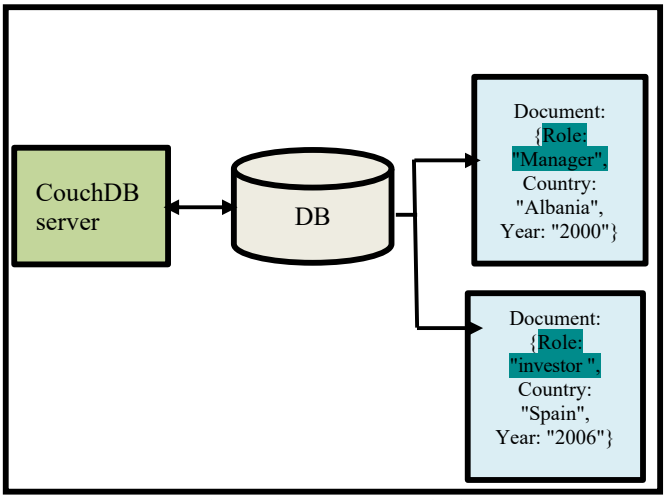

Figure 3: CouchDB documents with the proposed mechanism

Listing 1: shows the pseudocode of the access control component.

Listing 1 Access control component

```
{"_id": "_design/role",

"shows": {

"AccessbyRole": "function(doc, req){\r\n if (doc.role ==
req.userCtx.roles ) {\r\n   return toJSON(doc)  } else \r\n      {\r\n
return \"unauthorized to access this document\";\r\n    \r\n}}"

},

"language": "javascript",

"rewrites": [

{

"from": "dataset/*",

"to": "../../_design/role/_show/AccessbyRole/*",

"method": "GET"}]}
```

The rules are defined in a string, or array field called rewrites within the design document.
This combination of features within the design document resource means that the application can be deployed without disclosing the full CouchDB API to the users by using a virtual host.
CouchDB can assigns requests to various locations based on the Host header. This permits different virtual hosts on the same device to map to various design documents or databases.
The most prevalent use case is to provide entire control over the application's URIs by mapping a virtual host to a rewrite handler. To insert a virtual host, it is enough to edit the host's file and add an entry. In this work, we add a virtual host called localdb in the windows host file instead of the localhost of CouchDB.

We work in a virtual host named localdb. To set a virtual host configuration, we work on default.ini and local.ini, which are CouchDB configuration files. In the default.ini file, we create a section called Virtual Host (vhost) preceded by Cross-Origin Resource Sharing (Cors). Then, we edit the vhosts section in the local.ini and add localdb to it. Now, all GET document requests to the database will be redirected to the proposed mechanism.

3.2 User Scenario

When the user wants to access a specific document, she/he sends a GET request with a document id to the CouchDB. The CouchDB uses an access control component that exists in the database to check whether the user is authorized to access the requested document or not.
Every user has a role, and if the role field of the requested document is the same as the user's role, the document is returned to the user. Otherwise, the user is unauthorized to access this document. Fig.4 shows the user scenario.

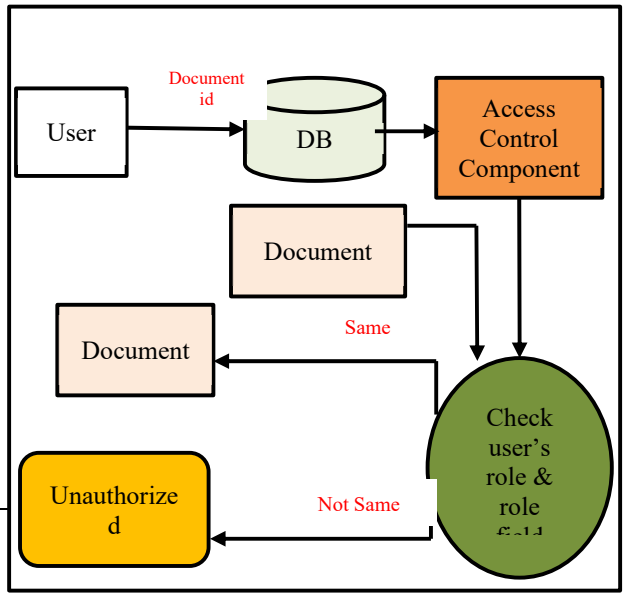

Figure 4: User scenario of the proposed access control mechanism

4. Experimental Setup

To evaluate the proposed mechanism, the test was performed with document requests for different users to evaluate the performance of the mechanism and its execution time.

We followed several steps to test the efficiency of the proposed mechanism, which are:

1. First, the admin creates users, roles, and assigns roles to the users.

2. Second, the admin adds role fields to every document.

3. Third, we specify which user will send the document's Get request.

4. Fourth, we specify the execution time for each request.

4.1 Dataset Description

To evaluate the proposed model, we made use of UNECE's Country Overview, which is a public dataset that includes 884 documents with 79 fields. This dataset was retrieved in the JSON-stat format using the bulk document insert feature provided by CouchDB. The admin adds a role field to each document in the dataset.

4.2 Result Analysis

To evaluate the performance of the proposed mechanism, we tested the validity of the mechanism and discussed its execution time. We conducted two different test cases to validate our mechanism (authorized access and unauthorized access).

To implement the mechanism, we created two users with different roles. Alice, with a manager role, and Bob with an investor role. The user can access the document if the user's role matches the document's role.

4.2.1 Authorized Access Case

In our mechanism, the user is authorized to access the required document when the user's role matches the role field of the requested document. For example, Alice can access any document where the role field is a manager.

The figures below show the result of applying the mechanism. Fig.5 shows the document with the role field "Manger" and Fig.6 shows the result when the user "Alice" sent a Get request for this document. Alice has a Manager role, which is the same role as the requested document. CouchDB returns the requested document to the user.

Figure 5: Document with the role field "Manger."

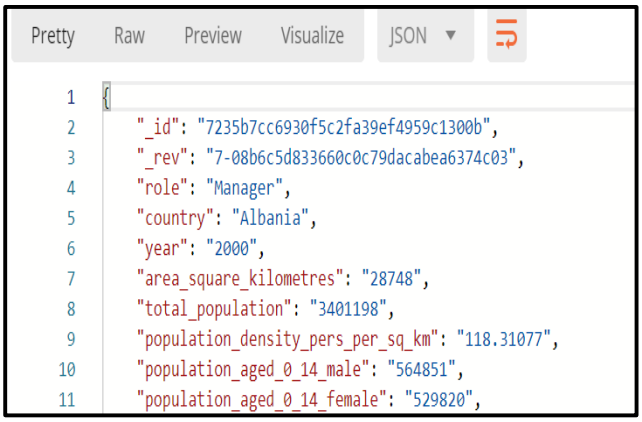

Figure 6: Result of an authorized request.

4.2.2 Unauthorized Access Case

In our mechanism, access to documents is unauthorized when the roles are not the same. Fig.7 shows the result of the unauthorized Get request. Bob, who has an "investor" role, tried to access a document with the "Manager" role field. By using the proposed mechanism, CouchDB returns "unauthorized to access this document."

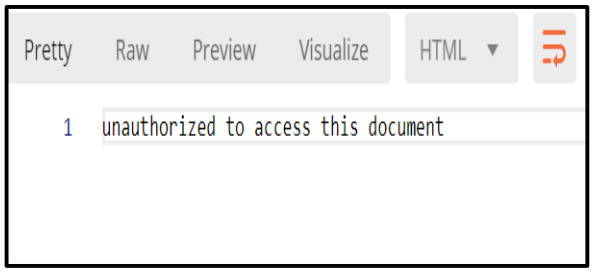

Figure 7: Result of an unauthorized request.

4.2.3 Execution Time

In this subsection, we will compare the execution time of the Get request using the proposed mechanism and without it. The execution time of the Get request is the sum of prepare, socket initialization, Transmission Control Protocol (TCP), Domain Name System (DNS) lookup, handshake, transfer start, download, and process.

First, we sent the GET request for a document without using the mechanism. As shown in Fig.8, the execution time was about 20ms. In the second case, we sent a Get request for a document using the proposed mechanism.

Fig.9 shows the execution time was about 22ms for an unauthorized GET request, while Fig.10 shows the execution time for the authorized GET request was about 21ms.

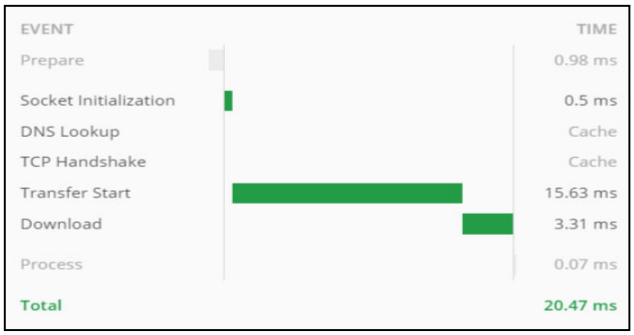

Figure 8: The execution time of a Get request without using the mechanism.

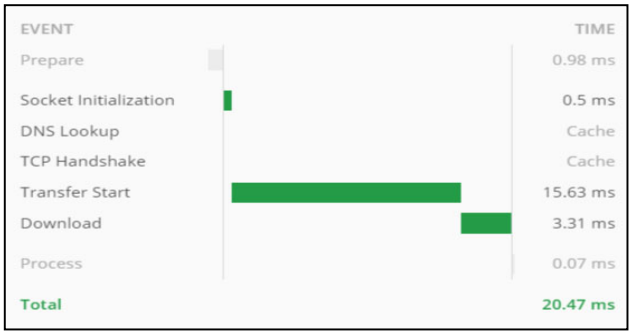

Figure 9: The execution time of an unauthorized Get request using the mechanism

From the above figures, the execution time of a Get request without using the proposed mechanism is close to the execution time with using it. The proposed mechanism did not affect the database's performance. However, the proposed mechanism does not work with mango query including find and index. Moreover, it does not support other NoSQL databases.

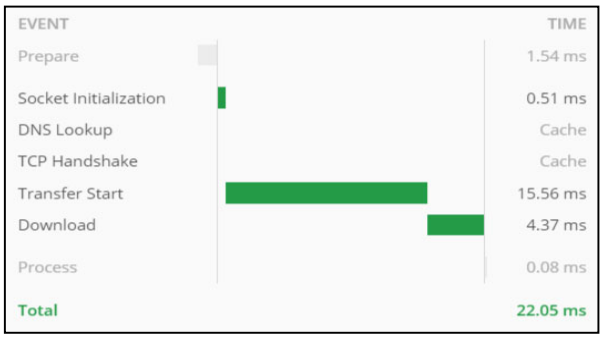

Figure 10: The execution time of an authorized Get request using the mechanism.

5 Conclusion

Recent developments in big data and cloud computing have created the need to store a huge amount of data in databases that offer high scalability and availability. This has prompted companies to switch from relational databases toward the NoSQL database. NoSQL gained extensive market interest and became a popular database choice. It can process huge volumes of unstructured, structured, semi-structured data.

NoSQL provides flexibility, scalability, better performance, and often open source in the schema. However, it does not have a standard query language and lacks the appropriate security techniques. As large amounts of sensitive user data stored in NoSQL databases, the security provided by these systems has become a growing concern. NoSQL databases need effective data protection.

Access control and data protection are among the important security issues. Access control is the primary part of data protection for any DBMS. The majority of NoSQL systems implement basic access control techniques that work on a coarse-grained level. However, FGAC is the main requirement in many data management systems and applications. Implement FGAC in the NoSQL databases is a new research topic recently covered.

Acknowledgment

This work was supported by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under grant No. (DG-044-612-1440). The authors, therefore, gratefully acknowledge the DSR technical and financial support.

Conflicts of Interest

We have NO affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

[1] M. Chen, S. Mao and Y. Liu, "Big data: A survey," *Mobile networks and applications*, vol.19, no. 2, pp. 171-209, 2014.

[2] E. Sahafizadeh and M. A. Nematbakhsh, "A survey on security issues in Big Data and NoSQL," *Advances in Computer Science: an International Journal*, vol. 4, no. 4, pp. 68-72, 2015.

[3] N. Dadapeer, G. Adarsh and M. Indravasan, "A Survey on Security of NoSQL Databases," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 4, no. 4, pp. 5250-5254, 2016.

[4] P. Noiumkar and T. Chomsiri, "A comparison the level of security on top 5 open source NoSQL databases," *The 9th International Conference on Information Technology and Applications (ICITA2014)*, 2014.

[5] P. Colombo and E. Ferrari, "Fine-Grained Access Control Within NoSQL Document-Oriented Datastores," *Data Science and Engineering*, vol. 1, no. 3, pp. 127-138, 2016.

[6] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes and J. Abramov, "Security issues in nosql databases," *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pp. 541-547, 2011.

[7] P. Colombo and E. Ferrari, "Evaluating the effects of access control policies within NoSQL systems," *Future Generation Computer Systems*, vol. 114, pp. 491-505, 2021.

[8] S. Rizvi, A. Mendelzon, S. Sudarshan and P. Roy, "Extending query rewriting techniques for fine-grained access control," *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp 551-562, 2004.

[9] A. Alotaibi, R. Alotaibi and N. Hamza, "Access Control Models in NoSQL Databases: An Overview," *journal of king abdulaziz university computing and information technology sciences*, vol. 8, no. 1, pp: 1 – 9, 2019.

[10] P. Colombo and E. Ferrari, "Access control in the era of big data: State of the art and research directions," *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, pp: 185–192, 2018

[11] Fidels Cybersecurity, "Current Data Security Issues of NoSQL Databases", 2014.

[12] G. Vonitsanos, E. Dritsas, A. Kanavos, P. Mylonas and S. Sioutas, "Security and Privacy Solutions associated with NoSQL Data Stores," 2020 IEEE *15th International Workshop on Semantic and Social Media Adaptation and Personalization (SMA),* pp. 1-5*,* 2020.

[13] D. Kulkarni, "A fine-grained access control model for key-value systems," *Proceedings of the third ACM conference on Data and application security and privacy*, pp. 161-164, 2013.

[14] L. Huang, Y. Zhu, X. Wang and F. Khurshid, "An Attribute-Based Fine-Grained Access Control Mechanism for HBase," *International Conference on Database and Expert Systems Applications*, Springer, pp. 44-59, 2019.

[15] C. Morgado, G. Busichia Baioco, T. Basso and R. Moraes, "A Security Model for Access Control in Graph-Oriented Databases," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Lisbon, pp. 135-142, 2018.

[16] B. Crawford, "Granular security in a graph database," Master's thesis, Naval Postgraduate School Monterey United States, 2017.

[17] P. Colombo and E. Ferrari, "Enhancing MongoDB with purpose based access control," *IEEE Transactions on Dependable and Secure Computing*, 2015.

[18] D. Ferraiolo, R. Sandhu, S. Gavrilla and R. Kuhn, "Proposed NIST standard for role based access control," *ACM Transactions on Information and System Security (TISSEC),* vol.4, no.3, pp.224–274, 2001.

[19] P. Colombo and E. Ferrari, "Towards virtual private NoSQL datastores," in Data Engineering (ICDE), 2016 IEEE 32nd International Conference on, pp. 193-204, 2016.

[20] J. Longstaff and J. Noble, "Attribute based access control for big data applications by query modification," *Big Data Computing Service and Applications (BigDataService)*, 2016 IEEE

Second International Conference on, pp. 58-65, 2016.

[21] V. Hu, D. Kuhn and D. Ferraiolo, "Attribute-based access control," *Computer,* vol. 48, no. 2, pp. 85-88, 2015.

[22] P. Colombo and E. Ferrari, "Towards a unifying attribute based access control approach for NoSQL datastores," *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pp. 709-720, 2017.

[23] A. CouchDB, Documentation,", URL: https://docs.couchdb.org/en/stable/, 2017.