International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887*
*Volume 5 Issue VII, July 2017- Available at www.ijraset.com*

# Data Management and Privacy Preservation in Mongo DB

Kusum Kakwani[1], Naziya Pathan[2] , Shyam Dubey[3], I.C. Mehta[4]
[1]*M. Tech. Scholar,* [2,3] *Asstt. Prof., Department of CSE., Nuva College of Engineering and Technology, Nagpur (MS), India.*
[4]*Associate Prof., Department of Computer Technology, MIET, Gondia (MS), India.*

***Abstract:*** ***Space, Time and Privacy have become a key requirement for data management systems. NoSQL datastores, namely highly compress data on non-relational database management systems, which often support data management of Internet applications, still do not provide support for privacy policies. With this work, we begin to solve this issue, by proposing an approach for securing data using Role-based policy capabilities into MongoDB, one of the most popular NoSQL datastore. It consists of the enhancement of the MongoDB Role based access control model with privacy keys for security and monitor. The proposed monitor is easily used into any MongoDB deployment through simple methods. The results show that our monitor uses Role-based access key control which helps to reduce the computation overhead and improves security by providing privacy preserving data access policies.***
***Keywords:*** ***Role-based access control, Privacy, No SQL, data stores, Mongo DB.***

## I. INTRODUCTION

NoSQL datastores are emerging non-relational databases committed to provide high security for database operations over several servers. These platforms are getting increasing attention by companies and organizations for the ease and efficiency of handling high volumes of heterogeneous and even unstructured data. Although NoSQL datastores can handle high volumes of personal and sensitive information, up to now the majority of these systems provide poor privacy and security protection. Initial research contributions started studying these issues, but they have mainly targeted security aspects. To the best of our knowledge, we are not aware of any work targeting privacy-aware access control for NoSQL systems, but we believe that, similar to what has been proposed for relational DBMSs, privacy-aware access control is urgency for NoSQL datastores as well. However, different from relational databases, where all existent systems refer to the same data model and query language, NoSQL datastores operate with various languages and data models. This variety makes the definition of a general approach to have of privacy-aware access control into NoSQL datastores a very important goal. We believe that a stepwise approach is necessary to define such a general solution. As such, in this, we start focusing on: 1) a single datastore, and 2) selected rules for privacy policies. We approach the problem by focusing on MongoDB, which, according to the DB-Engines Ranking, 2 ranks, by far, as the most popular NoSQL datastore. MongoDB uses a document-oriented data model. Data are modelled as documents, namely records, possibly images collections that are stored into a database. Several privacy-aware access control models proposed for relational DBMSs are analysed to identify the characteristics of privacy policies to be supported. In all the analysed models; privacy policies require rule based and enforcement mechanisms, as different data owners can have different privacy requirements on their data. The purpose for which data is stored and the reason for its access is considered as the key required condition to grant the access [3]. Thus it is important for any privacy policy. As such, fine grained Role-based policies have been selected as the target policy type for our proposal.

MongoDB integrates a role-based access control (RBAC) model which supports user and role management, and enforces access control at collection level. However, no support is provided for privacy based policies. As such, in this work we extend MongoDB RBAC with the support for privacy policy specification and enforcement at document level. More precisely, the rule level at which the MongoDB RBAC model operates, integrating the required support for purpose related concepts. On top of this enhanced model we have developed an efficient enforcement monitor, called Mem (MongoDB enforcement monitor), which has been designed to operate in any MongoDB deployment. Within the client/server architecture of a MongoDB deployment, a MongoDB server front-end interacts, through message exchange, with multiple MongoDB clients. Mem operates as a proxy in between a MongoDB server and its clients, monitoring and possibly altering the flow of messages that are exchanged by the counterparts [5].

Access control is enforced by means of MongoDB message rewriting. More precisely, either Mem simply forwards the intercepted message to the respective destination, or injects additional messages that encode commands or queries. In case the intercepted message encodes a query, Mem writes it in such a way that it can only access documents for which the specified policies are

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887*
*Volume 5 Issue VII, July 2017- Available at www.ijraset.com*

satisfied. The integration of Mem into a MongoDB deployment is straightforward and only requires a simple configuration. No programming activity is required to system administrators. Additionally, Mem has been designed to operate with any MongoDB driver and different MongoDB versions. First experiments conducted on a MongoDB dataset of realistic size have shown a low Mem enforcement overhead which has never compromised query usability.

## II. ANALYSIS AND DESIGN

MongoDB databases reside on a MongoDB server that can host more than one of such databases which are independent and stored separately by the MongoDB server. A database contains one or more collections consisting of documents. Data in MongoDB has a flexible schema. Unlike SQL databases, where one must determine and declare a table's schema before inserting data, MongoDB collections do not enforce document structure. This flexibility facilitates the mapping of documents to an entity or an object. Once the first document is inserted into a database, a collection is created automatically and the inserted document is added to this collection. Such an implicitly created collection gets configured with default parameters by MongoDB. ObjectId, a special 12-byte BSON type guarantees uniqueness within the collection. The ObjectId is generated based on timestamp, machine ID, process ID, and a process-local incremental counter. MongoDB uses ObjectId values as the default values for _id fields.

GridFS is the file system layer built on top of MongoDB. GridFS is the specification for chunking files. Instead of storing a large file in a single document, it is chunked into smaller files and each of them is stored as separate document. GridFS stores these in two collections. One collection stores the chunks and other stores the metadata related to the file. Hence the metadata of the file is tightly coupled with the data file. When a query is made to the GridFS stored file, the driver or client will reassemble the chunks.

Access control is a fundamental security technique in systems in which multiple users share access to common resources. It is the process of stating and enforcing security in MongoDB. MongoDB's comprehensive security framework features:

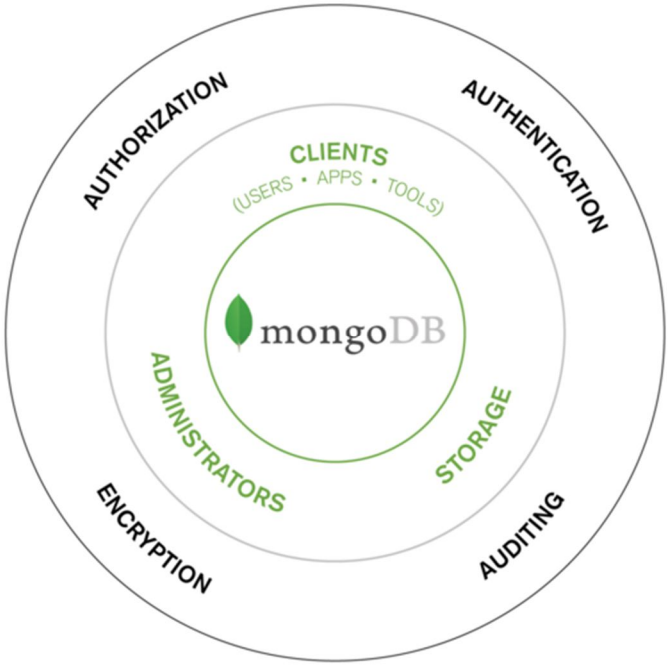


Figure 1. Security Model of MongoDB

### *A. Authentication*

Authentication is the process of verifying the identity of a client. When access control, i.e. authorization, is enabled, MongoDB requires all clients to authenticate themselves in order to determine their access.

### *B. Authorization*

User-defined roles means you can configure granular permissions for a user or application, based on the privileges they need to do their job. It determines the verified user's access to resources and operations.

*C. Auditing*

A native audit log lets you track access and operations performed on the database which works for regulatory compliance.

*D. Encryption*

MongoDB data can be encrypted on the network, on disk and in backups. With the Encrypted storage engine, protection of data at-rest is an integral feature within the database. By natively encrypting database files on disk, administrators eliminate both the management and performance overhead of external encryption mechanisms. Only those staff who have the appropriate database authorization credentials can access the encrypted data, providing additional levels of defense.

The RBAC model of MongoDB does not operate at a granularity level which is suited to privacy policy specification and enforcement. Collections represent the finest granularity level at which RBAC is enforced; however, this granularity is not adequate in quite common application scenarios where documents refer to multiple subjects. We aim at addressing these MongoDB RBAC shortcomings through the definition of Mem, a monitor supporting the efficient enforcement of privacy and role based access control at document level within MongoDB platforms. The MongoDB RBAC model is characterized by the concepts of privilege, data resource, action, role and user. When a role is assigned to a user, the user receives the authorization to execute all the actions specified by the privileges associated with the role on the specified data resources. We have enhanced the MongoDB RBAC model with additional conceptual elements, instrumental to support Role-based access control. The key element of the enhanced model is the concept of privacy preservation as described in figure 2. The monitor encrypts the file and uploads it in MongoDB. The encrypted key is generated for each file being uploaded. When the end users request the file access from MongoDB, it finds the end users' requested file key and decrypts all related data sets and provides the original files to the end users after verification.

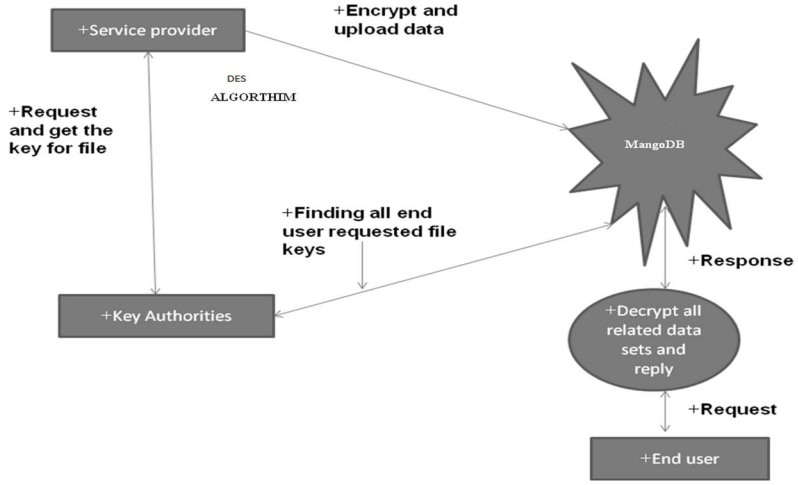


Figure 2. Data Encryption and key Generation

The general approach to the rule of privacy-aware access control into NoSQL data stores is a very important goal. Users are only allowed to execute for access purposes for which they have a proper authorization. Purpose authorizations are granted to users as well as to roles. The data storage and network transfer format for documents, simple and fast. Most regulatory requirements mandate that the encryption keys must be rotated and replaced with a new key at least once annually. MongoDB can achieve key rotation without incurring downtime by performing rolling restarts of the replica set. When using appliance, the database files themselves do not need to be re-encrypted, thereby avoiding the significant performance overhead imposed by key rotation in other databases. Only the master key is rotated, and the internal database key store is re-encrypted.

## III. IMPLEMENTATION DETAILS

Mongo DB stores its data in BSON. BSON is a lightweight binary format capable of representing any MongoDB document as a string of bytes. The database understands BSON, and BSON is the format in which documents are saved to disk. When a driver is given a document to insert, use as a query, and so on, it will encode that document to BSON before sending it to the server. Likewise, documents being returned to the client from the server are sent as BSON strings. This BSON data is decoded by the driver to its native document representation before being returned to the client. The BSON format has three primary goals:

*A. Efficiency*

BSON is designed to represent data efficiently, without using much extra space.

*B. Traversability*

In some cases, BSON does sacrifice space efficiency to make the format easier to traverse.

*C. Performance*

Finally, BSON is designed to be fast to encode to and decode from.

Triple DES (3DES) is the common name for the Triple Data Encryption Algorithm (TDEA or Triple DEA) symmetric-key block cipher, which applies the Data Encryption Standard (DES) cipher algorithm three times to each data block. The original DES cipher's key size of 56 bits was generally sufficient when that algorithm was designed, but the availability of increasing computational power made brute-force attacks feasible. Triple DES provides a relatively simple method of increasing the key size of DES to protect against such attacks, without the need to design a completely new block cipher algorithm.

Triple DES uses a "key bundle" that comprises three DES keys, $K_1$, $K_2$ and $K_3$, each of 56 bits (excluding parity bits). The encryption algorithm is: ciphertext = $E_{K3}(D_{K2}(E_{K1}(\text{plaintext})))$

i.e.DES encrypt with $K_1$, DES decrypt with $K_2$, then DES encrypt with $K_3$.

Decryption is the reverse: plaintext = $D_{K1}(E_{K2}(D_{K3}(\text{ciphertext})))$

I.e., decrypt with $K_3$, encrypt with $K_2$, then decrypt with $K_1$.

Each triple encryption encrypts one block of 64 bits of data.

In each case the middle operation is the reverse of the first and last. This improves the strength of the algorithm when using keying option 2, and provides backward compatibility with DES with keying option 3.

The standards define three keying options:

1) *Keying option 1:* All three keys are independent.
2) *Keying option 2:* $K_1$ and $K_2$ are independent, and $K_3 = K_1$.
3) *Keying option 3*: All three keys are identical, i.e. $K_1 = K_2 = K_3$.

The MD5 hash function was originally designed for use as a secure cryptographic hash algorithm for authenticating digital signatures. MD5 has been deprecated for uses other than as a non-cryptographic checksum to verify data integrity and detect unintentional data corruption. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit 'fingerprint' or 'message digest' of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be 'compressed' in a secure manner before being encrypted with a private (secret) key. Sometimes we have to serialize objects, e.g. to send them over a network, store and restore them locally or for any other reason. Now it can be useful if we would know after the deserializing process the object has been restored correctly. A possible solution to this problem is to provide a MD5 - Hash String so the receiver can compare if the file has been transmitted without any modifications.

As this paper primarily focuses on privacy so it is necessary to enable access control and specify the authentication mechanism. Authentication requires that all clients and servers provide valid credentials before they can connect to the system. Create a user administrator first and then create additional users. Create a unique MongoDB user for each person and application that accesses the system. Create roles that define the exact access a set of users needs. Follow a principle of least privilege. Then create users and assign them only the roles they need to perform their operations. A user can be a person or a client application. To protect data, documents are encrypted. Encryption can be implemented at the application level, or via external file system and disk encryption solutions. With the introduction of the Encrypted storage engine in MongoDB 3.2, protection of data at-rest becomes an integral feature of the database. By natively encrypting database files on disk, administrators eliminate both the management and performance overhead of external encryption mechanisms. This new storage engine provides an additional level of defence, allowing only those staff with the appropriate database credentials access to encrypted data. Using the Encrypted storage engine, the raw database content, referred to as plaintext, is encrypted using an algorithm that takes a random encryption key as input and generates cipher text that can only be read if decrypted with the decryption key. The process is entirely transparent to the application. MongoDB supports a variety of encryption schema. The storage engine encrypts each database with a separate key. The key-

wrapping scheme in MongoDB wraps all of the individual internal database keys with one external master key for each server. In the proposed monitor, Plaintext of the text format files is converted into cipher text using 64 bit 3DES algorithm and 128 bit MD5 algorithm is used to generate the encrypted key for each file which is generated in a single Icrypto Package. The encrypted files get stored in MongoDB.

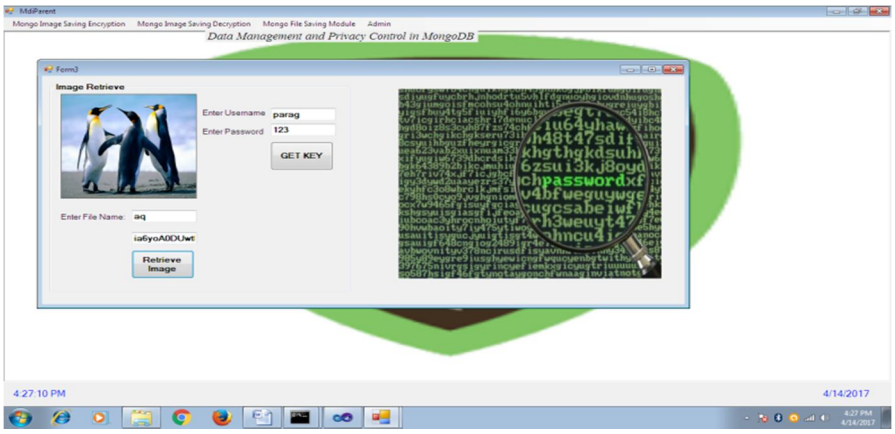


Figure 3. Image Retrieval using Encrypted Key

Figure 3 shows the form used to access image from MongoDB. When user wants to retrieve the image from the MongoDB, he needs to give the username, filename and file password. The system verifies all this data and gives the encrypted key only after authentication. After providing the encrypted key for decryption, user can retrieve the original image from MongoDB. The administrator has complete access over the system. The administrator can access all users' information like filenames, passwords and privacy keys as shown in figure 4.

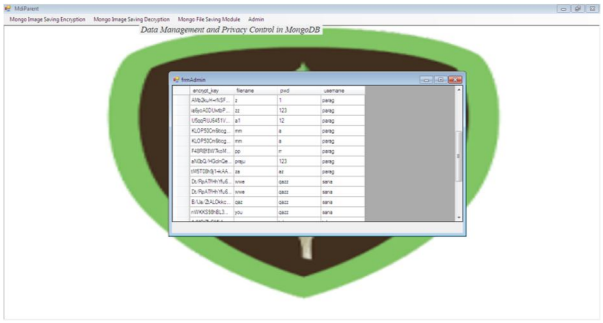


Figure 4. Privacy Key Management

## IV. RESULT ANALYSIS

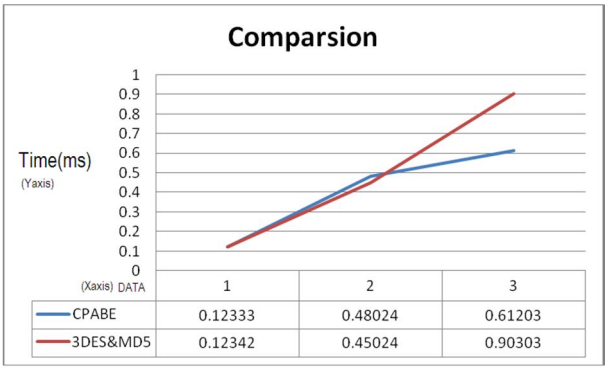


Figure 5. Comparison between Ciphertext-Policy Attribute-Based Encryption (CP-ABE) and 3DES with MD5

CPABE has single layer encryption whereas the hybrid 3DES and MD5 has higher security. CPABE cannot handle the larger data, its conversion ratio falls down as the time lapse. Triple DES attempts to solve this by using three keys and three cipher operations handles very large data efficiently, this requires very less time. 3DES is easy to implement and accelerate in both hardware and software. The proposed crypto hybrid algorithm using 3DES with Hash MD5 encrypted Key secures the larger data with high level protection system.

## V. CONCLUSION

This work presents the secure Role-based concepts and gives mechanisms to regulate the access at document level on the basis of role and key based policies. An enforcement monitor called Mem (MongoDB enforcement monitor) has been designed to implement the proposed security. It operates as a proxy between MongoDB user and a MongoDB server, and enforces access control by monitoring and possibly manipulating the flow of exchanged messages between them. Furthermore, the presented approach can be generalized to support for multiple NoSQL datastores. Mem incorporates the support for context sensitive to introduce access control and contribute towards a much needed privacy framework in the relatively new NoSQL database.

## REFERENCES

[1] Parinaz Ameri ,Jorg Meyer and Achim Streit “On a New Approach to the Index Selection Problem using Mining Algorithms” 2015 IEEE International Conference on Big Data (Big Data
[2] H. Ulusoy, P. Colombo, E. Ferrari, M. Kantarcioglu, and E. Pattuk. GuardMR: fine-grained security policy enforcement for MapReduce systems. In ACM ASIACCS 2015
[3] P. Colombo and E. Ferrari. Efficient enforcement of action aware purpose-based access control within relational database management systems. IEEE Transactions on Knowledge and Data Engineering, 27(8), 2015
[4] D. Kulkarni. A fine-grained access control model for key-value systems. In Proceedings of the third ACM conference on Data and application security and privacy, pages 161–164. ACM, 2013
[5] P. Colombo and E. Ferrari. Enhancing MongoDB with Purpose Based Access Control. In IEEE Transactions on Dependable and Secure Computing, November 2015 DOI:10.1109/TDSC.2015.2497680
[6] P. Colombo and E. Ferrari. Privacy aware access control for big data: A research roadmap. Big Data Research, 2015. http://dx.doi.org/10.1016/j.bdr.2015.08.001.
[7] P. Colombo and E. Ferrari. Fine-Grained Access Control within NoSQL Document Oriented Datastores. Data Sci. Eng. (2016) 1(3):127–138 DOI 10.1007/s41019-016-0015-z.
[8] P. Colombo and E. Ferrari. Enforcing obligations within relational database management systems. IEEE Transactions on Dependable and Secure Computing (TDSC), 11(4), 2014. ]
[9] J. Byun and N. Li. Purpose based access control for privacy protection in relational database systems.The VLDB Journal, 17(4), 2008.
[10] Y. Guo, L. Zhang, F. Lin, and X. Li. A solution for privacy preserving data manipulation and query on NoSQL database. Journal of Computers, 8(6):1427–1432, 2013.
[11] R. Lutz, P. Ameri, T. Latzko, and J. Meyer, “Management of Meteorological Mass Data with MongoDB,” in 28th International Conference on Informatics for Environmental Protection: ICT for Energy Efficiency, EnviroInfo 2014, Oldenburg, Germany, September 10-12, 014.,J.M.Gomez, M. Sonnenschein, U. Vogel, A. Winter, B. Rapp, and N. Giesen, Eds. BIS-Verlag, 2014, pp. 549–556.